CHAPITRE 7 : TRADUCTION DIRIGEE PAR LA SYNTAXE ET GENERATION DE CODE INTERMEDIAIRE

7.1 Code à trois adresses

Les méthodes de traduction dirigées par la syntaxe sont utilisées pour produire les formes intermédiaires associées au langage de programmation. Parmi ces formes, on trouve les représentations graphiques telles que les arbres abstraits et les DAG ainsi que les représentations textuelles telles que le code à trois adresses qui est proche du langage d'assemblage. Dans ce code, chaque instruction contient en général trois adresses : 2 pour les opérandes et une pour le résultat.

Le code à trois adresses est une représentation intermédiaire bien adaptée à la complexité des problèmes des expressions arithmétiques et aux structures de contrôle imbriquées.

Le code à trois adresses correspond à une représentation linéaire des arbres abstraits et des DAG dans laquelle des noms explicites correspondent aux nœuds internes des graphes.

Un code à trois adresses est une séquence d'instructions de la forme x := y op z où x,y et z sont des noms de constantes ou de variables temporaires produites par le compilateur. op étant un opérateur arithmétique ou logique.

Exemple

Soit à trouver la représentation sous forme d'arbre abstrait et de code à trois adresses de l'instruction d'affectation suivante : a := x + y * -z

La figure 7.1 donne les représentations demandées :

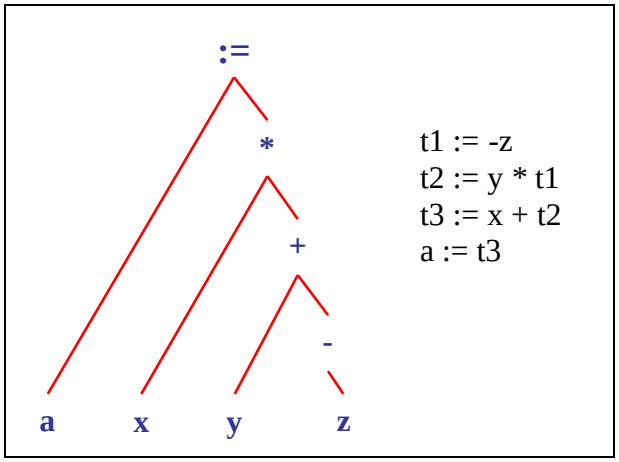

Figure 7.1. Représentations de l'affectation a := x + y * -z

Remarque

Une instruction à trois adresses est une forme abstraite de code intermédiaire qui peut être implémentée dans un compilateur par des structures dont les champs contiennent l'opérateur, les opérandes et les résultats. Parmi ces structures, on trouve les quadruplets, les triplets et les triplets indirects.

7.2 Structure de quadruplet

Une structure de quadruplet est représentée par une table «TQ » contenant N lignes. Un quadruplet correspond à une ligne et consiste en quatre champs :

TQ	Opérateur	Argument 1	Argument 2	Résultat	
i	*	y	t1	t2	t2 := y * t1
N					

Remarque

Argument 1, Argument 2 et Résultat sont généralement des pointeurs sur les entrées (de la table des symboles) qui correspondent aux nœuds représentés par ces champs.

Dans un schéma de traduction utilisant les quadruplets, on peut faire appel à une procédure permettant de générer un quadruplet correspondant à Res := A1 op A2.

```
Procédure Générer (oper :opérateur, A1, A2, Res : opérande) ;
Début
  TQ[quadSuiv].Opérateur := oper ;
  TQ[quadSuiv].Argument1 := A1;
  TQ[quadSuiv].Argument2 := A2;
  TQ[quadSuiv].Résultat := Res ;
  quadSuiv := quadSuiv + 1 ;
Fin ;
```

Remarque

TQ est la table de stockage des quadruplets.

quadSuiv : la prochaine ligne libre dans TQ.

Il existe aussi une fonction CréerTemp qui peut être utilisée dans les schémas de traduction, pour créer une variable temporaire dont le nom doit être différent de ceux utilisés par le programmeur.

```
Fonction CréerTemp (Var cpt : entier) : chaîne ;
Début
  CréerTemp := '$'+ConvEntChaîne(cpt) ;
  cpt := cpt + 1 ;
Fin ;
```

cpt est un compteur de variables temporaires initialisé à zéro. ConvEntChaîne est une fonction qui convertit un entier en une chaîne de caractères.

Ainsi, CréerTemp est une fonction qui permet de générer séquentiellement des noms de variables temporaires $0 ; $1, $2, …

7.3 Traduction des expressions arithmétiques

Considérons la traduction des l'expression suivante en quadruplets : a * b + 3 * (b - c) + d

Le code quadruplet est le suivant :

$0 := a * b
$1 := b - c
$2 := 3 * $1
$3 := $0 + $2
$4 := $3 + d

En analysant le code quadruplet obtenu, on constate que :

- Le nombre de quadruplets est égal au nombre d'opérateurs dans l'expression
- Pour chaque quadruplet, il faut créer une variable temporaire, donc, pour chaque règle de la grammaire contenant un opérateur, l'action sémantique associée doit contenir une instruction de création de variables temporaire et une instruction de génération de quadruplet.

Schéma de traduction

Soit la grammaire des expressions arithmétiques :

E → E + E | E – E | E * E | E / E | (E) | -E | id | nb

Règle de production	**Schéma de traduction**
E → E1 (+|–|*|/) E2	x := créerTemp(cpt) E.nom := x Générer ((+/–/*/|), E1.nom, E2.nom,x)
E → – E1	x := créerTemp(cpt) E.nom := x Générer(x := -E1.nom)
E → (E1)	E.nom := E1.nom
E → id	E.nom := id.nom
E → nb	E.nom := nb.nom

Remarque

Pour pouvoir générer les quadruplets, il faut transmettre le nom de la variable résultat de l'expression E à l'unité syntaxique qui contient E par l'intermédiaire de la traduction du nom associé à E.

Exemple

On se propose de construire l'arbre syntaxique de l'expression a *b + 3 * (b – c) + d avec l'évaluation de sa traduction en code quadruplet selon le schéma de traduction précédent.

On obtient le la séquence de code suivante :

$0 := a * b
$1 := b - c
$2 := 3 * $1
$3 := $0 + $2
$4 := $3 + d

La figure 7.2 donne l'arbre abstrait de l'expression :

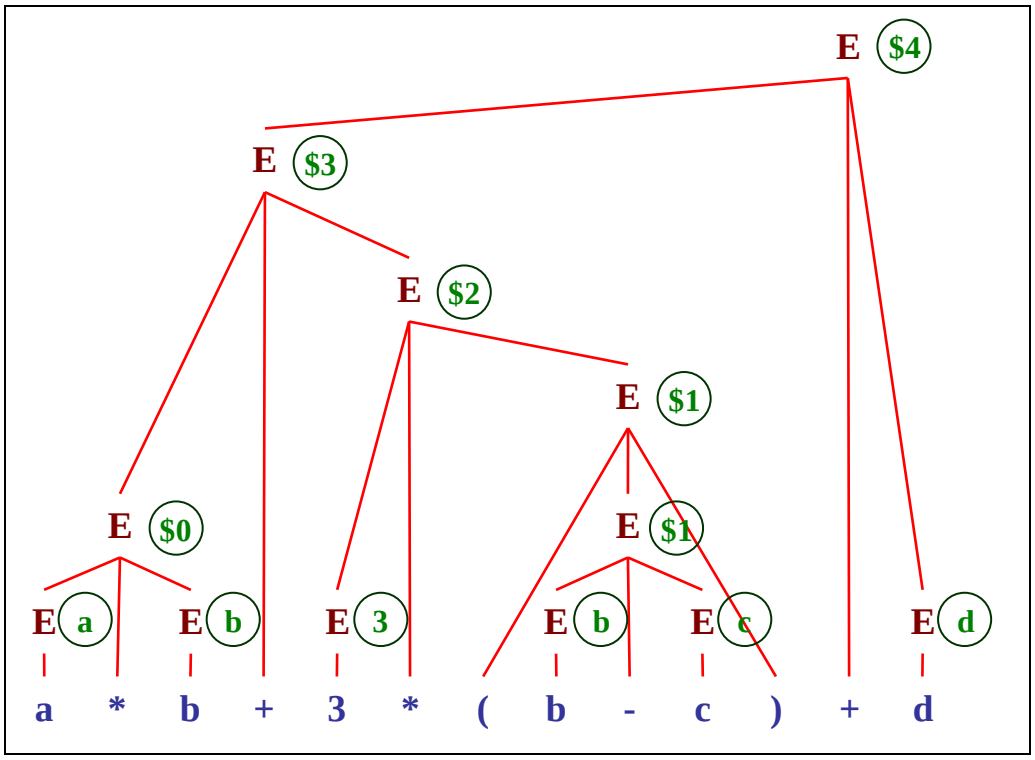

Figure 7.2. Arbre syntaxique de l'expression a *b + 3 * (b – c) + d
avec l'évaluation de sa traduction en code quadruplet

7.4 Traduction des affectations de variables simples

Soit l'affectation a := (a + b) * c. Le code quadruplet qui lui correspond est :

$0 := a + b
$1 := $0 * c
a := $1

On remarque qu'il faut d'abord traduire l'expression (a + b) * c en code quadruplet puis générer le code quadruplet qui affecte le résultat à a.

id.nom étant le résultat de l'expression E.nom

A la production I $\rightarrow$ id := E correspond le schéma de traduction : Générer (id.nom :=E.nom)

La figure 7.3 donne l'arbre de dérivation de l'affectation considérée.

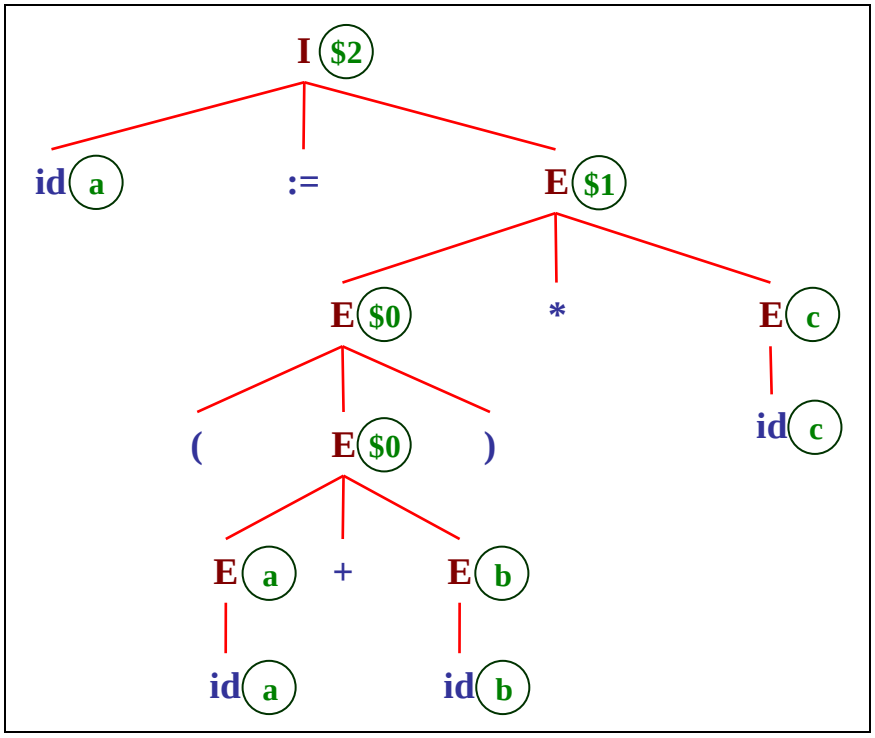

Figure 7.3. Arbre de dérivation de l'affectation a := (a + b) * c
avec l'évaluation de sa traduction en code quadruplet

7.5 Traduction des expressions booléennes

Les expressions booléennes sont traduites de manière analogue aux expressions arithmétiques.

Traduisons en code quadruplet l'affectation suivante : c := a < b

On associe la variable temporaire $0 à l'expression a < b de telle sorte que si a est inférieur à b, $0 sera égale à 1 et elle sera égale à 0 dans le cas contraire.

On obtient le code quadruplet suivant :

0 : if a < b goto 3
1 : $0 := 0
2 : goto 4
3 : $0 := 1
4 : c := $0

Soit la grammaire suivante, permettant de générer les expressions booléennes:

E → E and E | E or E | not E | (E) | id | id OpRel id

OpRel → < | <= | > | >= | = | <>

Le tableau suivant donne les schémas de traduction correspondant aux règles de production de la grammaire :

Règle de production	Schéma de traduction
E → E1 (and|or) E2	x := créerTemp(cpt) E.nom := x Générer (x := E1.nom (and|or) E2.nom)
E → not E1	x := créerTemp(cpt) E.nom := x Générer(x := not E1.nom)
E → (E1)	E.nom := E1.nom
E → id	E.nom := id.nom
E → id1 OpRel id2	x := créerTemp(cpt) E.nom := x Générer (if id1.nom OpRel id2.nom goto quadSuiv + 3) Générer (x := 0) Générer (goto quadSuiv + 2) Générer (x := 1)

Exemple

Soit à traduire en code quadruplet l'affectation suivante :

a := b < c or d > f and g = h

$0	0 : if b < c goto 3 1 : $0 := 0 2 : goto 4 3 : $0 := 1
$1	4 : if d > f goto 7 5 : $1 := 0 6 : goto 8 7 : $1 := 1
$2	8 : if g = h goto 11 9 : $2 := 0 10 : goto 12 11 : $2 := 1
	12 : $3 := $1 and $2 13 : $4 := $0 or $3 14 : a := $4