ANALYSE SYNTAXIQUE DESCENDANTE

ESI – École nationale Supérieure en Informatique

Analyseur syntaxique descendant

- Etude des analyseurs syntaxiques descendants déterministes.
- Analyse LL et analyse par descente récursive.
- Le fonctionnement général d'un analyseur descendant débute par l'empilement de l'axiome de la grammaire et l'utilisation de plusieurs dérivations pour essayer d'aboutir à l'entrée à analyser.

Analyseur syntaxique descendant

- Pour qu'une analyse syntaxique descendante puisse être menée, la grammaire du langage doit vérifier certaines conditions spécifiques.
- Si ces conditions ne sont pas vérifiées, la grammaire devra être transformée pour se conformer à ces exigences.

Analyse syntaxique LL

- **LL**(**k**) : **L**eft to right scanning using the **L**eftmost derivation taking decision by reading **k** tokens from the input stream.
- La traduction de l'anglais conduit à la définition suivante : "*analyser le flot d'entrée de gauche à droite en utilisant la dérivation la plus à gauche après lecture de k items du flot d'entrée*".

Analyse syntaxique LL

- Le nombre k est dans la majorité des cas égal à 1, car pour si k > 1 l'analyse devient moins intéressante.
- On se focalisera sur l'analyse LL(1) et on abordera l'analyse LL(k) de manière sommaire.

Analyse syntaxique LL(1)
Ensembles DEBUT et SUIVANT

- **<u>Ensemble DEBUT :</u>**
- Soit G une grammaire non contextuelle G=<N,T,S,P> et X un symbole de G, alors :
- DEBUT(X) = $\{a \mid X \Rightarrow^* a.\alpha,\ a \in T \text{ et } \alpha \in (T \cup N)^*\}$.
- Si $X \Rightarrow^* \varepsilon$ alors $\varepsilon \in$ DEBUT(X)

Analyse syntaxique LL(1)
Ensembles DEBUT et SUIVANT

- **DEBUT(X)**
- Si X est un terminal alors DEBUT(X) = X ;
- Si $X \rightarrow \varepsilon$ alors ajouter ε à DEBUT(X) ;
- Si $X \rightarrow Y_1 Y_2 \ldots Y_n$
- Si $\varepsilon \notin$ DEBUT(Y_1) on ajoute DEBUT(Y_1) à DEBUT(X) ;
- Si $\varepsilon \in$ DEBUT(Y_1), …, DEBUT(Y_{i-1}) avec $2 \leq i \leq n$
- Alors ajouter DEBUT(Y_1), …, DEBUT(Y_i) ε exclu à DEBUT(X)
- Si $\varepsilon \in$ DEBUT(Y_1), DEBUT(Y_2), …, DEBUT(Y_n)
- Alors ajouter ε à DEBUT(X)

Analyse syntaxique LL(1)
Ensembles DEBUT et SUIVANT

- <u>**Remarque:**</u>
- Ces règles sont appliquées jusqu'à ce qu'aucun terminal ni ε ne puisse être ajouté aux ensembles DEBUT.

Analyse syntaxique LL(1)
Ensembles DEBUT et SUIVANT

- **<u>Ensemble SUIVANT :</u>**
- Soit G une grammaire non contextuelle G=<N,T,S,P> et X un non-terminal appartenant à N, alors :
- SUIVANT(X) = $\{a \mid S \Rightarrow^* \alpha X a \beta,\ a \in T \cup \{\#\}$ et $\alpha, \beta \in (T \cup N)^*\}$.

Analyse syntaxique LL(1)
Ensembles DEBUT et SUIVANT

- **SUIVANT(X)**
  - Mettre # (qui est le marqueur de fin de l'entrée à analyser) dans SUIVANT(S) où S est l'axiome de la grammaire ;
  - S'il y a une production $A \rightarrow \alpha X \beta$ et $X \in N$
  - Alors ajouter DEBUT(β) sauf ε à SUIVANT(X) ;
  - S'il y a une production $A \rightarrow \alpha X$
  - ou une production $A \rightarrow \alpha X \beta$ avec $\varepsilon \in$ DEBUT(β)
  - Alors ajouter SUIVANT(A) à SUIVANT(X) ;

Analyse syntaxique LL(1)
Ensembles DEBUT et SUIVANT

- **<u>Remarque:</u>**
- Ces règles sont appliquées jusqu'à ce qu'aucun terminal (y compris #) ne puisse être ajouté aux ensembles SUIVANT.

Analyse syntaxique LL(1)
Ensembles DEBUT et SUIVANT

- <u>**Exemple 1 :**</u>
- Calculer les ensembles DEBUT et SUIVANT pour les non-terminaux de la grammaire dont les productions sont données ci-après :
  - $S \rightarrow aSBA \mid \varepsilon$
  - $A \rightarrow aSb \mid b$
  - $B \rightarrow bB \mid \varepsilon$

Analyse syntaxique LL(1)
Ensembles DEBUT et SUIVANT

- **Exemple 2 :**
- Calculer les ensembles DEBUT et SUIVANT pour les non-terminaux de la grammaire dont les productions sont données ci-après :
  - $S \rightarrow ABSb \mid \varepsilon$
  - $A \rightarrow aBb \mid b$
  - $B \rightarrow bB \mid cS \mid \varepsilon$

Analyse syntaxique LL(1)
Grammaire LL(1)

- Une grammaire G = <N,T,S,P> est LL(1) si et seulement si pour toute paire de règles $A \to \alpha \mid \beta$ les conditions suivantes s'appliquent :
  - Pour tout terminal a, α et β ne se dérivent toutes les deux en des chaînes commençant par a.
  - Une des chaînes au plus α ou β se dérive en la chaîne vide.
  - Si $\beta \Rightarrow^* \varepsilon$, α ne se dérive pas par un terminal de SUIVANT(A) et vice-versa.

Analyse syntaxique LL(1)
Grammaire LL(1)

- **<u>Remarque :</u>**
- Les trois conditions précédentes peuvent être résumées par la formule ci-après.
- Une grammaire G = <N,T,S,P> est LL(1) si et seulement si pour toute paire de règles $A \rightarrow \alpha \mid \beta$ on a :

  $DEBUT(\alpha.SUIVANT(A)) \cap DEBUT(\beta.SUIVANT(A)) = \emptyset$

Analyse syntaxique LL(1)
Table d'analyse LL(1)

- Pour chaque production $A \rightarrow \alpha$, procéder aux étapes suivantes :

i. Pour chaque terminal a dans DEBUT(α)
   - Ajouter la règle $A \rightarrow \alpha$ à M[A,a] ;

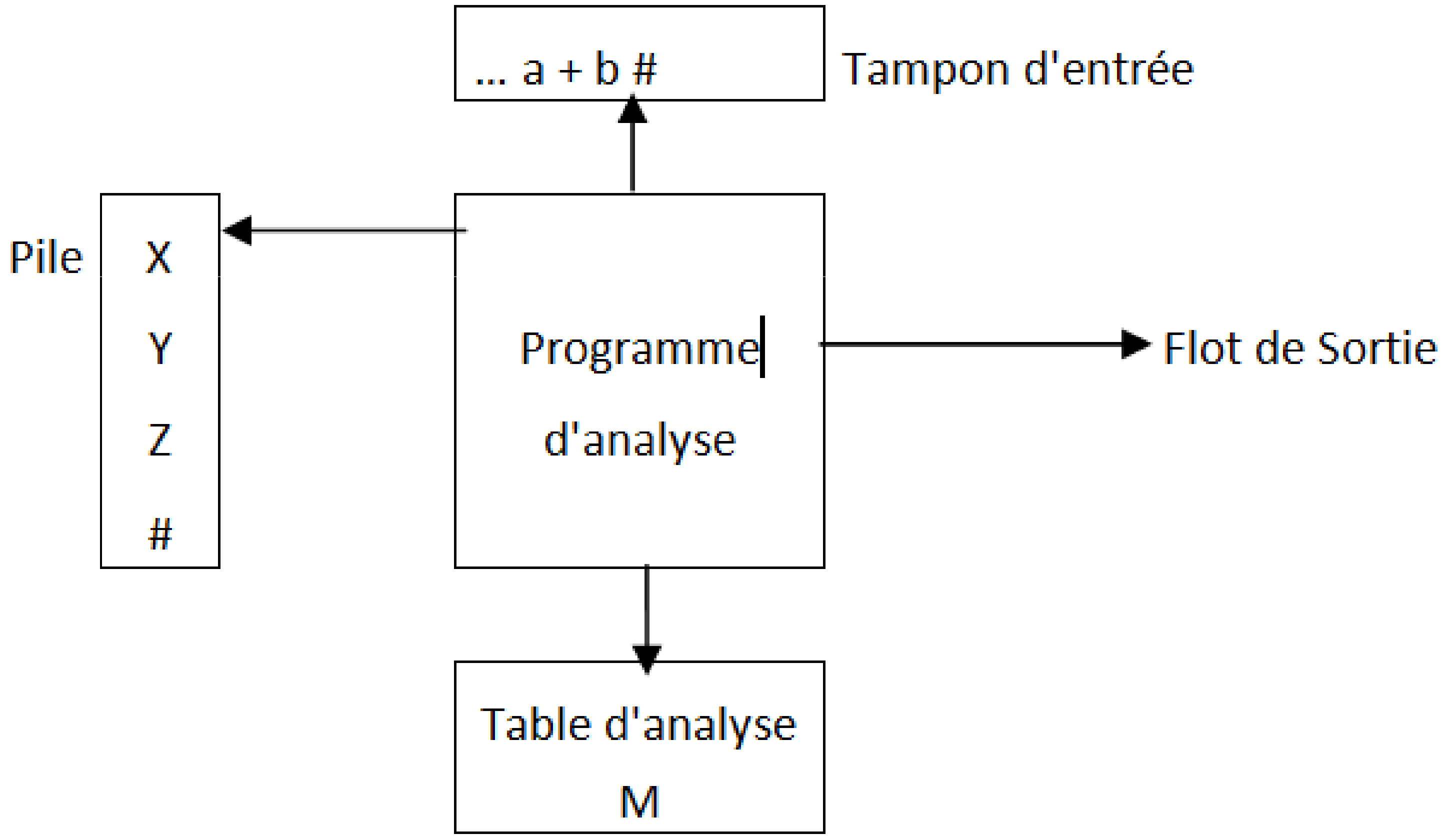
ii. Si $\varepsilon \in$ DEBUT(α)
   - Alors ajouter la règle $A \rightarrow \alpha$ à M[A,b] pour $b \in$ SUIVANT (A);

- Faire de chaque entrée non définie "une erreur".

Analyse syntaxique LL(1)
Grammaire LL(1)

- **<u>Propriétés :</u>**
- Une grammaire vérifie les conditions LL(1) si et seulement si sa table d'analyse est mono-définie (i.e. chaque entrée de la table contient au plus une règle de production de la grammaire).
- Si une grammaire G est LL(1) alors G est non ambiguë.
- Une grammaire G LL(1) permet de faire une analyse syntaxique descendante sans retour arrière.

Analyse syntaxique LL(1)
Analyseur LL(1)



Analyse syntaxique LL(1)
Analyseur LL(1)

- Algorithme d’analyse simple.
- Explicité par :
  - Empiler (#) ; Empiler (axiome de la grammaire) ;
  - Soit X le symbole en sommet de pile et a le symbole d'entrée courant ;
  - Si X = a = #, l'analyseur s'arrête et annonce la réussite finale de l'analyse ;
  - Si (X est un terminal ≠ a), l'analyseur s'arrête et signale une erreur ;
  - Si (X = a), l'analyseur enlève X de la pile et avance le pointeur du flot d'entrée;

Analyse syntaxique LL(1)
Analyseur LL(1)

- Algorithme (suite) :
  - **Si** X est un non-terminal
  - **Alors** consulter l'entrée de la table M[X,a]
    - **Si** M[X,a] = $\{A \rightarrow \alpha\}$
    - **Alors** Dépiler (X) ;
    - Empiler les symboles de α de droite à gauche
    - **FinSi**
    - **Si** M[X,a] = "erreur"
    - **Alors** l'analyseur s'arrête et signale sur erreur.
    - **FinSi**

Analyse syntaxique LL(1)
Exemple

- $E \rightarrow T\ E'$
- $E' \rightarrow +\ T\ E' \mid \varepsilon$
- $T \rightarrow F\ T'$
- $T' \rightarrow *\ F\ T' \mid \varepsilon$
- $F \rightarrow (E) \mid id$

	DEBUT	SUIVANT
E	(id	#)
E'	+ ε	#)
T	(id	+ #)
T'	* ε	+ #)
F	(id	* + #)

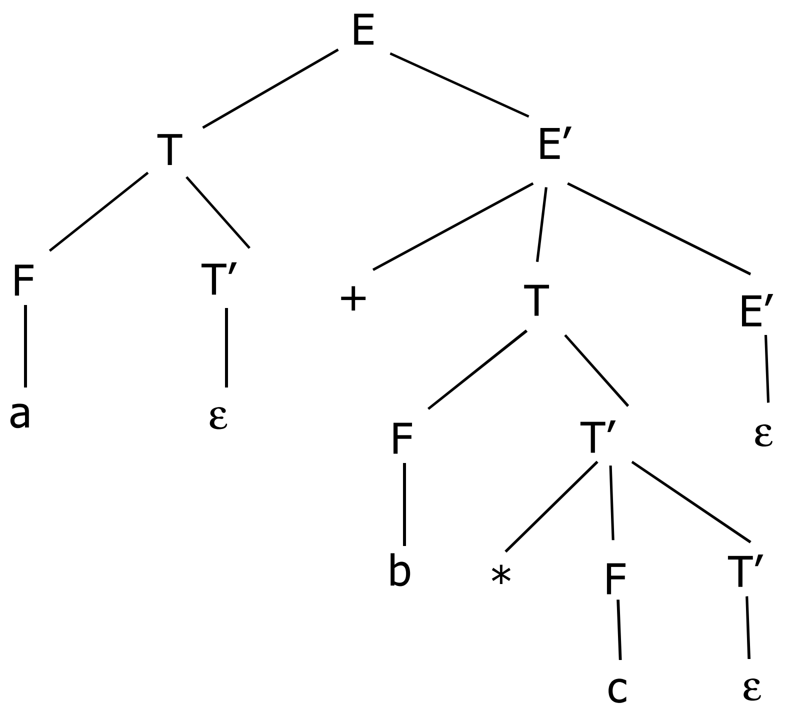
Analyse syntaxique LL(1)
Exemple

	+	*	(	)	id	#
E			E → T E'		E → T E'	
E'	E' → + T E'			E' → ε		E' → ε
T			T → F T'		T → F T'	
T'	T' → ε	T' → * F T'		T' → ε		T' → ε
F			F → (E)		F → id	

Analyse syntaxique LL(1)
Exemple Analyse de la chaîne a + b * c

Contenu Pile (sommet pile à droite)	Restant de chaîne à analyser	Action
# S	id + id * id #	Remplacer S par E' T
# E' T	id + id * id #	Remplacer T par T' F
# E' T' F	id + id * id #	Remplacer F par id
# E' T' id	id + id * id #	avancer
# E' T'	+ id * id #	Dépiler T'
# E'	+ id * id #	Remplacer E' par E' T +
# E' T +	+ id * id #	avancer
# E' T	id * id #	Remplacer T par T' F
# E' T' F	id * id #	Remplacer F par id
# E' T' id	id * id #	avancer
# E' T'	* id #	Remplacer T' par T' F *
# E' T' F *	* id #	avancer
# E' T' F	id #	Remplacer F par id
# E' T' id	id #	avancer
# E' T'	#	Dépiler T'
# E'	#	Dépiler E'
#	#	Chaîne acceptée

Analyse syntaxique LL(1)
Exemple Analyse de la chaîne a + b * c



Analyse syntaxique LL(1)
Récupération sur erreur

- La récupération sur erreur est la tentative de poursuivre l'analyse syntaxique si une erreur se produit et ne pas s'arrêter à la première erreur détectée.
- On détecte une erreur au cours d'une analyse LL(1) lorsque :
  - le terminal en sommet de pile ne correspond pas au symbole d'entrée courant ou,
  - un non terminal A est en sommet de pile, le symbole d'entrée est a et M[A,a] est vide.

Analyse syntaxique LL(1)
Récupération sur erreur en mode panique

- La récupération des erreurs en mode panique est fondée sur l'idée de sauter les symboles du flot d'entrée jusqu'à ce qu'apparaisse une entité lexicale appartenant à un ensemble sélectionné d'unités lexicales de synchronisation :
  - SUIVANT du non terminal en sommet de pile ;
  - ; dans les langages comme C ou Pascal ; …

Analyse syntaxique LL(1)
Transformation de grammaires

- Il n'existe pas de procédé automatique pour rendre une grammaire G LL(1).
- Mais par contre, certaines conditions sont nécessaires pour qu'une grammaire G soit LL(1). Ces conditions sont les suivantes :
  - La grammaire doit être factorisée à gauche ;
  - La grammaire ne doit pas être récursive gauche.

Analyse syntaxique LL(1)
Transformation de grammaires

- **Factorisation à gauche d'une grammaire :**
- Des productions non factorisées à gauche d'une grammaire du type :
  - $A \rightarrow \alpha\ \beta_1 \mid \alpha\ \beta_2 \mid \ldots \mid\ \alpha\ \beta_n \mid \gamma$
  - où $\alpha \neq \varepsilon$
- sont remplacées par les productions suivantes :
  - $A \rightarrow \alpha\ A' \mid \gamma$
  - $A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid\ \beta_n$

Analyse syntaxique LL(1)
Transformation de grammaires

- **<u>récursivité gauche d'une grammaire :</u>**
- Une grammaire est dite récursive à gauche si elle contient un non terminal A tel que :
  - $A \Rightarrow^* A\ \alpha$
  - où est une chaîne quelconque.

Analyse syntaxique LL(1)
Transformation de grammaires

- **Elimination de la récursivité à gauche immédiate :**
- Les règles sous la forme suivante :
  - $A \rightarrow A\,\alpha_1 \mid A\,\alpha_2 \mid \ldots \mid A\,\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$
  - où aucun β ne commence par A
- sont remplacées par les productions suivantes :
  - $A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'$
  - $A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon$

Analyse syntaxique LL(1)
Transformation de grammaires

- Cette transformation n'élimine pas la récursivité gauche indirecte.
- Par exemple, cette transformation n'a aucun effet sur la récursivité gauche indirecte de la grammaire dont les productions sont données ci-après :
  - $S \rightarrow A\ a \mid b$
  - $A \rightarrow A\ c \mid S\ d \mid \varepsilon$

Analyse syntaxique LL(1)
Transformation de grammaires

- **<u>Elimination de la récursivité à gauche indirecte :</u>**
- L'algorithme suivant, élimine systématiquement, les récursivités gauches (directes et indirectes) d'une grammaire.
- Il fonctionne correctement si la grammaire est sans cycle (dérivations $A \Rightarrow +A$ et sans production vide ($A \rightarrow \varepsilon$).

Analyse syntaxique LL(1)
Transformation de grammaires

- Ordonner les non terminaux A1, A2, …, An ;
- **Pour** i := 1 to n
- **Faire** **Pour** j := 1 to i – 1
- **Faire**
- Remplacer chaque production de la forme Ai $\rightarrow$ Aj γ
- par les productions Ai $\rightarrow$ $\delta 1\ \gamma \mid \delta 2\ \gamma \mid \ldots \mid \delta k\ \gamma$
- où Aj $\rightarrow$ $\delta 1 | \delta 2 | \ldots | \delta k$ sont les productions Aj courantes
- **Fait** ;
- Eliminer les récursivités gauches immédiates des Ai productions
- **Fait**.

Analyse syntaxique LL(1)
Transformation de grammaires

- Ordonner les non terminaux A1, A2, …, An ;
- **Pour** i := 1 to n
- **Faire** **Pour** j := 1 to i – 1
- **Faire**
- Remplacer chaque production de la forme Ai $\rightarrow$ Aj γ
- par les productions Ai $\rightarrow$ δ1 γ | δ2 γ | … | δk γ
- où Aj $\rightarrow$ δ1|δ2| … |δk sont les productions Aj courantes
- **Fait** ;
- Eliminer les récursivités gauches immédiates des Ai productions
- **Fait**.

Analyse syntaxique LL(1)
Transformation de grammaires

- **Exemple :**
- Eliminer la récursivité gauche de la grammaire dont les productions sont données ci-après :
  - $S \rightarrow A\ b \mid b \mid S\ b$
  - $A \rightarrow a \mid AS \mid S\ b$

Analyse syntaxique LL(1)
Transformation de grammaires

- Ordre S, A
- Etape 1:
  - Elimination de la récursivité gauche directe
  - $S \rightarrow A\ b\ S' \mid b\ S'$
  - $S' \rightarrow b\ S' \mid \varepsilon$
- Etape 2:
  - Substitution
  - $A \rightarrow a \mid A\ S \mid A\ b\ S'\ b \mid b\ S'\ b$
  - Elimination de la récursivité gauche directe
  - $A \rightarrow a\ A' \mid b\ S'\ b\ A'$
  - $A' \rightarrow S\ A' \mid b\ S'\ b\ A' \mid \varepsilon$

Analyse syntaxique LL(1)
Transformation de grammaires

- Grammaire finale :
  - $S \rightarrow A\ b\ S' \mid b\ S'$
  - $S' \rightarrow b\ S' \mid \varepsilon$
  - $A \rightarrow a\ A' \mid b\ S'\ b\ A'$
  - $A' \rightarrow S\ A' \mid b\ S'\ b\ A' \mid \varepsilon$

Analyse syntaxique LL(1)
Transformation de grammaires

- **Remarque :**
- Pour l'élimination de la récursivité à gauche d'une grammaire, l'ordre des non terminaux peut être pris aléatoirement.
- L'ordonnancement des non terminaux peut par contre influer sur les propriétés de la nouvelle grammaire.

Analyse syntaxique par descente récursive

- L'analyse syntaxique par descente récursive n'est en fait que la version récursive (au sens implémentation informatique) de l'analyse LL(1).
- C'est à dire qu'au lieu de manipuler la pile explicitement, celle-ci sera gérée implicitement lors des appels. L'analyseur (programme d'analyse) est constitué d'une suite de procédures.

Analyse syntaxique par descente récursive

- **Conditions préalables à une descente récursive**
- Pour faire une analyse syntaxique par descente récursive pour analyser les mots d'un langage L(G), la grammaire G doit vérifier les conditions LL(1) i.e.
- Pour toute paire de règles de G tel que $A \rightarrow \alpha \mid \beta$ on a :

$$DEBUT(\alpha.SUIVANT(A)) \cap DEBUT(\beta.SUIVANT(A)) = \emptyset$$

Analyse syntaxique par descente récursive

- **Ecriture des procédures**
- Les étapes suivantes montrent le principe d'écriture de l'ensemble des procédures de l'analyseur :
- On ajoute la règle de production suivante : $Z \rightarrow S$ # où S est l'axiome de la grammaire et # marqueur EOF.
- A chaque non terminal de la grammaire correspond une procédure sans paramètres ;
- On utilisera les variables **tc** et **ts** pour désigner, respectivement, le symbole courant du flot d'entrée et le symbole suivant dans ce flot ;

Analyse syntaxique par descente récursive

- Soit A un non terminal quelconque de la grammaire tel que : $A \rightarrow \alpha_1 \mid \ldots \mid \alpha_n$
- Le corps de la procédure A est définie comme suit :
  - **Procédure A()**
  - { **case** tc **of**
  - DEBUT(α_1.SUIVANT(A)) : "RESULTAT Traiter(α_1)" ;
  - DEBUT(α_2.SUIVANT(A)) : "RESULTAT Traiter(α_2)" ;
  - :
  - DEBUT(α_n.SUIVANT(A)) : "RESULTAT Traiter(α_n)" ;
  - **default** : "Erreur Syntaxique" ;
  - **end-case** ;
  - }

Analyse syntaxique par descente récursive

- **Traiter (α_i)**
- { if (α_i=ε) then Afficher ("**;**") endif;
- if (α_i=$X_1X_2\ldots X_m$)
- then Code(X_1);
- endif ;
- }

Analyse syntaxique par descente récursive

- **Code(Xj);**
- {if (j<=m)
- Then if (X_j est un terminal)
- then Afficher (**"if (tc=X_j)**
- **then tc=ts ;"**) ; Code (X_{j+1}) ;
- Afficher ("**else Erreur Syntaxique"**
- **endif ;"**
- endif ;
- if (X_j est un non terminal)
- Then Afficher (**"Call X_j() ;"**) ; Code (X_{j+1}) ;
- endif;
- endif ;
- }

Analyse syntaxique par descente récursive

- **<u>Remarque :</u>**
- Si le premier symbole de α_i est un terminal, il est inutile de rajouter le test "if (tc=X_j)" car nécessairement tc=X_j du fait que ce test initial est fait dans la procédure A().

Analyse syntaxique par descente récursive

- **<u>Exemple :</u>**
  - $E \rightarrow T\ E'$
  - $E' \rightarrow +\ T\ E' \mid -\ T\ E' \mid \varepsilon$
  - $T \rightarrow F\ T'$
  - $T' \rightarrow *\ F\ T' \mid /\ F\ T' \mid \varepsilon$
  - $F \rightarrow (E) \mid n$

Analyse syntaxique par descente récursive
Exemple

- **Procédure Z( )**
- **{**
- **case** tc of :
- (,n : **Call** E( ) ;
-         **if** tc ='#"
-         **then** "Chaîne syntaxiquement correcte"
-         **else** "Erreur syntaxique"
-         **endif** ;
- default :   "Erreur syntaxique"
- **end-case**;
- }

Analyse syntaxique par descente récursive
Exemple

- **Procédure E ( )**
- **{** **case** tc **of** :
-   (,n : **Call** T( ) ; **Call** E′( ) ;
-   **default** : "Erreur syntaxique" ;
- **end-case** ;
- }

Analyse syntaxique par descente récursive
Exemple

- **Procédure E'( )**
- **{**
- **case** tc **of** :
- \+ : tc=ts ; **Call** T( ) ; **Call** E'( ) ;
- \- : tc=ts ; **Call** T( ) ; **Call** E'( ) ;
- #,): ;
- **default** : "Erreur syntaxique" ;
- **end-case** ;
- }

Analyse syntaxique par descente récursive
Exemple

- **Procédure T ( )**
- **{**
- **case** tc **of** :
- (,n : **Call** F( ) ; **Call** T'( ) ;
- **default** : "Erreur syntaxique" ;
- **end-case** ;
- }

Analyse syntaxique par descente récursive
Exemple

- **Procédure T'( )**
- **{** **case** tc **of** :
- * : tc=ts ; **Call** F( ) ; **Call** T'( ) ;
- / : tc=ts ; **Call** F( ) ; **Call** T'( ) ;
- +,-,#,): ;
- **default** : "Erreur syntaxique" ;
- **end-case** ;
- }

Analyse syntaxique par descente récursive
Exemple

- **Procédure F( )**
- **{** **case** tc **of** :
- n : tc=ts ;
- (: tc=ts ;**Call** E();**if** tc=) **then** tc=ts **else** "Erreur syntaxique" ; **endif**;
- **default** : "Erreur syntaxique" ;
- **end-case** ;
- }