CHAPITRE 6 : TRADUCTION DIRIGEE PAR LA SYNTAXE ET ANALYSE SEMANTIQUE

6.1 Introduction

La grammaire hors contexte d'un langage de programmation ne peut pas décrire certaines propriétés fondamentales qui dépendent du contexte telles que :

- L'impossibilité d'utiliser dans ses instructions une variable non déclarée préalablement
- L'impossibilité de déclarer deux fois la même variable
- L'impossibilité de multiplier un réel par une chaîne de caractères
- La nécessité de correspondance (en nombre et en type) entre les paramètres formels et les paramètres effectifs des procédures et des fonctions

L'analyse sémantique (contextuelle) a pour rôle de vérifier ce genre de contraintes. Elle se fait, en général, en associant des règles sémantiques aux productions de la grammaire. Deux notations peuvent être utilisées pour effectuer cette association :

- Les définitions dirigées par la syntaxe ou
- Les schémas de traduction

Les règles sémantiques calculent la valeur des attributs attachés au symbole de la grammaire. Conceptuellement la chaîne en entrée est analysée syntaxiquement pour construire un arbre qui est parcouru autant de fois que nécessaire pour évaluer les règles sémantiques à ses nœuds. L'évaluation des règles sémantiques peut produire du code, sauvegarder de l'information dans la table des symboles, etc., pour obtenir la traduction de la chaîne en entrée.

6.2 Définitions dirigées par la syntaxe (DDS)

6.2.1 Introduction

Une DDS est un formalisme permettant d'associer des actions à une règle de production de la grammaire. On appelle ainsi DDS la donnée d'une grammaire et des règles sémantiques. On parle de grammaire **attribuée**.

Dans une DDS, on a les éléments suivants :

- Chaque symbole de la grammaire ($\in V_T$ ou V_N) possède un ensemble d'**attributs** (variables) pouvant être **synthétisés** ou **hérités** de ce symbole. Si un nœud d'arbre syntaxique (étiqueté par un symbole) est représenté par un enregistrement, un attribut correspond à un nom de champ de cet enregistrement.
- Chaque règle de production de la grammaire possède un ensemble de règles sémantiques qui permettent de calculer la valeur des attributs associés au symbole apparaissant dans la production
- Une règle sémantique est une suite d'instructions algorithmiques (elle peut contenir des affectations, des instructions d'affichage, des instruction de contrôle, etc.)

Les règles sémantiques impliquent des dépendances entre les attributs. Ces dernières sont représentées par un graphe de dépendance (GD) à partir duquel en déduit un ordre d'évaluation des règles sémantiques.

Remarque

Un arbre syntaxique muni des valeurs d'attributs à chaque nœud est appelé **arbre décoré** (ou **annoté**). Le processus de calcul des valeurs des attributs aux nœuds est appelé décoration (ou annotation) de l'arbre.

6.2.2 Structure d'une DDS

Dans une DDS, chaque production de la grammaire ($A \rightarrow \alpha$) possède un ensemble de règles sémantiques de la forme :

b := f(C1, C2, …, Ck)

f : est une fonction, souvent écrite sous forme d'expressions et parfois sous forme d'un appel de procédure.

b : est soit un attribut synthétisé de A, soit un attribut hérité d'un symbole en partie droite de la règle.

C1, C2, …, Ck : sont des attributs de symboles quelconques dans la règle de production.

6.2.3 Attributs synthétisés

La valeur d'un attribut synthétisé en un nœud est calculée à partir des attributs au niveau des fils de ce nœud dans l'arbre syntaxique.

Les attributs synthétisés sont très utilisés en pratique. Une DDS qui utilise uniquement les attributs synthétisés est appelée *Définition S-attribuée*.

Un arbre syntaxique pour une définition S-attribuée est toujours décoré en évaluant les règles sémantiques pour les attributs de chaque nœud du bas vers le haut (peut être adapté à une analyse ascendante : grammaire LR, par exemple).

Exemple

Soit la grammaire d'un programme de calculatrice de bureau :

$L \rightarrow En$
$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid Chiffre$

Productions	**Règles sémantiques**
$L \rightarrow En$	Imprimer (E.val)
$E \rightarrow E1 + T$	E.val := E1. val + T
$E \rightarrow T$	E.val := T .val
$T \rightarrow T1 * F$	T.val := T1.val * F.val
$T \rightarrow F$	T.val := F.val
$F \rightarrow (E)$	F.val :=E.val
$F \rightarrow Chiffre$	F.val := Chiffre.valLex

Remarques

- Cette DDS associe des attributs synthétisés « val » de type entier à chacun des non terminaux E, T et F.
- Le terminal Chiffre a un attribut synthétisé valLex (valeur lexicale) dont la valeur est fournie par l'analyse lexicale.
- Imprimer(E.val) est un appel de procédure pour imprimer la valeur de l'expression arithmétique E.

Exemple

Soit la chaîne 3*4+5n. On veut établir l'arbre syntaxique correspondant à cette chaîne, avant et après la décoration, en utilisant la DDS précédente.

Avant la décoration, on obtient l'arbre de la figure 6.1.

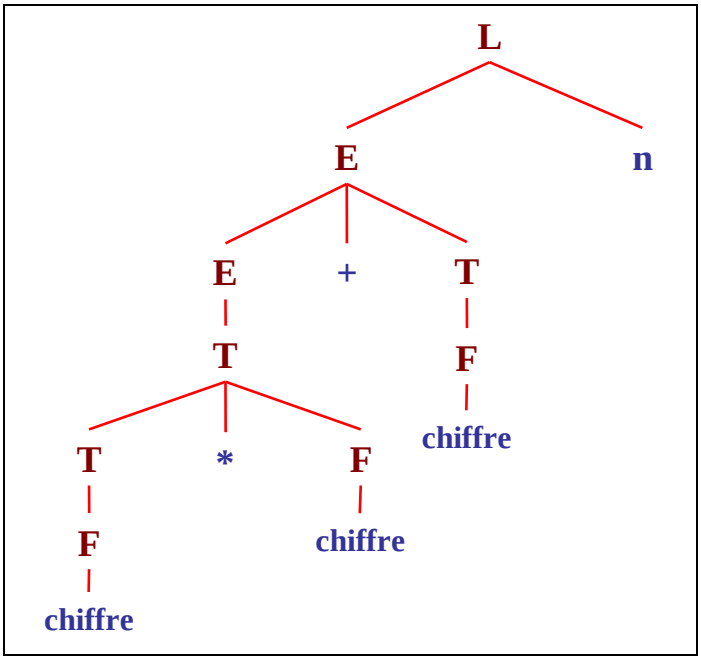

Figure 6.1. Arbre syntaxique de l'expression 3*4+5

Après la décoration, nous obtenons l'arbre décoré de la figure 6.2.

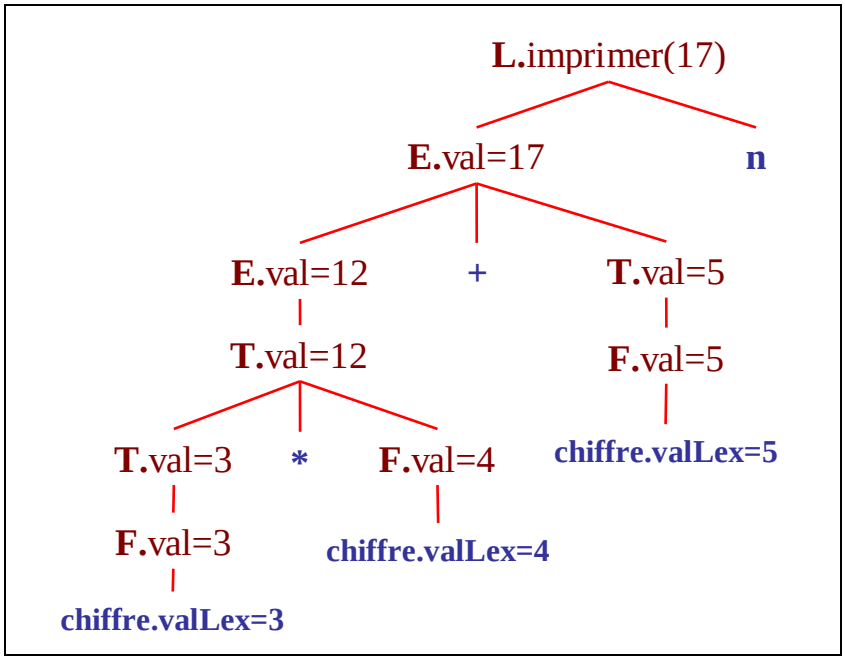

Figure 6.2. Arbre syntaxique décoré de l'expression 3*4+5

Remarque

Dans une DDS, les terminaux de la grammaire ne sont sensés avoir que des attributs synthétisés dont les valeurs sont fournies, généralement, par l'analyseur lexical.

6.2.4 Attributs hérités

Un attribut hérité est un attribut dont la valeur, à un nœud d'un arbre syntaxique, est définie en fonction du père et/ou des frères de ce nœud. Il est souvent plus naturel d'utiliser des attributs hérités qui sont bien adaptés à l'expression des dépendances contextuelles.

Exemple

Soit une grammaire permettant de déclarer des entiers et des réels :

D → T L
T → entier | réel
L → L, id | id

Dans la DDS suivante, on utilise un attribut hérité pour distribuer l'information de typage aux différents identificateurs d'une déclaration.

Productions	**Règles sémantiques**
D → T L	L.Type h := T. type
T → entier	T.Type := entier
T → réel	T.Type := réel
L → L1, id	L1.Type h := L.Type h ajouterType(id.entrée, L.Type h)
L → id	ajouterType (id.entrée, L.Type h)

La première règle sémantique L.Type h := T.Type, associée à la production D → T L, donne à l'attribut hérité L.Type h la valeur de type de la déclaration (entier ou réel). Les autres règles sémantiques font descendre ce type le long de l'arbre par l'intermédiaire de l'attribut hérité L.Type h. La procédure ajouterType() est appelée pour permettre d'ajouter le type de chaque identificateur au niveau de son entrée dans la table des symboles.

Exemple

En utilisant la DDS précédente, donner l'arbre syntaxique décoré de la déclaration : entier a, b, c

Avant la décoration, nous avons l'arbre syntaxique de la figure 6.3.

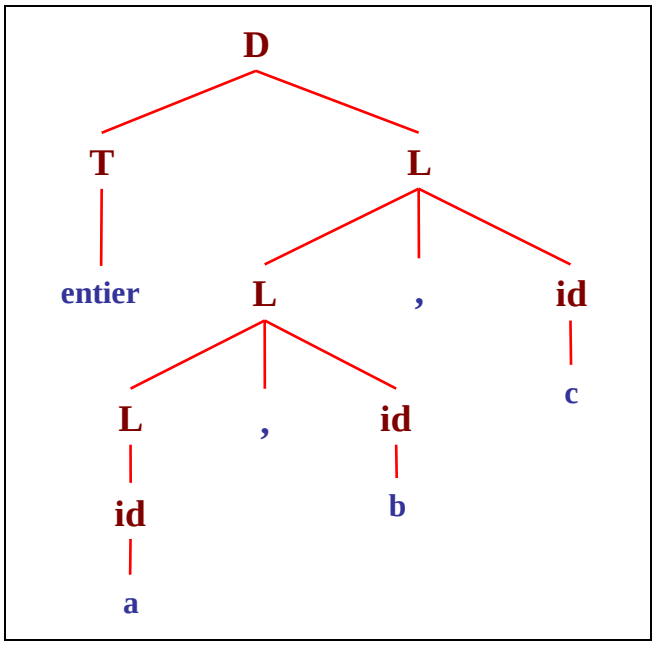

Figure 6.3. Arbre syntaxique de : entier a, b, c

Après la décoration, nous obtenons l'arbre syntaxique décoré de la figure 6.4.

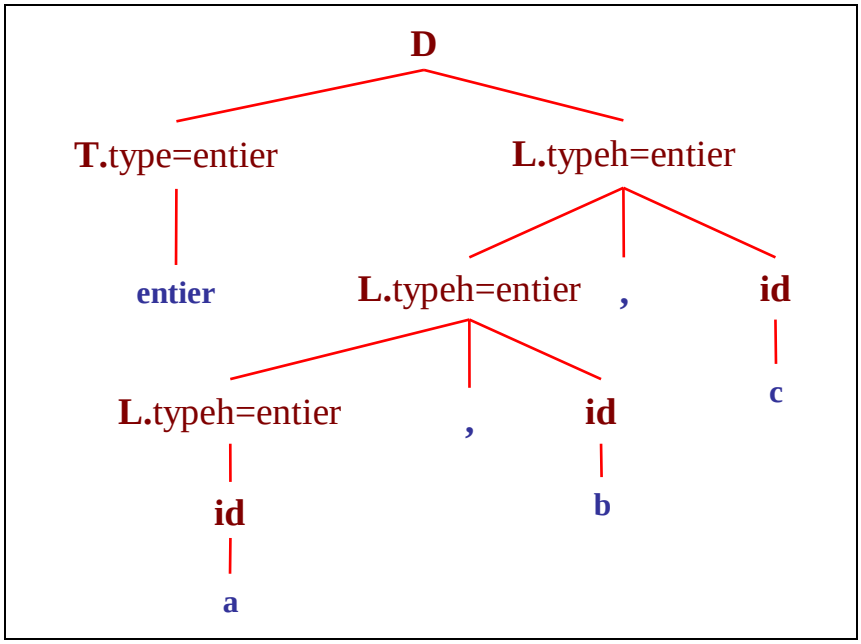

Figure 6.4. Arbre syntaxique décoré de : entier a, b, c

6.2.5 Graphe de dépendances

On appelle graphe de dépendance un graphe orienté représentant les interdépendances entre les attributs au niveau des nœuds un arbre syntaxique. Ce graphe permet de déduire l'ordre d'évaluation des règles sémantiques. Le graphe a un sommet pour chaque attribut. Si un attribut **b** dépend d'un attribut **c**, il y aura un arc du sommet c vers b, ceci signifie que la règle sémantique qui définit c doit être évaluée avant celle qui définit b.

Exemple 1

Productions	**Règles sémantiques**
E → E1 + E2	E.val := E1.val + E2.val

A chaque fois qu'on utilise la production E → E1 + E2 dans un arbre syntaxique, on ajoute les deux arcs ci-dessous (voir figure 6.5) au niveau du graphe de dépendance pour indiquer que l'évaluation de E.val doit être effectuée après celle de E1.val et E2.val

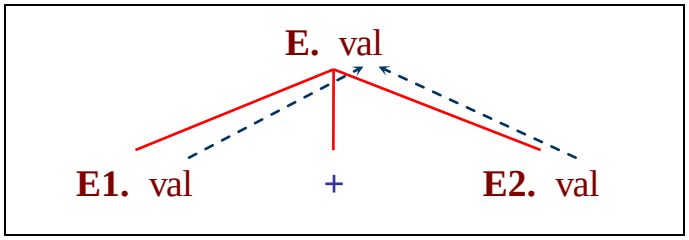

Figure 6.5. Graphe des dépendances de E → E1 + E2

Exemple 2

Productions	**Règles sémantiques**
A → XY	A.a := f(X.x, Y.y) X.h := g(A.a, Y.y)

Le graphe de dépendance est donné dans la figure 6.6.

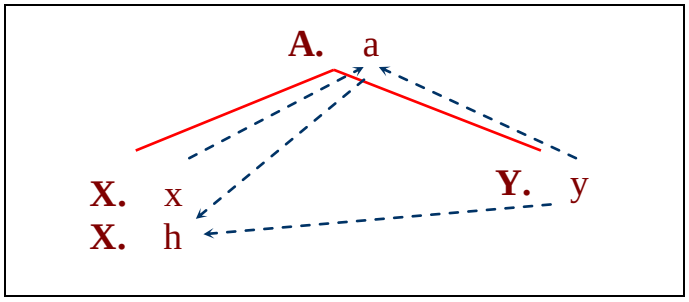

Figure 6.6. Graphe des dépendances de A → XY

Algorithme de construction du graphe des dépendances pour un arbre syntaxique donné

Pour chaque nœud n de l'arbre syntaxique **faire**

  Pour chaque attribut a du symbole de la grammaire étiquetant n **faire**

    Construire un sommet dans le graphe des dépendances pour a

Pour chaque nœud n de l'arbre syntaxique **faire**

  Pour chaque règle sémantique b := f(c1, …, ck) associé à la production appliquée en n **faire**

    Pour i := 1 à k **faire** construire un arc du sommet associé à ci au sommet associé à b

6.3 Utilisation des DDS pour la construction de représentation graphiques associées aux langages

6.3.1 Arbre abstrait

Un arbre abstrait est une forme condensée d'arbre syntaxique adaptée à la représentation des constructions des langages. Dans un arbre abstrait, les opérateurs et les mots clés n'apparaissent pas comme des feuilles mais sont plutôt associés au nœud intérieur qui sera le père de cette feuille dans l'arbre. De plus, les chaînes de productions simples peuvent être éliminées.

La traduction dirigée par la syntaxe peut se baser sur les arbres abstraits, de la même façon que sur les arbres syntaxiques concrets, en attachant des attributs aux nœuds. L'utilisation des arbres abstraits comme représentation intermédiaire permet de dissocier la traduction de l'analyse syntaxique.

Exemple 1

Soit à représenter l'arbre syntaxique concret et l'arbre abstrait de la chaîne 3 * 4 + 5 en utilisant la grammaire :

E → E + T | T

T → T * F | F

F → Chiffre

La figure 6.7 illustre les deux arbres.

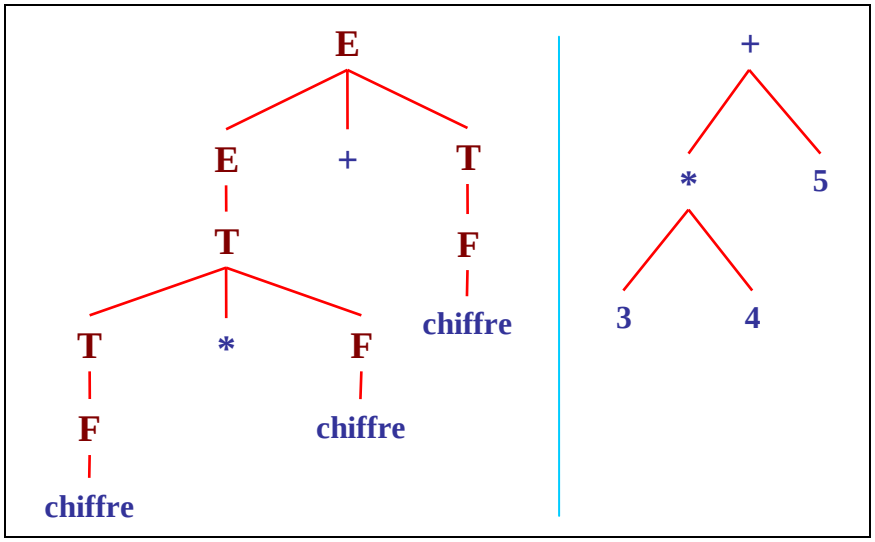

Figure 6.7. Arbre syntaxique concret et arbre abstrait de la chaîne 3 * 4 + 5

Exemple 2

Soit la règle de production I → Si C alors I1 Sinon I2

La figure 6.8 donnes les deux arbres

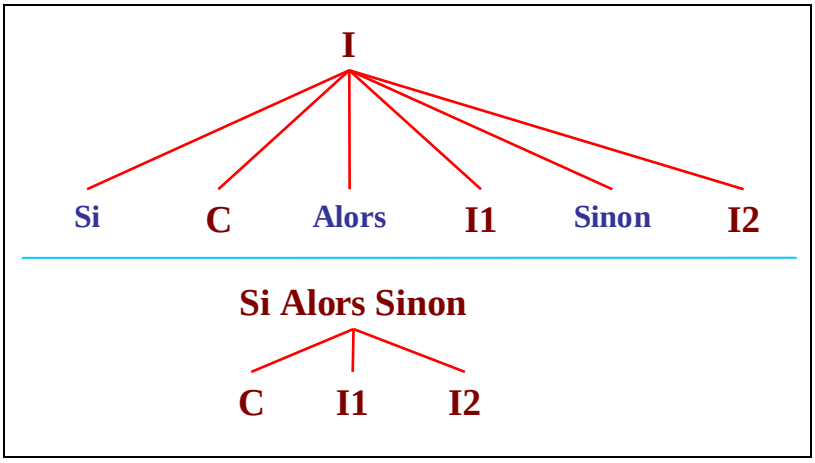

Figure 6.8. Arbre syntaxique concret et arbre abstrait de l'instruction Si-Alors-Sinon

6.3.2 Construction de l'arbre abstrait pour les expressions

La construction d'un arbre abstrait pour une expression est semblable à la traduction de l'expression en forme post-fixée (3 * 4 + 5 en forme post-fixée sera 3 4 * 5 +).

Chaque nœud de l'arbre abstrait peut être implémenté sous la forme suivante :

Opérateur	gauche	droit

Pour créer les nœuds des arbres abstraits correspondant à des expressions avec des opérateurs binaires, on utilise les fonctions suivantes :

1- CréerNoeud (Op, gauche, droit) : Crée un nœud opérateur ayant une étiquette op avec deux champs contenant des pointeurs vers gauche et droit.

2- CréerFeuille(id, entrée) : Crée un nœud identificateur (feuille) correspondant à un identificateur ayant pour étiquette id avec un champ entrée contenant un pointeur vers l'entrée de l'identificateur dans la table des symboles.

3- CréerFeuille(nb, val) : Crée une feuille associée à un nombre ayant pour étiquette « nb » avec un champ « val » contenant la valeur de ce nombre.

Exemple

On veut déterminer la séquence d'appels pour créer l'arbre abstrait représentant l'expression a - 4 + c

La séquence d'appels pour les fonctions est :

P1 := CréerFeuille(id, a)
P2 := CréerFeuille(nb, 4)
P3 := CréerNoeud('-', P1, P2)
P4 := CréerFeuille(id , c)
P5 := CréerNoeud('+', P3, P4)

L'arbre abstrait correspondant est donné par la figure 6.9. L'arbre abstrait physique est illustré dans la figure 6.10.

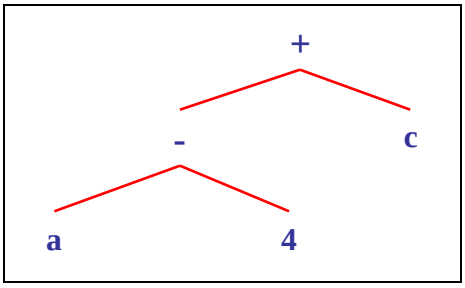

Figure 6.9. Arbre abstrait de l'expression a - 4 + c

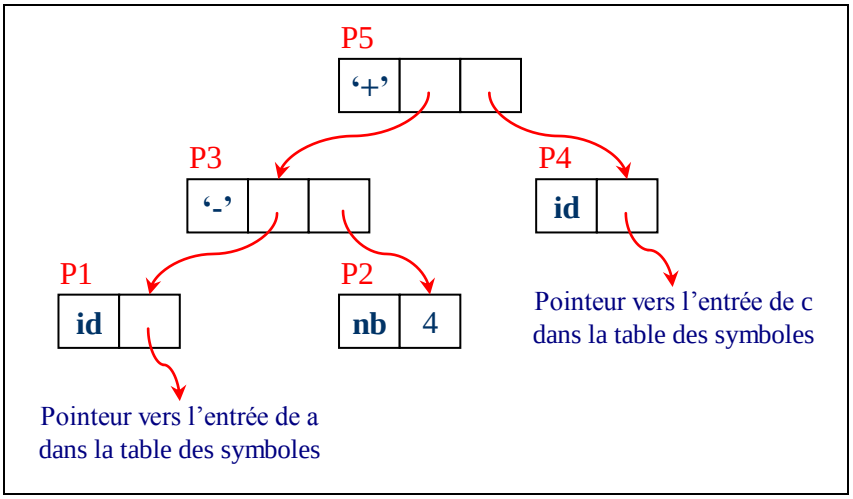

Figure 6.10. Représentation physique de l'arbre abstrait de l'expression a - 4 + c

Remarque

La construction est effectuée de manière ascendante.

6.3.3 Utilisation de la DDS pour la construction de l'arbre abstrait

La DDS suivante est une DDS s-attribuée pour la construction d'arbre abstrait d'expression contenant les symboles '+' et '-'.

Productions	Règles sémantiques
E → E1 + T	E.pnoeud := CréerNoeud('+', E1.pnoeud , T.pnoeud)
E → E1 - T	E.pnoeud := CréerNoeud('-', E1.pnoeud , T.pnoeud)
E → T	E.pnoeud := T.pnoeud
T → (E)	T.pnoeud := E.pnoeud
T → id	T.pnoeud := CréerFeuille(id, id.entrée)
T → nb	T.pnoeud := CréerFeuille(nb, nb.val)

En utilisant la DDS précédente, la création de l'arbre abstrait pour l'expression « a – 4 + c » sera effectuée (de façon ascendante) de la manière suivante :

- Construction de l'arbre syntaxique concret
- Association des règles sémantiques aux règles de production correspondantes dans DDS

La figure 6.11 donne la représentation physique de l'arbre abstrait pour l'expression en question.

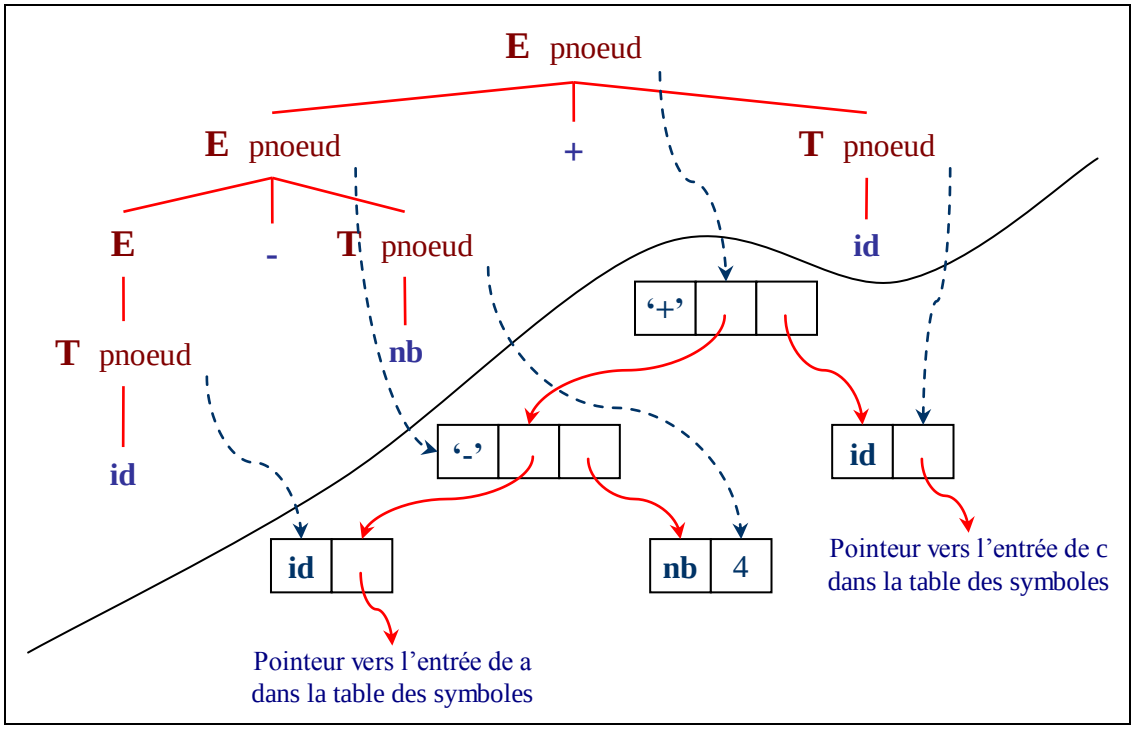

Figure 6.11. Construction de la représentation physique de l'arbre abstrait de l'expression a - 4 + c

6.3.4 Graphe orienté acyclique pour les expressions

Un graphe orienté acyclique (DAG : Direct Acyclic Graph) correspondant à une expression identifie les sous-expressions communes qu'elle contient. Le DAG renferme un nœud pour chaque sous-expression. Les nœuds intérieurs représentent des opérateurs et leurs fils représentent les opérandes.

La différence entre DAG et arbre abstrait est qu'un nœud du DAG représentant une sous-expression commune a plus d'un père.

Exemple

La figure 6.12 donne l'arbre abstrait et le DAG pour l'expression: a + a * (b - c) + (b-c) * d.

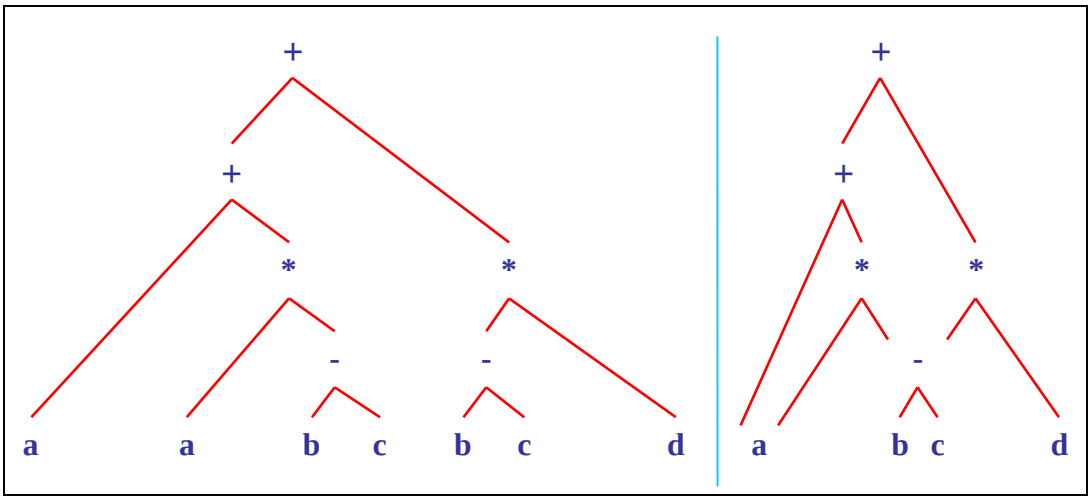

Figure 6.12. Arbre abstrait et DAG pour l'expression a + a * (b - c) + (b-c) * d

Les DDS peuvent être utilisés pour construire des DAG de la même manière que pour les arbres abstraits. Les corps des fonctions créerNoeud et créerFeuille doivent être modifiés pour vérifier, avant la création, si le nœud ou la feuille considérée n'a pas déjà été créée. Dans le cas de l'existence, ces fonctions retournent un pointeur vers le nœud ou la feuille construite précédemment

6.4 Evaluation ascendante des définitions s-attribuées

Les attributs synthétisés peuvent être évalués par un analyseur syntaxique ascendant au fur et à mesure que le texte d'entrée est lu. L'analyseur syntaxique peut conserver dans sa pile les valeurs des attributs synthétisés associés aux symboles de la grammaire. A chaque fois qu'une réduction est effectuée, les valeurs des nouveaux attributs synthétisés sont calculées à partir des attributs des symboles en partie droite de la règle de production qui sont disponibles dans la pile.

On peut supposer que la pile est implémentée en utilisant deux tableaux « état » et « val ». Si état[i] contient le symbole « X » alors val[i] contient la valeur de l'attribut attaché au nœud de l'arbre syntaxique correspondant à ce « X » . A tout moment, le sommet de la pile est repéré par le pointeur sommet.

Exemple

Productions	**Règles sémantiques**
A → XYZ	A.a := f(X.x, Y.y, Z.z)

Avant la réduction de XYZ en A , Val[sommet] contient Z.z, Val[sommet-1] contient Y.y, Val[sommet-2] contient X.x

	Etat	**Val**
	X	X.x
	Y	Y.y
Sommet	Z	Z.z

Après la réduction de XYZ en A, le sommet est décrémenté de 2, etat[sommet] contient A, val[sommet] contient A.a.

	Etat	**Val**
Sommet	A	A.a

6.5. Evaluation en profondeur pour les attributs d'un arbre syntaxique

Quand la traduction se fait pendant l'analyse syntaxique, l'ordre d'évaluation des attributs est liée à l'ordre de création des nœuds de l'arbre syntaxique par la méthode d'analyse. Pour beaucoup de méthodes de traduction (ascendantes ou descendantes), un ordre naturel d'évaluation est obtenu en appliquant à la racine d'un arbre syntaxique la procédure suivante :

```
Procédure VisiterEnProfondeur (n :noeud) ;
Début
  Pour Tout fils m de n, de gauche à droite faire
  Début
    Evaluer les attributs hérités de m ;
    VisiterEnProfondeur(m) ;
  Fin ;
  Evaluer les attributs synthétisés de m
Fin ;
```