Plan

I. Limites de l’analyse syntaxique

II. Rôle de l’analyseur sémantique

III. Définition (ou analyse) dirigée par la syntaxe

IV. Table des symboles

V. Représentation par arbres

   1. Arbre syntaxique abstrait
   2. Arbre syntaxique décoré

VI. Évaluation des attributs

Plan

I. Limites de l’analyse syntaxique

II. Rôle de l’analyseur sémantique

III. Définition (ou analyse) dirigée par la syntaxe

IV. Table des symboles

V. Représentation par arbres

VI. Évaluation des attributs

I- Limites de l’analyse syntaxique

- L’analyseur syntaxique s’assure que les phrases (instructions) sont bien formées.

- Cependant, d’autres propriétés fondamentales d’un langage de programmation ne peuvent pas être vérifiées par l’analyseur syntaxique.

I- Limites de l'analyse syntaxique

Parmi les propriété non vérifiée :

- **Déclaration des variables** :

  Dans les langages à typage statique, on ne peut pas utiliser une variable sans qu'elle ne soit déclarée au préalable.

  Exemple :

  ```
  int a;
  a = b + 5 ;
  ```

  Syntaxiquement parlant, les deux instructions sont bien formées.
  Cependant, *b* n'est pas déclarée avant son utilisation.

I- Limites de l'analyse syntaxique

Parmi les propriété non vérifiée :

- **Déclarations multiples dans un même bloc**:

  On ne peut pas déclarer une variable plusieurs fois dans le même bloc (dans la même portée) .

  Exemple :

  ```
  int a;
  a = 1;
  int a;
  ```

  Syntaxiquement parlant, les trois instructions sont bien formées.
  Cependant, *a* est déclarée à deux reprises dans le même bloc.

I- Limites de l'analyse syntaxique

Parmi les propriété non vérifiée :

- **Paramètres obligatoires des fonction**:

  Lors d'un appel d'une fonction, le nombre et les types des paramètres obligatoires doivent être respectés (différence entre la déclaration et l'utilisation).

  Exemple :

  ```
  int somme(int a, int b){...};
  somme(3, 897.65) ;
  somme(10) ;
  ```

  Syntaxiquement parlant, les trois instructions sont bien formées.
  Cependant, les deux appels ne respectent pas la définition de la fonction.

NB: des langages comme Ruby permettent d'avoir un nombre variable de paramètres

I- Limites de l'analyse syntaxique

Parmi les propriété non vérifiée :

- **<u>Opérations sur des types non autorisées</u>**:

  Certaines opérations ne peuvent pas être effectuées sur certains types.

  <u>Exemple :</u>

  ```
  int a= 5;
  string b = "Salut";
  string c = a * b;
  ```

  Syntaxiquement parlant, les trois instructions sont bien formées.
  Cependant, multiplier un entier par une chaîne de caractères n'as pas de sens dans un langage de programmation tel que C# ou Java .

Plan

I. Limites de l’analyse syntaxique

II. Rôle de l’analyseur sémantique

III. Définition (ou analyse) dirigée par la syntaxe

IV. Table des symboles

V. Représentation par arbres

VI. Évaluation des attributs

II- Rôle de l’analyseur sémantique

- Le rôle de l’analyseur sémantique est de vérifier des propriétés comme celles qui viennent d’être présentées.

- Les contraintes à vérifier dépendent de la nature du langage considéré.

II- Rôle de l'analyseur sémantique

- Souvent l'analyse sémantique se fait en même temps que l'analyse syntaxique en utilisant les actions sémantiques intégrées dans les règles de productions.
  C'est ce qu'on appelle : définition (ou analyse) dirigée par la syntaxe (**DDS**)

- Cependant, il n'existe pas de méthode universelle pour effectuer l'analyse sémantique. Cela dépend, entre autres, des contraintes à résoudre et du contexte (les particularités du langage de programmation).

Plan

I. Limites de l’analyse syntaxique

II. Rôle de l’analyseur sémantique

III. Définition (ou analyse) dirigée par la syntaxe

IV. Table des symboles

V. Représentation par arbres

VI. Évaluation des attributs

III- Définition dirigée par la syntaxe (DDS)

- Une définition dirigée par la syntaxe (DDS) est un formalisme qui permet d'associer des actions (**règles sémantiques**) aux règles d'une grammaire où:
  - Chaque symbole (terminal ou non-terminal) de la grammaire possède un ensemble d'attributs (valeurs, types, ...)
  - Chaque règle de production possède un ensemble de règles sémantiques qui permettent de calculer les valeurs des attributs associées aux symboles de la règle de production

III- Définition dirigée par la syntaxe (DDS)

- Une **règle sémantique** est une suite d'instructions écrites dans un langage déterminé (C, Java, C#, ...). Elle peut contenir :
  - des affectations
  - des instructions d'affichage
  - des instructions conditionnelles (if ... else...)
  - …

III- Définition dirigée par la syntaxe (DDS)

Exemple de règles sémantiques

En supposant que:

- Chaque non-terminal possède un attribut "valeur" qui retourne la valeur calculée ou héritée de l'expression associée
- Chaque terminal possède un attribut "valeur_lexicale" qui retourne la valeur du symbole telle que lu dans le fichier du code source du programme à analyser (valeur du lexème)

Production de grammaire	Règle sémantique (Action)
Z → E$	Z.valeur ← E.valeur
E → E + T	E.valeur ← E.valeur + T.valeur
E → T	E.valeur ← T.valeur
T → T * F	T.valeur ← T.valeur * F.valeur
T → F	T.valeur ← F.valeur
F → (E)	F.valeur ← E.valeur
F → nb	F.valeur ← nb.valeur_lexicale

Plan

I. Limites de l’analyse syntaxique

II. Rôle de l’analyseur sémantique

III. Définition (ou analyse) dirigée par la syntaxe

IV. Table des symboles

V. Représentation par arbres

VI. Évaluation des attributs

IV- Table des symboles

La **table des symboles** est une structure de données où sont stockés les symboles et leurs attributs.

Exemple : À la rencontre de :

int X = 6;

float Y = 3.14;

La table de symboles contiendra :

Nom de l'objet	Type	Taille	Adresse	Valeur
X	int	4 octets	$(0)_H$	6
Y	float	4 octets	$(4)_H$	3.14

Plan

I. Limites de l’analyse syntaxique

II. Rôle de l’analyseur sémantique

III. Définition (ou analyse) dirigée par la syntaxe

IV. Table des symboles

V. Représentation par arbres

VI. Évaluation des attributs

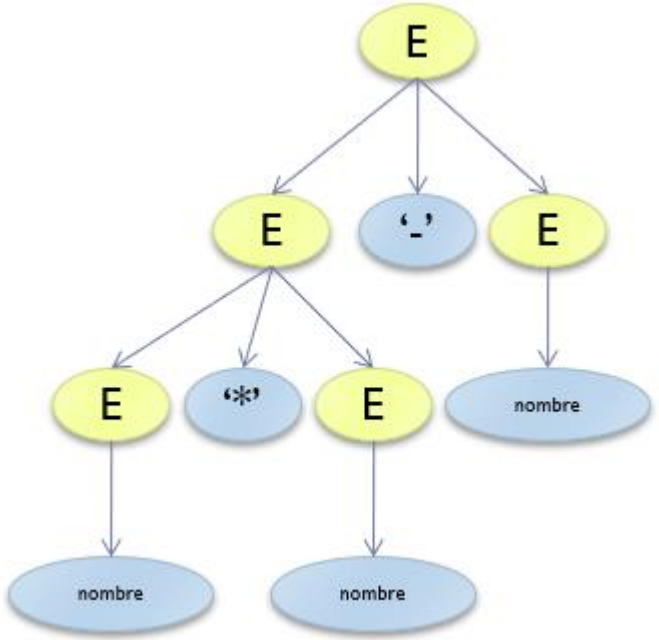
V- Représentation par arbres

- Dans l'analyse syntaxique, on génère un arbre de dérivation syntaxique pour représenter les phrases bien formées d'un langage;

- Cet arbre est appelé : **arbre syntaxique concret**

Exemple : E → E - E
| E * E
| nombre

Expression : **3 * 4 - 7**



arbre syntaxique concret

V- Représentation par arbres

- L'arbre syntaxique (concret) comporte des non-terminaux appartenant aux règles de la grammaire du langage qui **ne sont pas utiles pour l'analyse sémantique.**

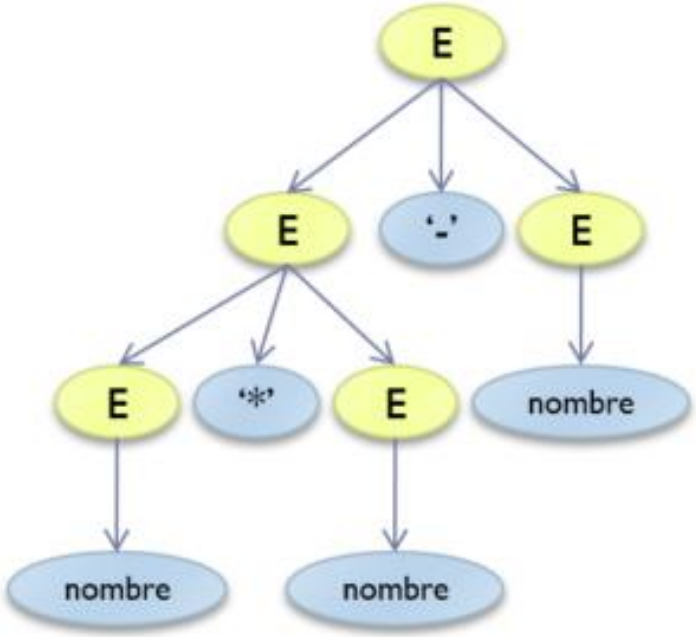
- En les éliminant, on obtient des nouvelles représentations d'un programme

  - Arbre syntaxique abstrait

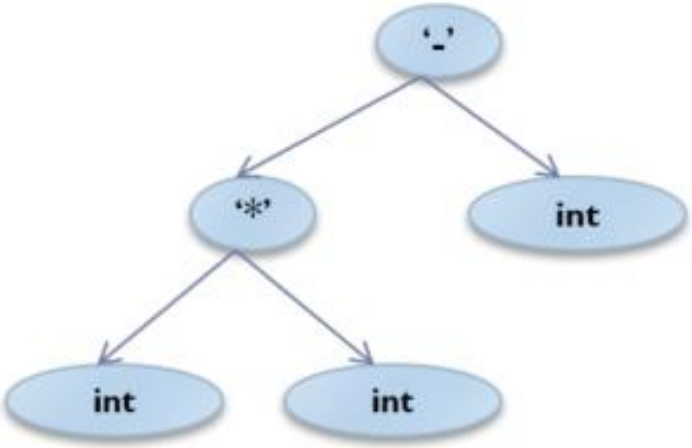
  - Arbre syntaxique décoré

V-1. Arbre syntaxique abstrait

- **L'arbre syntaxique abstrait** est construit à partir de l'arbre syntaxique de dérivation
- Il ne contient aucun non-terminal
- Il est (automatiquement) construit à partir des actions sémantiques des règles de production de l'analyseur syntaxique



Arbre syntaxique concret

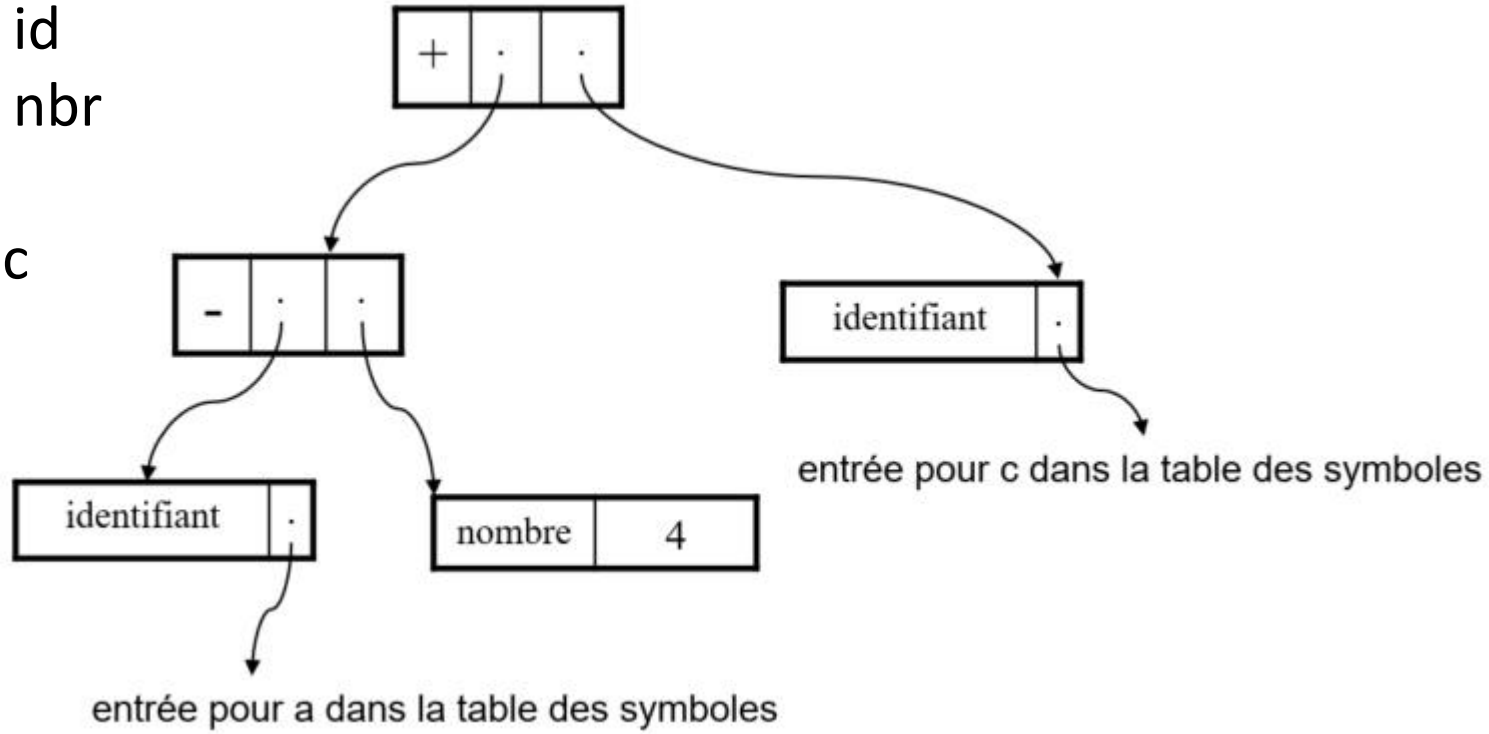


Arbre syntaxique abstrait

V-1. Arbre syntaxique abstrait

Exemple : E → E + T
E → E - T
E → T
T → (E)
T → id
T → nbr

Expression : a - 4 + c

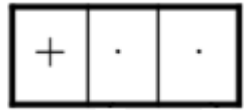


V-1. Arbre syntaxique abstrait

Construction d'un arbre abstrait :

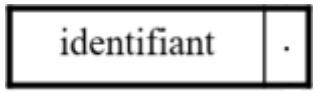
- Crée un nœud dont l'étiquette est l'opérateur "operateur" et avec deux champs de pointeurs vers les nœuds des opérandes gauche et droite:

  créerNoeud(operateur, ptr_operande_gauche,ptr_operande_droite)



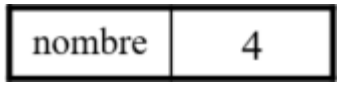
- Crée un nœud dont l'étiquette est "identifiant" et avec un champ pour un pointeur vers une entrée de la table des symboles (afin de pouvoir récupérer la valeur):

  créerFeuille(identifiant, ptr_entrée)



- Crée un nœud dont l'étiquette est "nombre" et avec un champ pour la valeur de la constante:

  créerFeuille(nombre, valeur)



V-1. Arbre syntaxique abstrait

Construction d'un arbre abstrait :

- Productions et actions sémantiques associées pour la création d'un arbre abstrait:

E → E + T	E.ptr → créerNoeud('+', E.ptr, T.ptr)
E → E - T	E.ptr → créerNoeud('-', E.ptr, T.ptr)
E → T	E.ptr → T.ptr
T → (E)	T.ptr → E.ptr
T → id	T.ptr → créerFeuille(identifiant, id.entree)
T → nbr	T.ptr → créerFeuille(nombre, nbr.val)

- Les deux attributs E.ptr et T.ptr contiennent des pointeurs vers les parties de l'arbre déjà construites

V-1. Arbre syntaxique abstrait

Construction d'un arbre abstrait :

- Étapes (algorithmiques) de construction de l'arbre pour l'expression "a-4+c":

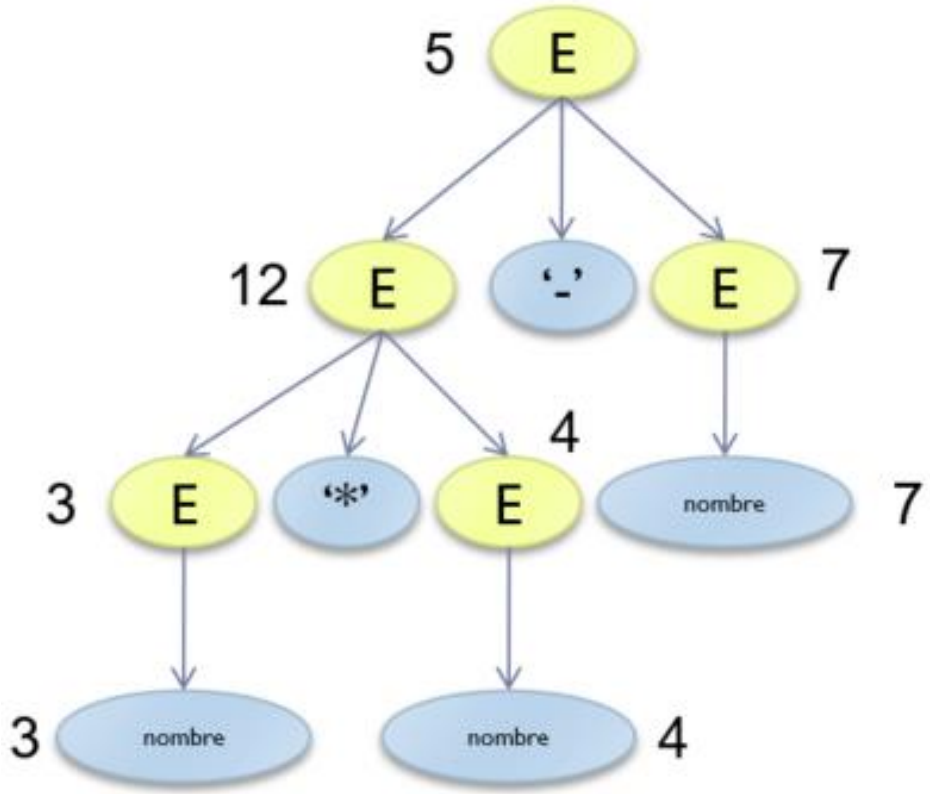
  Pointeur1 → créerFeuille(identifiant, ptr(a))
  Pointeur2 → créerFeuille(nombre, 4)
  Pointeur3 → créerNoeud('-', Pointeur1, Pointeur2)
  Pointeur4 → créerFeuille(identifiant, ptr(c))
  Pointeur5 → créerNoeud('+', Pointeur3, Pointeur4)

V-2. Arbre syntaxique décoré

- **Un arbre syntaxique décoré** est un arbre syntaxique de dérivation où les valeurs des attributs des non-terminaux sont calculées.

Exemple : E → E - E
| E * E
| nombre

Expression : **3 * 4 - 7**



Arbre syntaxique décoré

V-2. Arbre syntaxique décoré

Graphe de dépendances :

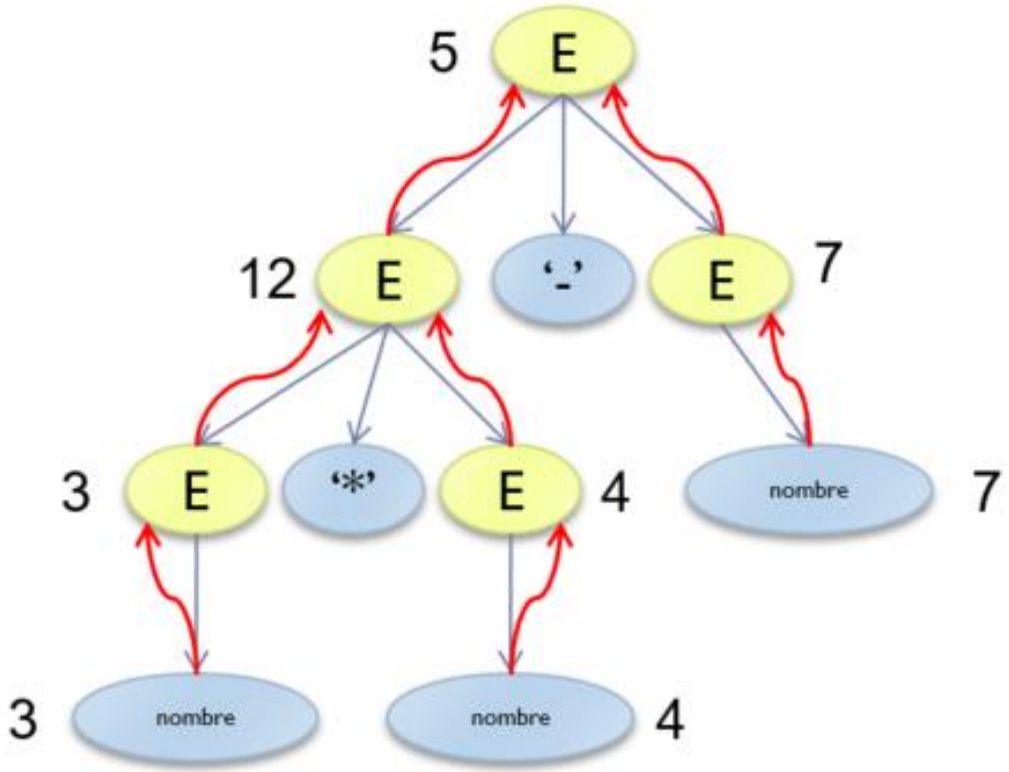
- Sur un arbre décoré, on ajoute une flèche si un attribut est employé pour calculer un autre.
  - Les flèches changent au maximum d'un niveau dans l'arbre
  - Elles peuvent aussi partir d'un niveau et atterrir dans le même niveau
  - L'ordre du calcul doit suivre les flèches
  - S'il y a un cycle, le calcul n'est pas possible

Le graphe de dépendances permet de spécifier l'ordre d'évaluation ("calcul") des attributs

V-2. Arbre syntaxique décoré

Graphe de dépendances :

Exemple : Les flèches en rouge sont ajoutées pour avoir le graphe de dépendances



Arbre syntaxique décoré

Plan

I. Limites de l’analyse syntaxique

II. Rôle de l’analyseur sémantique

III. Définition (ou analyse) dirigée par la syntaxe

IV. Table des symboles

V. Représentation par arbres

VI. Évaluation des attributs

VI- Évaluation des attributs

Les attributs peuvent être évalués de deux manières :

- Après l’analyse syntaxique
- Pendant l’analyse syntaxique

VI- Évaluation des attributs

Après l'analyse syntaxique:

- Le calcul des attributs est fait indépendamment de l'analyse syntaxique.
  Lors de l'analyse syntaxique, on construit l'arbre syntaxique. Une fois l'arbre construit, on calcule les attributs en parcourant cet arbre.

  - Inconvénient
    Méthode couteuse en mémoire : stockage de l'arbre.

  - Avantage
    Le calcul des attributs est indépendant de la méthode de construction de l'arbre (descendante ou ascendante)

VI- Évaluation des attributs

Pendant l'analyse syntaxique:

- Le calcul des attributs se fait en même temps que l'analyse syntaxique.
  Une pile est utilisée pour conserver les valeurs des attributs rencontrés.

  - Inconvénient

    L'évaluation des attributs est liée à l'ordre de la méthode de parcours syntaxique utilisée.

  - Avantage

    On peut utiliser la même pile que celle utilisée pour l'analyse syntaxique.
    => pas de surcoût significatif en terme d'espace mémoire.

VI- Évaluation des attributs

Exercice

Comment vérifier si une variable est utilisée sans être déclarée ?

VI- Évaluation des attributs

Solution

Comment vérifier si une variable est utilisée sans être déclarée ?

- Chaque symbole rencontré est inséré dans la table des symboles.
- Lors de l'utilisation d'une variable, le compilateur recherche dans la table des symboles si cette variable existe:
  - Si elle existe ➔ elle a été déclarée auparavant
  - Si elle n'existe pas ➔ le compilateur déclenche une erreur du type : identifiant utilisé avant qu'il soit déclaré