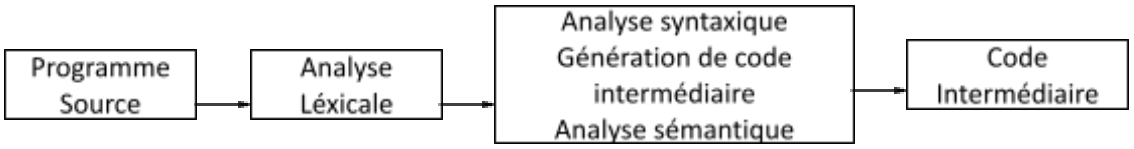
Chapitre II

Traduction dirigée par la syntaxe



L'analyse sémantique en liaison étroite avec l'analyse syntaxique, permet de donner un sens à ce qui été reconnu dans cette phase.

Exemple :

- Vérifier si un identificateur a été bien déclaré. A, x : integer ; y : boolean
- ……. x:= a+b b non déclaré
  y:= x *2 vous avez déclaré x mais pas y, il vous dira y non déclaré !!!

- La concordance des types dans une expression.
- Utilisation correcte d'une étiquette (déclarée, utilisée, référencée).
  Go to etiquette or vous oubliez de mettre etiquette : inst nous allos en faire une erreur de compil
  etiquette : a:= a+b etiquette referencée mais non définie (ou utilisée)

  go to étiquette

  procedure exemple (liste-param) la sémantique du mot clé procédure est énorme !!!!!!

  le call à cette procédure !!! les paramètres sont empilés
  nom -procedure (paramètres) ; dans la ts
- …….

On dispose d'un analyseur syntaxique, notre but est de récupérer comme résultat du code intermédiaire. Au code intermédiaire (forme post fixée, quadruplés, …..), nous rajoutons des actions sémantiques appelées les routines sémantiques. Ceci est appelé une traduction dirigée par la syntaxe ou analyse sémantique.

Une **routine sémantique** est donc appelée, à chaque fois qu'un contrôle sémantique est nécessaire, après une réduction, s'il s'agit de l'analyse ascendante, sinon dérivation, dans le cas descendante.

On appelle **schéma de traduction l'ensemble :**

- **Grammaire sémantique (transformée).**
- **Code intermédiaire. On obtientce code**
- **Routines sémantiques.**

II.1 Traduction dirigée par la syntaxe dans le cas de l'analyse descendante :

Les routines sémantiques correspondant aux dérivations. La grammaire doit être transformée en conséquence :

Si A ⟶ αβ et si une action sémantique doit être insérée entre α et β,

Alors A ⟶ αBβ A, B € N, α , β € (TUN)*

B ⟶ ε

On introduit des non-terminaux dérivant en ε.

II.1.1 Instruction conditionnelle :

<Instruction-IF> ⟶ IF <exp-bool> THEN <inst1> ELSE <inst2>

Forme intermédiaire sous forme de quadruplés :

Quadruplés de <exp-bool> ----------→ Résultat exp-bool

on sauvegarde ce numéro -----> (BZ, else, <exp-bool>.temp,)

Quadruplés de <inst1>

(BR, Fin, ,)

Quadruplés de <inst2>

om met à jour l'adresse de la fin

Inclusion des routines sémantiques :

<Instruction-IF> ⟶ IF <exp-bool> **<A>** THEN <inst1> **<B>** ELSE <inst2> **<C>**

<A> ⟶ ε (* Permet la génération d'un branchement vers debut else, si <exp-bool> est fausse, sauvegarde de l'étiquette de BZ *)

<B> ⟶ ε (* Génération d'un BR, sauvegarde de l'étiquette de BR, et Mise à jour de l'étiquette de BZ *)

<C>——————→ ε (* Mise à jour de l'étiquette de BR *)

Remarque :

QUAD : Matrice des quadruplés.

Qc : 1ère quadruplé libre dans la matrice QUAD. Quadruplet courant Qc quad(1)

Les routines sémantiques :

on va ecrire les procédures sémantiques

Routine <A> , A -----→ epsilon

Debut

QUAD(Qc) := (BZ, , <exp-bool>.temp,); on incrémente le Qc

Sauv-BZ := Qc;

Qc := Qc+1;

Fin.

Routine <B>

Debut

QUAD(Qc) := (BR, adresse de la fin , ,);

Sauv-BR := Qc;

Qc := Qc+1;

QUAD(Sauv-BZ, 2) := Qc;

Fin.

Routine <C>

Debut

QUAD(Sauv-BR, 2) := Qc;

Fin.

II.1.2 Instruction While :

<Inst-While>——————→ WHILE <condition> DO <Inst> fonctionnement de l'instruction

Forme intermédiaire sous forme de quadruplés :

Quadruplés de <condition>

(BZ, Fin, <condition>.temp,)

Quadruplés de <Inst>

(BR, Debut, ,)

Grammaire transformée (associée) :

<Inst-While>——————→ WHILE **<A>** <condition> **<B>** DO <Inst> **<C>**

<A>——————→ ε (* Permet la sauvegarder du debut While afin d'y revenir *)

<B>——————→ ε (* Permet le test de la condition *)

<C>——————→ ε (* Générer un branchement BR au debut du While et met à jour le BZ *)

Les routines sémantiques :

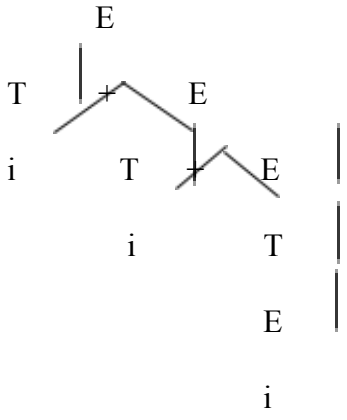
```
Routne <A>
Debut
    Sauv-deb := Qc;
FIN.

Routine <B>
Debut
    QUAD(Qc) := (BZ,     , <condition>.temp,     );
    Sauv-BZ := Qc;
    Qc := Qc+1;
Fin.

Routine <C>
Debut
    QUAD(Qc) := (BR, Sauv-deb,    ,      );
    Qc := Qc+1;
    QUAD(Sauv-BZ, 2) := Qc;
Fin.
```

II.1.3 Expressions arithmétiques :

E ⟶ T+E / T
T ⟶ F*T / F
F ⟶ i / (E)

On remarque que dans cette grammaire, l'associativité se fait de droite à gauche.

Exemple : i+i+i



On commencera à exécuter (i+i) dans i+(i+i).
On peut transformer, afin de garder l'associativité de gauche à droite de la façon suivante :

E ⟶ T {+T}*
T ⟶ F{*F}*
F ⟶ i / (E)

Insertion des procédures sémantiques :

E ⟶ T{+T <A>}*
T ⟶ F{*F <B>}*
F ⟶ i <C>/ (E)
<A> ⟶ ε
<B>, ⟶ ε, <C> ε

Analyse Syntaxico- Sémantique par la descente récursive :

```
Procédure E (X : entité, Y : Type ; B : booléan)
Debut
    Var : opérande1, opérande2 : entité;
          Type1, Type2          : Type ;
    T(opérande1, Type1, b);
    Si b= faux  Alors aller à fin Fsi;
    Tant que tc = '+'
       Faire  tc := tc+1;
              T(opérande2, Type2,b);
              Si erreur Alors aller à fin Fsi;
              A(opérande1, Type1, opérande2, Type2, X, Y);
              opérande1 : = X,
               opérande2:= Y
       Fait;
     X := opérande1;
     Y := Type1;
Fin.
```

```
Procédure T (X : entité, Y : Type, b : boolean) ;
Debut
    Var : opérande1, opérande2 : entité;
    Type1, Type2          : Type ;
    F(opérande1, Type1, b);
    Si b=fausx Alors aller à fin Fsi;
    Tant que tc = '*'
       Faire  tc := tc+1;
              F(opérande2, Type2,b);
              Si b=faux Alors aller à fin Fsi;
              B(opérande1, Type1, opérande2, Type2, X,Y);
              opérande1:= X ;
              Type1 := Y
       Fait;
     X := opérande1;
     Y := Type1;
Fin.
```

```
Procédure F (X : entité, Y : Type ; b : booléan))
Debut
    Si tc = '(' Alors tc := tc+1;
                    E(X, Y, b);
                    Si b= faux  Alors aller à fin Fsi;
                    Si tc = ')' Alors tc := tc+1;
                                Sinon b=faux
                    Fsi;
                Sinon si tc= id alors C(opérande1, Type1, X Y);
                                    tc := tc+1;
                                Sinon  b=faux
                    Fsi;
     Fsi;
Fin.
Procédure <A> (opérande1, opérande2 : entité; Type1, Type2 : Type ; b : booléan)
 Début
    - Vérifier compatibilité des types des deux opérandes;
    - CréerTemp (X);
    - Générer (+, opérande1, opérande2, X);
    - Y:=  calcul type (type1, type2)
      Fin.


Procédure <B> (opérande1, opérande2 : entité; Type1, Type2 : Type ; b : booléan)
Début
    - Vérifier compatibilité des types des deux opérandes;
    - CréerTemp (X) ;
    - Générer (*, opérande1, opérande2, X);
    - Y:= CalculType (Type1, Type2 )
      Fin

      Procédure C (opérande1: entité; Type1 : Type ; b : booléan)
      Début
       lookup (id. nom, p)
       si p =0 alors b= faux
      sinon  X:= p. nom ;
             Y:= p. type
fsi
Fin
```

II.2 Traduction dirigée par la syntaxe dans le cas de l'analyse ascendante :

Dans le cas de l'analyse ascendante, une routine est associée à une réduction. Il faudra donc transformer la grammaire en conséquence. Ce traitement est appelé découpage de la grammaire.

Si <A> $\longrightarrow$ $\alpha\beta$ et une routine doit être insérée entre α et β

Alors <A> ⟶ <B>β
<B> ⟶ α

Fsi.

II.2.1 Instruction conditionnelle :

<Inst-IF> ⟶ IF <exp-bool> THEN <Inst1> ELSE <Inst2>

Forme intermédiaire sous forme post fixée :

Pf(<exp-bool>) else BZ Pf(<Inst1>) fin BR Pf(<Inst2>)

Grammaire transformée :

<Inst-IF> ⟶ <debut-inst-if> <Inst2> R3
<debut-inst-if> ⟶ <debut-if> <Inst1> ELSE R2
<debut-if> ⟶ IF <exp-bool> THEN R1

Remarque :

Pf : vecteur de la forme post fixée.
i : 1ère position libre dans Pf.

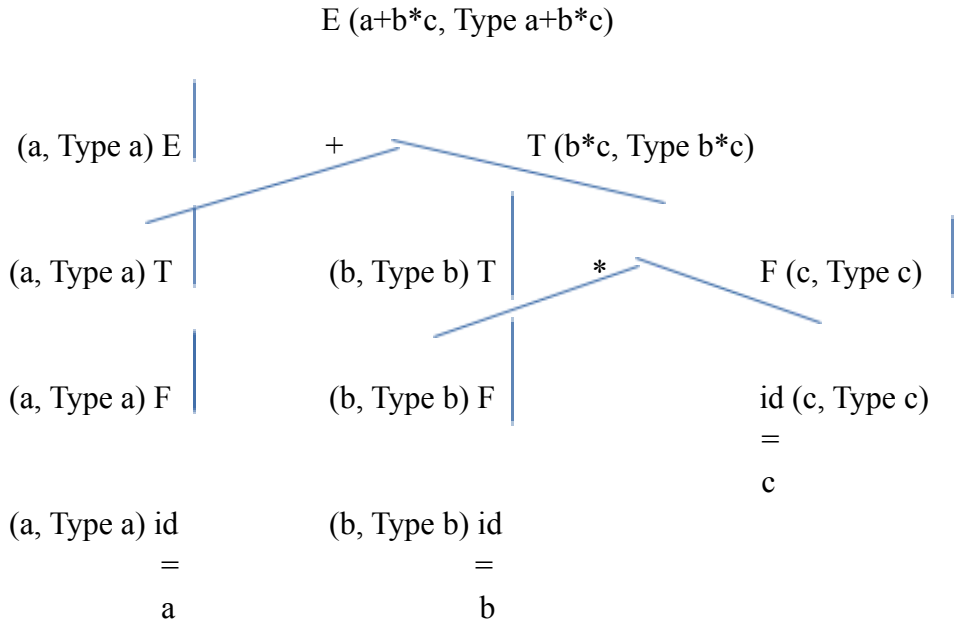
Les routines sémantiques :

```
Routine R1
Debut
    Sauv-BZ := i;
    i :=i+1;
    Pf(i) := BZ;
    i :=i+1;
Fin.

Routine R2
Debut
    Sauv-BR := i;
    i := i+1;
    Pf(i) := BR;
    i := i+1;
    Pf(Sauv-BZ) := i;
Fin.

Routine R3
Debut
    Pf(Sauv-BR) := i;
Fin.
```

II.2.2 Traduction des expressions arithmétiques sous forme des quadruplés dans le cas ascendant :

$$\begin{cases} E \longrightarrow E+T \,/\, T \\ T \longrightarrow T*F \,/\, F \\ F \longrightarrow id \,/\, (E) \end{cases}$$

Exemple : soit la chaine a+b*c



Les attributs concernant les opérandes sont stockés dans la TS. Dans le cas d'une analyse ascendante, donc on effectue des réductions. Mais comme on a besoin des informations concernant les opérandes, en particulier le type, on utilise une pile d'analyse à trois champs (pile des opérandes) :

- 1ère champ contient le MGP.
- 2ième champ contient le nom.
- 3ième champ contient le type.

à chaque symbole de la grammaire sont associés les attributs : nom et type des opérandes, pour la traduction des expressions.

Insertion des routines sémantiques :

E ⟶ E+T **R6** / T **R5**
T ⟶ T*F **R4** / F **R3**
F ⟶ id **R1** / (E) **R2**

Les routines sémantiques :

Routine R1 (*Permet la reconnaissance de l'id, de le réduire en F et d'empiler ses attributs *)
Debut
Lookup(id, P);
Si P=0 Alors 'Erreur : idf non déclaré'
Sinon empiler(pile-opérande, F, P.nom, P.type);
Fsi;
Fin.

Routine R2 (*Permet de réduire (E) en F, et d'empiler ses attributs *)
Debut
opérande := dépiler(pile-opérande);

```
    empiler(pile-opérande, F, opérande.nom, opérande.type);
Fin.

Routine R3 (*Permet de réduire F en T, et d'empiler ses attributs *)
Debut
    opérande := dépiler(pile-opérande);
    empiler(pile-opérande, T, opérande.nom, opérande.type);
Fin.

Routine R4 (*Routine permettant d'effectuer la multiplication et de ranger le resultat *)
Debut
    opérande2 := dépiler(pile-opérande);
    opérande1 := dépiler(pile-opérande);
    Si opérande1.type et opérande2.type sont incompatiblés
                Alors 'Erreur : incompatibilités des types'
                Sinon CréerTemp(R);
                        QUAD(Qc) := (*, opérande1.nom, opérande.nom, R);
                        Qc := Qc+1;
                        CalculType (opérande1.type, opérande2.type, R-Type);
                        empiler(pile-opérande, T, R, R-Type);
    Fsi;
Fin.

Routine R5 (*Permet de réduire T en E, et d'empiler ses attributs *)
Debut
    opérande := dépiler(pile-opérande);
    empiler(pile-opérande, E, opérande.nom, opérande.type);
Fin.


Routine R6 (*Routine permettant d'effectuer l'addition et de ranger le resultat *)
Debut
    opérande2 := dépiler(pile-opérande);
    opérande1 := dépiler(pile-opérande);
    Si opérande1.type et opérande2.type sont incompatiblés
                Alors 'Erreur : incompatibilités des types'
                Sinon CréerTemp(R);
                        QUAD(Qc) := (+, opérande1.nom, opérande.nom, R);
                        Qc := Qc+1;
                        CalculType (opérande1.type, opérande2.type, R-Type);
                        empiler(pile-opérande, E, R, R-Type);
    Fsi;
Fin.
```

II.2.3 Instruction de branchement inconditonnel (traitement des étiquettes) :

Goto étiquette

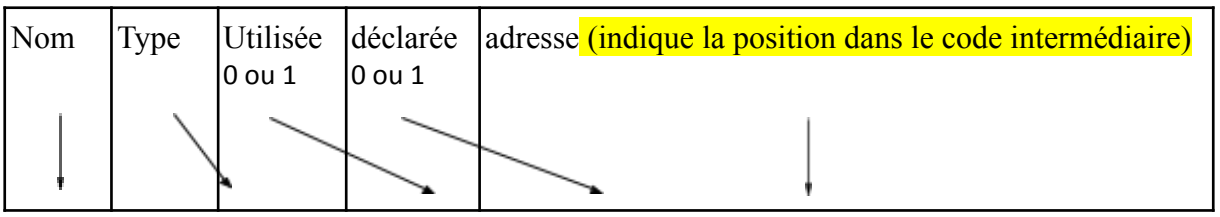
Remarque : Une étiquette peut être déclarée, référencée et utilisée.

1ère cas :	2ième cas :
.	.
.	.
.	.
étiquette : <inst>	Goto étiquette
.	.
.	.
.	.
Goto étiquette	Goto étiquette
.	.
.	.
.	.
Goto étiquette	étiquette : <inst>

- On remarque que dans le 1ère cas, la définition de l'étiquette est connu : étiquette utilisée avant d'être référencée, donc l'opérateur BRL peut être directement remplacé par BR vers début de <inst>.

- Dans le 2ième cas, la définition de l'étiquette n'est pas connu, on ne peut pas donc traduite directement Goto en BR.

- Organisation de la TS dans le cas des étiquettes :



nom de l'étiquette	c'est une étiquette (Label)	utilisée où pas	déclarée où pas	numéro du début de l'instruction étiquetée par l'étiquette dans le code intermédiaire.

- On suppose aussi que l'on dispose de la fonction **Lookup**, permettant la recherche d'une entité dans la TS et la fonction **Insert** permettant l'insertion de l'entité dans la TS.

1ère méthode : BRL P

Traduction de Goto étiquette en BRL P, où P est l'entrée de l'étiquette dans la table des symboles (TS). Deux cas se présentent :

- Etiquette déjà définie au moment de la référence : donc traduire Goto étiquette en BR P.adresse.
- étiquette non connu au moment de la référence : Goto étiquette se traduit en BRL P, qui sera traduit encore lors d'une 2ième passe en BR P.adresse.

2ième méthode : (méthode du chainage des références)

Dans cette méthode, on génère directement BR, au lieu de BRL, on lie tous les BR faisant référence à une même étiquette par la méthode du chainage dans une liste.

Cette liste est créer en utilisant le 2ième champ des quadruplés BR et le champ adresse de l'étiquette dans la TS.

La champ adresse : le numéro du dernier quadruplé qui fait référence à l'étiquette.

Exemple :

.
.
.
Goto etiq ← (20) (BR, 0, ,) ➡ adresse = 20
.
.
.
Goto etiq ← (60) (BR, 20, ,) ➡ adresse = 60
.
.
.
Goto etiq ← (80) (BR, 60, ,) ➡ adresse = 80
.
.
.
etiq : <inst> ← MAJ des BR et MAJ du champ adresse de la TS

Donc à la rencontre d'un Goto vers une même étiquette (etiq), on génère un BR vers le début du dernier Goto rencontré. A la rencontre de l'instruction : etiq :<inst>, on met à jour tous les BR.

Schéma de traduction pour la déclaration, référence et utilisation :

<decl-etiq> → Label <list-etiq>;
<list-etiq> → <list-etiq>,etiq / etiq
<ref-etiq> → Goto etiq
<inst-etiq> → etiq : <inst>

Découpage de la grammaire :

<decl-etiq> ⟶ Label <list-etiq>;
<list-etiq> ⟶ <list-etiq>,etiq **R1** / etiq **R1**
<ref-etiq> ⟶ Goto etiq **R2**
<inst-etiq> ⟶ <debut-inst> <inst>
<debut-inst> ⟶ etiq : **R3**

Ecriture des routines sémantiques :

```
Routine R1 (* Permet d'insérer l'étiquette dans la TS *)
Debut
    Lookup(tc, P);
    Si P=0 Alors Insert(tc, P);
                 P.déclare := 1;
                 P.util := 0;
                 P.type := Label;
                 P.adresse := 0;
           Sinon 'Erreur : double déclaration'
   Fsi;
Fin.
```

```
Routine R2 (* Elle référence une étiquette *)
Debut
    Lookup(tc, P);
    Si P=0 Alors 'Erreur : étiquette non déclarée'
           Sinon Si P.util=1 Alors QUAD(Qc) := (BR, P.adresse,   ,    )
                                   (*étiquette déjà utilisée*)
                                 Sinon QUAD(Qc) := (BR, P.adresse,   ,    );
                                   P.adresse := Qc;
                                   (*étiquette non utilisée*)
                 Fsi;
                 Qc := Qc+1;
    Fsi;
Fin.
```

```
Routine R3 (* Utilisation d'une étiquette *)      etiq2 : y =y+1          ……..
                                                                          ……..
Debut
    Lookup(tc, P);

    Si P=0 Alors 'Erreur : étiquette non déclarée'    Insert( etiq, p) , p.type = label, ….
           Sinon Si P.util=1 Alors 'Erreur : double utilisation'
                                 Sinon a :=P.adresse;                    P. adresse contient
```

```
                                        P.util :=1;
                                        Tant que a≠0
                                            Faire b:=QUAD(a, 2);
                                                  QUAD(a, 2):=Qc;
                                                   a:=b;
                                            Fait;
                        Fsi;
        Fsi;
Fin.
```