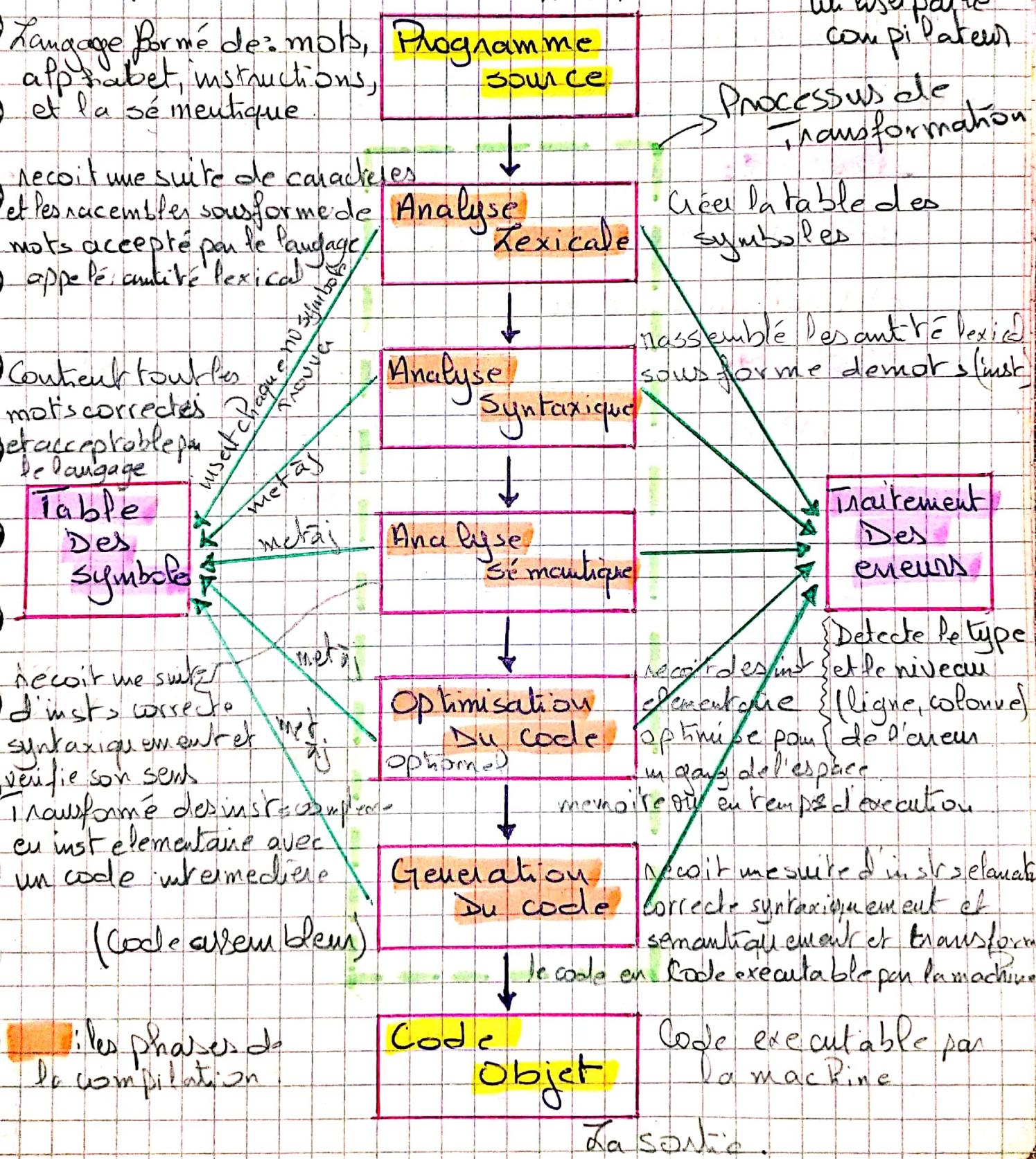


Chapitre 0 : Introduction à la compilation.

Schéma représentant la structure logique d'un compilateur :



Types de compilateur :

→ Compilateur monopasse :

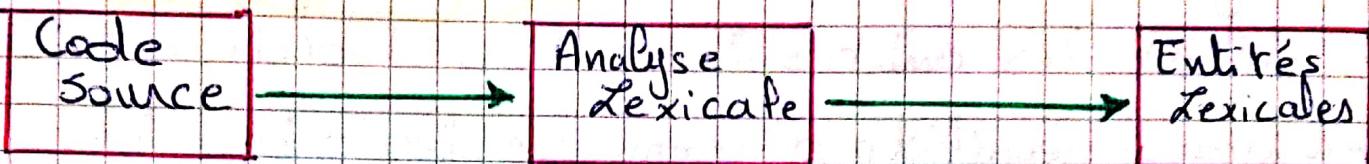
Il passe traiter depuis l'analyse lexicale jusqu'à la génération du code objet en faisant une seule passe (on exécute séquentiellement.)

→ Compilateur multipasse : (2 passes seulement)

- Passe 1 : On fait l'analyse lexicale, syntaxique, sémantique et l'optimisation du code, on obtient un fichier.

- Passe 2 : On génère le code objet en utilisant le fichier obtenu dans le 1er passe.

Chapitre 1 : Analyse Lexicale.



• Ce que fait l'analyse lexicale :

- Créer la table des symboles.
- Éliminer les blancs.
- Éliminer les commentaires (vérifier s'il est correct lexicalement).
- Codier les entités lexicales et les insérer dans la table des symboles en vérifiant s'il existe déjà.

Les idfs, const.
propre au
programme

• Les modes d'entités du langage (type) :

- Les identificateurs.
- Les mots clé.
- Les constantes.
- Les séparateurs.

Mot clé, séparateur:
propre au langage
de programmation

• La table des symboles :

→ Forme 1 :

Pour chaque entité, on a une table. On regroupe toutes les entités.

Table :

Constante	Séparateur
100	,

Mot clé

if

Identificateur
mais

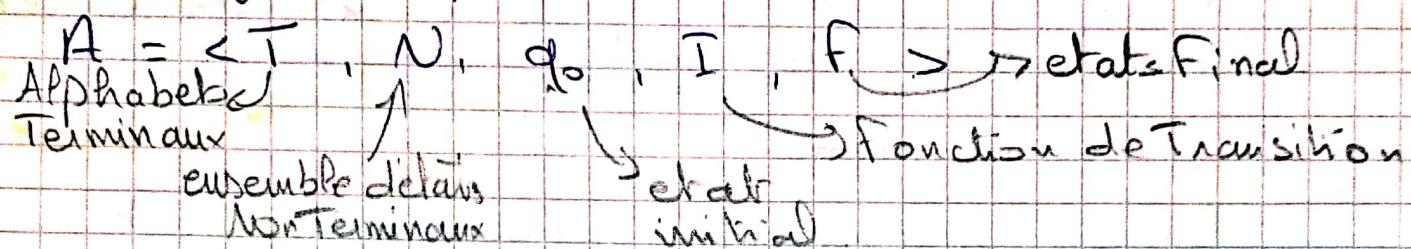
→ Forme 2 :

Mot clé	Type
if	mot clé
100	constante
,	séparateur

• Implementation d'un analyseur lexical :

Il y a des langages où les mots clés ne sont pas réservés

On décrit les unités lexicales à l'aide des grammaires régulières reconnues par les automates d'état fini simple déterministe.



• Algorithme de reconnaissance d'une unité lexicale :

Algorithm : Reconnaissance (E unité)

Début :

Tire (unité);

tc \leftarrow 1er char de l'unité;

TQ (Ec $\neq \emptyset$ et Tfin unité)

Faire:

Ec \leftarrow T[Ec, tc];

tc \leftarrow tc + 1;

Fait

Si (Ec = \emptyset) Alors :

'Erreur: unité incorrecte';

Sinon:

Si (Ec $\in F$) Alors :

"Erreur : entrée incorrecte";

Sinon :

"Entrée correcte"

Si (n'existe pas dans la table des symboles) :
codifier entrée;

insérer dans la table (TS);

Sinon :

renvoyer son code;

F.Si

F.Si

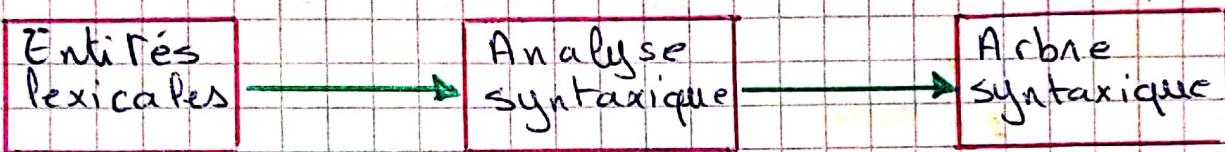
F.Si

Fin

Chapitre 2 : Analyse Syntaxique

- Un langage correct syntaxiquement :

c'est un langage qui a un arbre syntaxique qui le décrit.



- Grammaire syntaxique :

C'est une grammaire formée de règles de production permettant d'engendrer tous les programmes écrits dans un langage donné.

$$G = \langle T, N, S, P \rangle$$

Les terminaux ↗ l'ensemble de production
les non terminaux ↗ l'axiome

$$P: \{ m \in P \rightarrow m_1 p_1 \dots m_n p_n \}$$

membre ↗ membre droit de production
gauche de production ↗ membre droit de production

- Notion de dérivation (Analyse descendante) :

Réplacer un $m \in P$ par un $m_1 p_1 \dots m_n p_n$, en partant de l'axiome jusqu'à avoir la chaîne.

- Notion de réduction (Analyse ascendante) :

Réplacer un $m_1 p_1 \dots m_n p_n$ par un $m \in P$, en partant de la

phrase jusqu'à l'axiome.

• Arbre abstrait :

c'est un arbre syntaxique, où :

- La racine : c'est l'axiome.

- Les noeuds : c'est les mdp.

- Les feuilles c'est les chaînes.

• L'ambiguité :

Une grammaire est ambiguë si pour la même phrase on a deux arbres de dérivation (ou plus)

Dans le terminal, ils ont tous le même de prononcié

• La récursivité :

Associativité gauche

→ Récursivité gauche : On l'élimine

- Directe :

$$A \rightarrow A\alpha \mid \beta$$

- Indirecte :

$$A \rightarrow \beta \mid B\alpha$$

$$B \rightarrow \beta_2 \mid A\alpha_2$$

→ Récursivité droite directe : On ne l'élimine pas

$$A \rightarrow \alpha A \mid \beta.$$

Associativité droite

Eliminer la récursivité gauche directe :

On a : $A \rightarrow A\alpha_1 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \dots \mid \beta_m$

$$\Rightarrow S A \rightarrow \beta_1 \mid \dots \mid \beta_m \mid \beta_1 A' \mid \dots \mid \beta_m A'$$
$$\Rightarrow A' \rightarrow \alpha_1 \mid \dots \mid \alpha_n \mid \alpha_1 A' \mid \dots \mid \alpha_n A'$$

Pour l'ouverte RGFD on applique :

Si : $A \rightarrow B\alpha$ Alors $A < B \Rightarrow$ Si $A < \dots < A$ Alors y une RGFD

• Éliminer la nécessité d'aller à gauche en directe :

1) Transformez en RGID :

$$\begin{array}{l} \text{Ex: } S \rightarrow Aclb \\ \left\{ \begin{array}{l} A \rightarrow Bdla \\ B \rightarrow Acld \end{array} \right. \Rightarrow \left\{ \begin{array}{l} S \rightarrow Acld \\ A \rightarrow Acld \\ B \rightarrow Acld \end{array} \right. \end{array}$$

2) Ensuite on supprime la RGID.

• La factorisation :

Une grammaire non factorisée est une grammaire qui contient plusieurs mdp qui commence de la même manière.

$$\text{Ex: } A \rightarrow \alpha X \mid \alpha y \mid \alpha z \mid t \mid p.$$

$$\Rightarrow \left\{ \begin{array}{l} A \rightarrow \alpha B \mid \alpha \mid \beta \\ B \rightarrow X \mid y \mid z \end{array} \right.$$

Une grammaire doit être factorisée

• Analyse descendante déterministe :

- Grammaire LL(1) : $\rightarrow \text{left left}$.

Def informé

$$\text{Def: } A \rightarrow \alpha \mid \beta, \alpha, \beta \in (T \cup N)^*$$

$\alpha \in T$

① Si: $\alpha \stackrel{*}{\Rightarrow} \alpha \gamma$, Alors: $\beta \stackrel{*}{\not\Rightarrow} \alpha \gamma$ $\beta, \gamma \in (T \cup N)^*$

② Si: $\alpha \stackrel{*}{\Rightarrow} \varepsilon$ Alors: $\beta \stackrel{*}{\not\Rightarrow} \varepsilon$.

③ Si: $\beta \stackrel{*}{\Rightarrow} \varepsilon$ et $\alpha \stackrel{*}{\Rightarrow} \alpha \gamma$ Alors: $S \stackrel{*}{\Rightarrow} w, A \alpha \gamma$

Terminologie:

Si: $\alpha \stackrel{*}{\Rightarrow} \alpha \gamma$, Alors: $\alpha \in T$ est un symbole directeur de α

Ensemble début :

$$\begin{aligned} \text{Début}(\alpha) = & \{ \alpha \in T \mid \alpha \Rightarrow \alpha\gamma, \gamma \in (\text{TUN})^* \} \\ & \cup \{ t \in T \mid \alpha \Rightarrow t \} \end{aligned}$$

Etapes de construction d'un ensemble début :

1) Si : $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$, avec : $A \in N$
 $\alpha_i \in (\text{TUN})^*$

$$\text{Alors : } \text{Début}(A) = \text{Début}(\alpha_1) \cup \dots \cup \text{Début}(\alpha_n)$$

2) Si :

2) - $\text{Début}(\alpha) = \{\alpha\}$, $\alpha \in T$.

3) - $\text{Début}(\alpha\gamma) = \{\alpha\}$, $\alpha \in T$, $\gamma \in (\text{TUN})^*$

4) - $\text{Début}(\varepsilon) = \{\varepsilon\}$

5) - $\text{Début}(Bx_1 \dots x_n) \supset \text{Début}(B) - \{\varepsilon\}$

avec : $B \in N$ et $x_i \in (\text{TUN})^*$

Si : ($\varepsilon \in \text{Début}(B)$) Alors :

T $\text{Début}(Bx_1 \dots x_n) \supset \text{Début}(x_1) - \{\varepsilon\}$

T Si ($\varepsilon \in \text{Début}(x_1)$) Alors :

T $\text{Début}(Bx_1 \dots x_n) \supset \text{Début}(x_2) - \{\varepsilon\}$

T Si ($\varepsilon \in \text{Début}(x_{n-1})$) Alors

T $\text{Début}(Bx_1 \dots x_n) \supset \text{Début}(x_{n-1}) - \{\varepsilon\}$

T Si ($\varepsilon \in \text{Début}(x_n)$) Alors :

F.Si $\varepsilon \in \text{Début}(Bx_1 \dots x_n)$

F.Si F.Si F.Si

- Ensemble suivant :

Suivant(A) = $\{a \in T \cup \{\#\} \mid \exists \xrightarrow{w A a \#} \text{ (} a \in N \text{)}$
avec : $a \in \text{Debut } (\alpha \#)$ et $w, \alpha \in (N \cup \{\#\})^*$

Etapes de construction d'un ensemble suivant :

1) Si : $\exists \text{ MDP} : \alpha A a \beta$

Alors : $a \in \text{Suivant}(A)$, $A, B \in N$, $\overset{a \in I}{\alpha \in \overline{I}}$, $a, \beta \in (I \cup N)^*$

2) Si : $\exists \text{ MDP} : \alpha A B \beta$

Alors : $\text{Debut}(B\beta) - \{\epsilon\} \subset \text{Suivant}(A)$

3) Si : \exists production : $B \rightarrow x_1 \dots x_n A$, $A, B \in N$

Alors : $\text{Suivant}(B) \subset \text{Suivant}(A)$

Si ($A \xrightarrow{\epsilon} \epsilon$)

Alors : $\text{Suivant}(B) \subset \text{Suivant}(x_n)$

Si ($x_n \xrightarrow{\epsilon} \epsilon$)

Alors : $\text{Suivant}(B) \subset \text{Suivant}(x_{n-1})$

⋮

Si : ($x_2 \xrightarrow{\epsilon} \epsilon$)

Alors : $\text{Suivant}(B) \subset \text{Suivant}(x_1)$.

F.Si

F.Si

F.Si

F.Si

Analyse par des procédures récursive ; descente
n associe à chaque non terminal une procédure.

Sit : $A \rightarrow x_1 | x_2 | \dots | x_n$ $x_i \in (T \cup N)^*$

Procédure $A()$

Sebut :

Si ($x_i \in T$) Alors :

Com poser x_i avec t.c.;

Si ($x_i \neq t.c.$) Alors

Erreur;

Sinon

$t.c \leftarrow t.c + 1;$

Passer au traitement de x_{i+1} ;

F.Si

Sinon.

Appel à la procédure de x_i ;

Passer au traitement de x_{i+1} .

F.Si

in

→ Analyse LL(1) :

~~Analyse de contexte~~

Algorithm de construction de table d'analyse LL(1).

Début :

Pour (chaque non-terminal $A \in N$)

Faire :

Pour (chaque règle $A \rightarrow \alpha$)

Faire :

Pour (chaque $a \in \text{Debut}(\alpha) - \{\epsilon\}$)

Faire :

$T[A, a] = N^o$ de la règle $A \rightarrow \alpha$;

Fait

sinon ($\epsilon \in \text{Debut}(\alpha)$) Alors :

Pour (chaque $a \in \text{Suivant}(N)$)

Faire :

$T[A, a] = N^o$ de la règle $A \rightarrow \alpha$;

Fait

F.Si

Faut faire Soit être
Faut monodéfinie
Fini règle par entrée

Chaque entrée
monodéfinie de la
table d'analyse LL(1)
est unique

Grammaire LL(1) :

Elle doit être : non récursive, facteurisé,
à une table d'analyse LL(1) monodéfinie

Elle Doit être :

- Non récursive ou anche Factorisé

$$A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$$

↳ Debut(α_i) \cap Debut(α_j) = \emptyset , $\forall i, j$ et $i \neq j$

↳ Si $\alpha_i \rightarrow E$: Debut(α_j) \cap Suivant(A) = \emptyset , $\forall j \neq i$

↳ Si $\alpha_i \rightarrow E$: $\alpha_j \not\rightarrow E$, $\forall j$

- La table d'analyse LL(1) :

C'est une matrice de n lignes et m colonnes.

→ n = nombre de non terminaux

		Terminaux
Nb.	Terminaux	La n ^e de la règle
		Terminaux

→ m = nombre de terminaux

- Analyse Ascendante :

→ Soit : $G = \langle T, N, S, P \rangle$, une chaîne x est une forme sentencielle, si : $S \xrightarrow{\text{axiom}} x \quad x \in (T \cup N)^*$

→ Soit : $G = \langle T, N, S, P \rangle$, et $w = xy$ une forme sentencielle, il est une phrase simple de la forme sentencielle w relativement au non terminal U ,

si : $S \xrightarrow{\text{L}} xUy \xrightarrow{\text{R}} xuy$ (UGD de $U \rightarrow u$)

→ Une réduction consiste à remplacer une phrase simple u dans une forme sentencielle $w = xuy$ par U pour avoir xUy (UGD de $U \rightarrow u$)

La forme sentencelle initiale est la chaîne d'entrée

R₄

Les réductions se font de gauche à droite si les phrases simples les plus à gauche sont réduites avant les autres.

À la réduction de u en v, x ne doit pas contenir de phrases simples.

→ L'analyse LR :

L : le sens des parcours de l'entrée gauche → droite.

R : en construisant une déivation droite inverse.

Elle utilise une pile et une table d'analyse LR

1) - Méthode des items :

Un item LR(1) est de la forme :

[A → α.β, E]

ce qui a été déjà réduit

ce qui reste à analyser

A → αβ : négatif de G_n.

α ∈ A ∈ N, β ∈ (UN)*, E ∈ Début (Π^q, #)

Pour construire l'ensemble d'items :

- La fermeture d'un ensemble d'items LR(1) :

Fonction fermeture (I) :

Début :

Repéter :

Pour chaque item de la forme [A → α.Bβ, E] de l'ensemble des items I)

Faire :

 Pour (chaque nœud $B \rightarrow \gamma$)

 Faire :

 Pour (chaque $t \in \text{Déb}(B\beta t)$)

 Faire :

 Ajouter dans I l'item $(B \rightarrow \gamma, t)$;

 Fait

 Fait

 Fait

Jusqu'à ce qu'il n'y ait plus d'item à rajouter de I .

fermiture (I) : = I ;

in

onctis Goto (I, x) \rightarrow symbole de la grammaire
 $x \in (T \cup N)$

sebut :

$j^0 = \emptyset$;

Pour chaque item de I de la forme $(A \rightarrow \alpha \cdot x \beta, t)$

Faire :

$j := j \cup \{ A \rightarrow \alpha \cdot x \beta, t \}$;

 Fait

 Goto (I, x) := fermiture (j) ;

Fin

Construction de l'ensemble canonique des items LR(1)

ensemble des états de l'automate.

Début :

$$I_0 = \{ [x \rightarrow -s, \#] \};$$

$$C = \{ \text{fermeture}(I_0) \};$$

Repéter :

Pour chaque ensemble d'items I de C)

Faire :

Pour chaque $x \in T \cup N$)

Faire :

$$J := \text{Goto}(I, x);$$

Si ($J \neq \emptyset$) Alors :

Ajouter J dans C ;

F.Si

Fait

Fait

Jusqu'à ce qu'il n'y ait plus d'élément à ajouter dans C .

Fin.

Implementation de la méthode LR pour la méthode des items :

- Construction de l'ensemble canonique des items C.

- ① : Si $I_j = \text{Goto}(I_i, A)$, $A \in N \Rightarrow I[I_i, A] := I_j$.

② : Si $I_j = \text{Goto}(I_i, a)$; $a \in T$

$\rightarrow I[I_i, a] = D, F_j$ décalage d'un caractère

③ : Si dans I_i , on a des items de la forme $[A \rightarrow \alpha, T]$

$\Rightarrow T[I_i, T] := N^0 = A \rightarrow \alpha. T \rightarrow D, I_i$
 $N \rightarrow I_i$

Du \Rightarrow G est LR(1),ssi : la table LR(1) déduite de l'ensemble canonique d'items C est monodéfinie

\rightarrow L'analyse SLR(1).

Chaque case contient au plus une règle

C'est une analyse LR(0), où

\rightarrow On utilise l'ensemble tout

on prend pas d'élément de la chaîne (pas de partie droite dans les items).

NUT

C Règle | $D + I_i | T$
Acc

\rightarrow L'analyse LALR(1) :

C'est une analyse LR(1), avec fusion d'items :

$I_i = [S \rightarrow a.AB, aJ, [A \rightarrow \cdot, aC]D, \#J, [B \rightarrow A.Ca, bJ]$

$I_j = [S \rightarrow a.AB, bJ, [A \rightarrow \cdot, aC]D, cJ, [B \rightarrow A.Ca, \#J]$

$\Rightarrow I_i \cup I_j = [S \rightarrow a.AB, a|bJ, [A \rightarrow \cdot, aC]D, \# | cJ, [B \rightarrow A.Ca, b | \#J]$

Une grammaire est LR(1) | SLR(1) | LALR(1),ssi

la table correspondante est monodéfinie

Chapitre 3 : Les formes intermédiaires

• Forme postfixée :

opérande opérande opérateur

→ La FP d'une opération :

$$\text{FP}(t_1 \text{ opérateur } t_2) = \text{FP}(t_1) \text{ FP}(t_2) \text{ opérateur}$$

$$\text{FP}(\text{opérateur } t) = \text{FP}(t) \text{ opérateur}$$

$$\text{FP}(\text{cste}) = \text{cste}$$

$$\text{FP}(\text{Var Simple}) = \text{Var Simple}$$

On utilise une pile, on empile les opérandes, dès qu'un nouveau opérateur, on dépile 1 ou 2 opérandes selon le type d'opérateur.

Associativité gauche → droite

$$\begin{array}{c} + - \div \times \text{OR} \\ \hline + - \end{array} \quad \begin{array}{c} \text{And} \\ \hline + \end{array} \quad \begin{array}{c} \text{Not} \\ \hline + \end{array}$$

→ La FP d'une affectation :

$$\text{Var} := \langle \text{expression} \rangle \Rightarrow \text{FP}(\text{Var}) \text{ FP}(\text{expression}) :=$$

→ La FP d'un branchement :

- Inconditionnel (to Label)

Gbts eliq → eliq BR

(ne connaît pas la position (ou a pas encore passé par eliq)).

opérande BR, on connaît la position (ou

a déjà passé par eliq)

position dans la chaîne Postfixée

- Conditionnel :

opérande1 opérande2 opérateur-de-Cond-à-0

bool = 0 : Faut

BP(>0) BPZ(>0)

bool = 1 : Unai

BG(<0) BGZ(<0)

BZ(=0) BNZ(≠0)

→ La FP des Tableaux:

- Déclaration:

Array V [U₁:L₁, ..., U_n:L_n]

U_i, L_j: bonne inf et sup

⇒ FP(U₁) FP(L₁) ... FP(U_n) FP(L_n) N ADSC.

- Référencé un élément:

A[exp₁], ..., [exp_n]

⇒ FP(exp₁) ... FP(exp_n) A SUBS.

→ La FP d'une instruction conditionnel

IF <cdts> then <inst1> else <inst2>

⇒ FP(cdt) else B ↗ FP(inst1) Fin BR FP(inst2) ↘

• Forme préfixé:

opérateur opérande opérande

• Les quadriplés:

(opérateur, opérande₁, opérande₂, temporaire)

→ Affectation:

Var := <expressions>

[Quadriplé(<expressions>) →, T

(:=, T, , Var)

→ Branchement:

- Inconditionnel (to Label):

Goto etiq → (BR, etiq, ,)
(BR, N=quadriplé, ,)

• Fonctionnel :

(BE [BNZ, BM, BYZ, BP, BPZ], Nquad, Temps,)
" " + < ≤ ≥ >)

↳ Unaire

(BE [BNZ, BL, BLE, BG, BGZ], Nquad, T₁, T₂)
" " + < ≤ ≥ >)

↳ Binaire

→ Les tableaux :

• Déclaration :

A may AT[U₁:L₁, ..., U_n:L_n]

[Quadriplés (U₁) → T_{1.1}

[Quadriplés (L₁) → T_{1.2}

(Bounds, T_{1.1}, T_{1.2},)

[Quadriplés (U_n) → T_{n.1}

[Quadriplés (L_n) → T_{n.2}

(Bounds, T_{n.1}, T_{n.2},)

(ADEC, A, ., .)

• Référencé un élément :

A T[exp1], ..., (expn]

[Quadriplé <exp1> → T₁

[Quadriplé <expn> → T_n

(Opérateur, A[T₁, ..., T_n], opérande, résultat)

→ Instruction conditionnelle :

IF $c\text{cdt} \rightarrow T$ THEN $c\text{inst1} \rightarrow E\text{LSE} \rightarrow c\text{inst2}$

[Quaduplé ($c\text{cdt} \rightarrow T$)]

(BZ , ELSE, T ,)

[Quaduplé ($c\text{inst1} \rightarrow$)]

(BR , FIN, ,)

[Quaduplé ($c\text{inst2} \rightarrow$)]



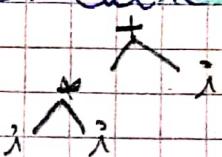
• Arbre Abstrait

Arbre syntaxique

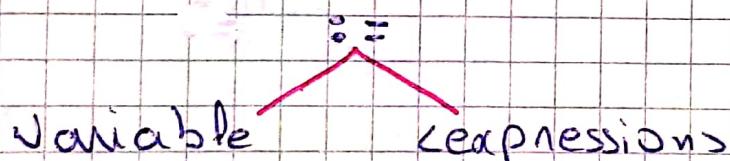
sous N

(vars)

Les non terminaux de la grammaire sont supprimés, on conserve que les terminaux (les relations entre entree opérateur et opande). $i * i + 1$

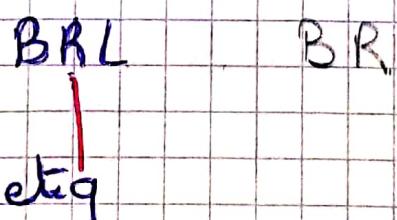


→ L'affectation :



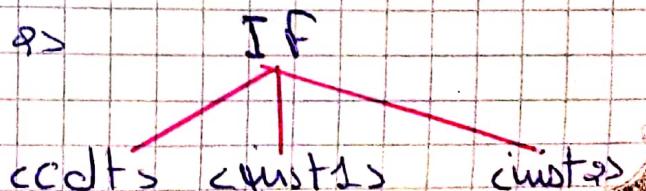
→ Branchement有条件 (To Label) :

Goto etiq :

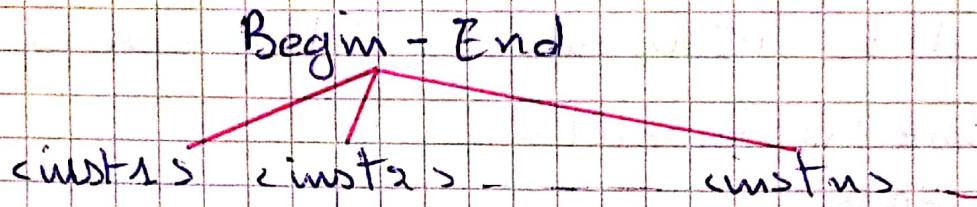


→ L'instruction conditionnel If :

If $c\text{cdt} \rightarrow T$ Then $c\text{inst1} \rightarrow E\text{lse} \rightarrow c\text{inst2}$



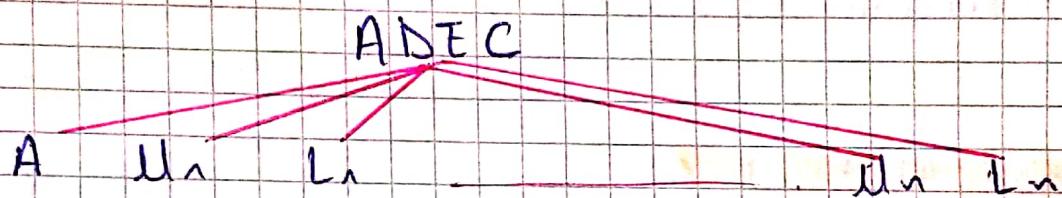
→ L'instruction: Begin <inst1><inst2> ... <instn> End:



→ Les tableaux:

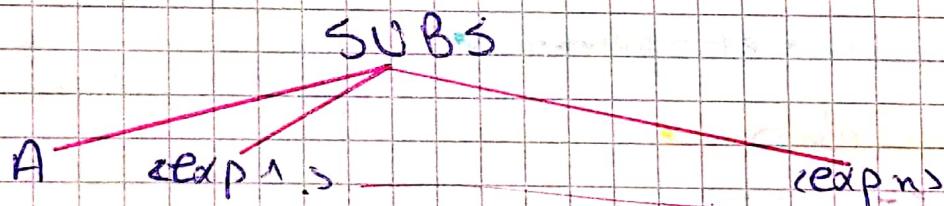
- Déclaration:

Array A [U₁:L₁, ..., U_n:L_n];



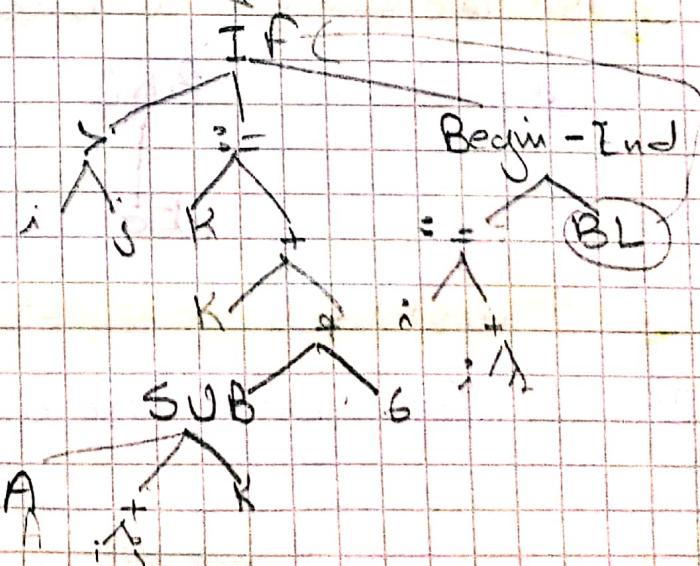
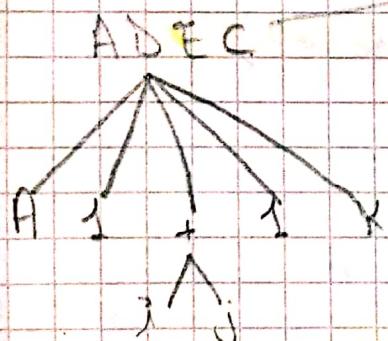
- Référence à un élément:

A [<exp1>, ..., <expn>]



Exp cours:

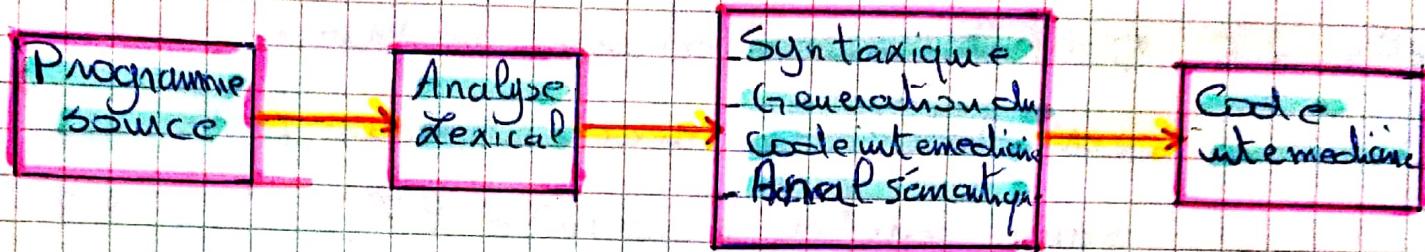
Begin - End



Chapitre 4:

Traduction dirigée par la syntaxe

Com 2



Le schéma de traduction :

- Grammaire associée (transformée)

- Code intermédiaire

- Routines sémantiques

L'analyse descendante :

Les routines sémantiques correspondant aux dérivations.

Si : $A \rightarrow \alpha \beta$, on insert une routine sémantique.

Absr : $A \rightarrow \alpha B \beta$

On introduit des non terminaux

$B \rightarrow \epsilon$

dérivant en ϵ .

Instruction conditionnelle :

<inst-if> → IF <exp-boole> THEN <insts> ELSE <insts>

- Forme intermédiaire (quaduplets):

[Quaduplé de <exp-boole>

(BZ, else, <exp-boole>, tmp,)

[Quaduplé de <insts>

(BR, Fin, ,)

[Quaduplé de <insts>

Inclusion des routines sémantiques :

<inst_if> → IF <exp_bool><A> Then <insts> else <insts><C>

<A> → E : permet la génération d'un branchement vers début else, si <exp_bool> = false, sauvegarder l'étiquette BZ.

 → E : génération d'un BR, sauvegarder l'étiquette BR, mettre à jour l'étiquette BZ.

<C> → E : mette à jour l'étiquette BR.

- Les routines sémantiques :

Routine <Ass>

QUAD : matrice de quadruplets

QUAD(qc) := (BR, , <exp_bool>.tmp,)

Sauv_BZ := qc ;

qc : 1er quadruplet

libre dans la matrice

QUAD

qc := qc + 1 ;
↳ aller au quadruplet suivant.

Routine

Matrice QUAD

QUAD(qc) := (BR, , ,);

qc ->

Les quadruplets de <ccdt>

Sauv_BR := qc ;

qc ->

de <inst>

qc := qc + 1 ;

qc ->

QUAD(Sauv_BR, 2) := qc ;

Les quadruplets de <inst>

Mettre à jour l'étiquette BR

qc ->

de <inst>

Routine <C>

Matrice QUAD

QUAD(Sauv_BR, 2) := qc ;

qc ->

Les quadruplets de <inst>

mettre à jour l'étiquette BR

qc ->

de <inst>

lors fin de BR

Matrice QUAD

→ Instruction While :

<inst-white> → WHILE <cond> Do <insts>

- Forme intermédiaire (quadriplés) :

[Quadriplé deconds]

(BF, Fin, <cond>, tmp,)

[Quadriplé deinsts]

(BR, Debut, ,)

- Grammaire transformée (associée) :

<inst-white> → WHILE <A> <conds> B Do <insts> <C>

<A> → E : Sauvegarder le début while pour y revenir.

 → E : Tester la condition.

<C> → E : Générer un branchement BR au début du while et mettre à jour QC.

- Les routines sémantiques :

Routine <A> {

Sauv-deb := qc;
qc = qc + 1;

Routine {

QUAD(qc) := BF, , <cond>, tmp,)

Sauv-BF := qc;

c := qc + 1;

Routine <C> {

QUAD(qc) := (BR, Sauv-deb, ,)

qc = qc + 1;

QUAD(Sauv-BF, 2) := qc;

qc →					
qc →	Les quadriplés deconds				
qc →	BR	Apres test			
qc →					
120 - qc →	Les quadriplés deinsts				
120 - qc →	BR	Sauv debut			

$$\begin{cases} E \rightarrow T + \{T\}^* \\ T \rightarrow F * \{F\}^* \\ F \rightarrow i \mid (E) \end{cases}$$

- Insertion des procédures :

$$\begin{cases} E \rightarrow T + \{T \langle A \rangle\}^* \\ T \rightarrow F * \{F \langle B \rangle\}^* \\ F \rightarrow i \mid (E) \\ \langle A \rangle \rightarrow E \\ \langle B \rangle \rightarrow E \end{cases}$$

- Analyse syntaxo-sémantique par la descente récursive:

Procédure E (X : entité, Y : Type)

Début :

Vari : opérande1, opérande2 : entité ;

Type1, Type2 : Type ;

T(opérande1, Type1) ;

Si (enem) Alors : Appeler à Fin F.Si ;

Tant que ($tc = '+'$) .

Faire :

$tc := tc + 1;$

T (opérande2, type2);

Si (enem) Alors : Appeler à Fin F.Si ;

A (opérande1, type1, opérande2, type2);

Fait ,

$x := opérande1;$

$y := type1;$

un

Procédure T (x : entrée ; y : Type)

but :

Nous, opérande1, opérande2 : entrée ;

Type1, type2 : Type ;

F (opérande1, Type1);

Si (enem) Alors : Appeler à Fin F.Si ;

Tant que ($tc = '*'$) .

Faire :

$tc := tc + 1;$

F (opérande2, Type2);

Si (enem) Alors : Appeler à Fin F.Si ;

B(operand1, type1, operand2, type2);

Fait;

X := operand1;

Y := type1;

Fin.

Procédure F (X:entité, Y:type)

Début:

Si (tc = '(') Alors :

tc := tc + 1;

E(X, Y);

Sai (erreur) Alors : Aller à Fin F.Si;

Si (tc = ')') Alors :

tc := tc + 1;

Sinon :

erreur := Vrai;

F.Si

Sinon :

lookup(tc, P);

F.Si

Si (P ≠ 0) Alors : Fin

X := P.nom;

Y := P.type;

tc := tc + 1;

Sinon :

"erreur := f non décl"

erreur := Vrai;

F.Si

rebut :

Si (Type1 = Type2) Alors Vérifier la compatibilité des types
CréerTemp(R); créer un temporaire

QUAD(qc) := (+, opde1, opde2, R); créer un quaduplet
 $qc := qc + 1$

Calcultype(Type1, Type2, R-Type); Calculer le type
des opérande

opde1 := R;

Type1 := R-Type;

Sinon :

"Erreur : Incompatibilité de types";

F.Si

in

procédure (opde1, opde2 : entité, type1, type2 : type)

but :

Si (Type1 = Type2) Alors Vérifier la compatibilité des types

CréerTemp(R); créer un temporaire

QUAD(qc) := (<, opd1, opd2, R); créer un quaduplet

$qc := qc + 1$; appeler la procédure case librairie de QUAD

Calcultype(Type1, Type2, TypeR); Calculer les types d'opéra-

opd1 = R; Type1 = TypeR;

Sinon :

"Erreur : incompatibilité de types";

F.Si

• L'analyse ascendante :

Décomposition de la grammaire :

Si : $\langle A \rangle \rightarrow \alpha \beta$

Alors : $\langle A \rangle \rightarrow \langle B \rangle \beta$

$\langle B \rangle \rightarrow \alpha$

→ Instruction conditionnelle :

$\langle \text{inst_if} \rangle \rightarrow \text{IF } \langle \text{exp_bool} \rangle \text{ Then } \langle \text{inst} \rangle \text{ Else } \langle \text{inst} \rangle$

Forme intermédiaire postfixée :

$\text{PF}(\langle \text{exp_bool} \rangle) \text{ else } B \not\in \text{PF}(\langle \text{inst1} \rangle) \text{ fin } BR \text{ PF}(\langle \text{inst2} \rangle)$

Grammaire transformée :

$\langle \text{inst_If} \rangle \rightarrow \langle \text{deb_inst_if} \rangle \langle \text{inst} \rangle \quad \textcircled{3}$

$\langle \text{deb_inst_if} \rangle \rightarrow \langle \text{deb_if} \rangle \langle \text{inst1} \rangle \text{ else } \textcircled{2}$

$\langle \text{deb_if} \rangle \rightarrow \text{IF } \langle \text{exp_bool} \rangle \text{ Then } \textcircled{1}$

Les routines sémantique :

Routine 1 {

Sauv-BR := i;

i := i + 1;

PF(i) := 'BR';

i := i + 1;

PF ; vecteur de
la forme postfixée

1 : 1 ère position de PF

Routine 2 {

Sauv-BR := i;

i := i + 1;

PF(i) := 'BR'; i := i + 1;

PF(Sauv-BR) := i;

Routine 3 {

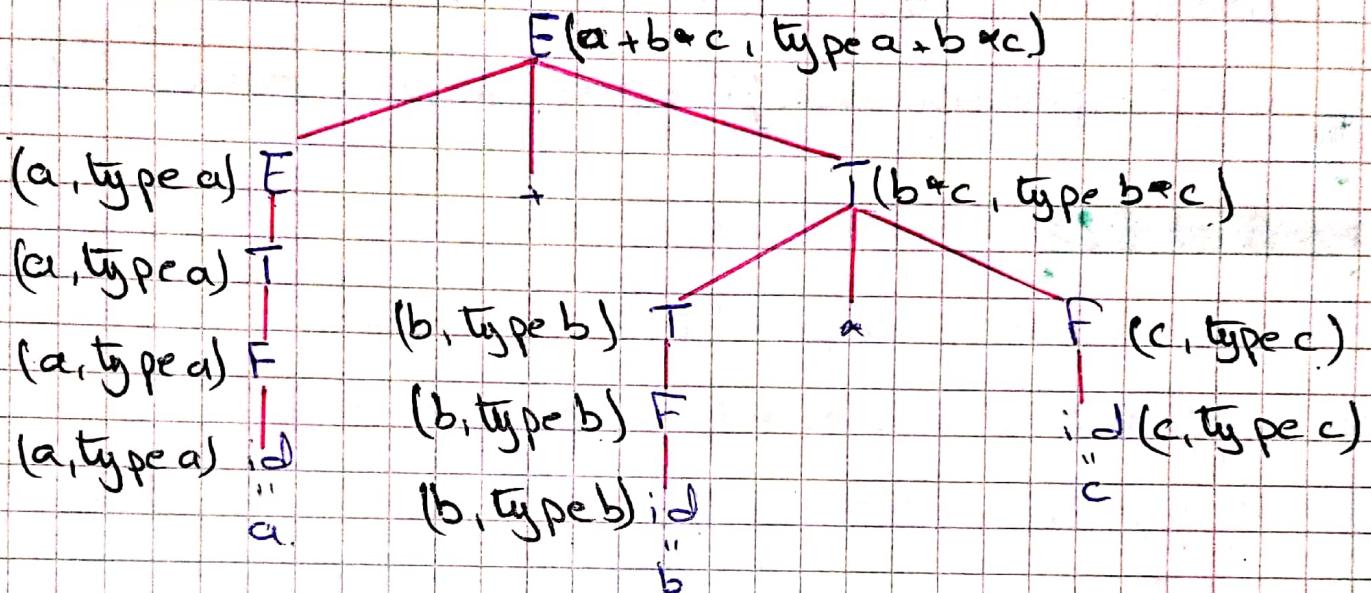
PF(Sauv-BR) := i;

→ Expressions arithmétiques:

$$\begin{cases} E \rightarrow E + T \mid T \\ T \rightarrow T * F \mid F \\ F \rightarrow \text{Id} \mid (E) \end{cases}$$

A chaque réduction
on associe une
routine

- L'arbre : $a + b * c$



- Insertion des routines sémantique :

$$\begin{aligned} E &\xrightarrow{R_E} E + T \mid T R_E \\ T &\xrightarrow{R_T} T * F \mid F R_T \\ F &\xrightarrow{R_F} \text{id} \mid (E) R_F \end{aligned}$$

- Les routines sémantique

Routine R_{NS}

`lookup(id, P);`

Si ($P = 0$) Alors : "Erreur : id f non déclaré";

Sinon : empiler (pile-operande, F, p.nom, p.type);

F.Si

Routine R₂

opérande := dépiler (pile-opérande);
empiler (pile-opérande, f, opérande.nom, opérande.type);

Routine R₃

opérande := dépiler (pile-opérande);
empiler (pile-opérande, T, opérande.nom, opérande.type);

Routine R₄

opérandes2 := dépiler (pile-opérande);

opérande1 := dépiler (pile-opérande);

Si (opérande1.type et opérandes2.type in compatible) Alors:
T "Erreur, incompatibilité de types";

Sinon:

CreerTemp (R);

QUAD (qc) := (*, opérande1, opérandes2, R); qc++;

Calculer type (opérande1.type, opérandes2.type, R.type);

Empiler (pile-opérande, T, R, R.type);

F.S.

Routine R₅

opérande := dépiler (pile-opérande);

empiler pile-opérande, E, opérande.nom, opérande.type);

Routine R

operaude2 := depiler(pile-operande);

operaude1 := depiler(pile-operande);

Si (operaude1.type et operaude2.type incompatible) Alors

 T "Erreur, incompatibilité de types";

Sinon :

 CreerTemp(R);

 QUAD (qc) := (+, operaude1, operaude2, R);

 qc := qc + 1;

 CalculeType (operaude1.type, operaude2.type, R.type);

 Empiler (pile-operande, F, R, R.type);

F.Si

→ Instruction de branchement inconditionnel:

nom	type	Util.	déclaré	adresse
		011	011	

N° debut
de l'étiquette

nom de Label si elle est déclaré
l'étiquette ou pas ou pas

Méthode d'chainage des références:

On utilise que BR, et on lie tous les BR de la

mem en par la méthode de chainage

@'(BR, @, .)

N° dernier quad qui fait référence à chq

• Schéma de traduction pour la déclaration, référence et utilisation:

<dec-ctqs> → Label <list-ctqs>
<list-ctqs> → <list-ctqs>, etiq / etiq
<ref-ctqs> → Goto etiq

<inst-ctqs> → etiq : <insts>

• Déroulage de la grammaire :

<dec-ctqs> → Label <list-ctqs>
<list-ctqs> → <list-ctqs>, etiq R₁ / etiq R₁
<ref-ctqs> → Goto etiq R₂

<inst-ctqs> → <debs-insts> <insts>

<debs-insts> → etiq : R₃

• Les routines sémantiques :

Routine R₁:

Lookup (tc, P);

Si (P = 0) Alors:

Insert (tc, P), P

P.declare := 1; P.whl := 0; P.type := Label;

P.@ := 0;

Sinon :

"Erreur : double déclaration";

F.Si

Lookup : recherche une autre dans la TS

Insert : inscrit l'autre dans la TS

Routine REG

Lookup (tc, P);

Si ($P = \emptyset$) Alors :

| "Erren : étiquette non déclaré";

sinon :

| Si ($P.\text{util} = 1$) Alors :

| | QUAD(qc) := (BR, P.@, ., .);

| | Sinon :

| | QUAD(qc) := (BR, P.@, ., .);

| | | P. @ := qc;

| | | qc := qc + 1;

| F.Si

Routine REG

Lookup (tc, P);

Si ($P = \emptyset$) Alors : "Erren : étiquette non déclaré";

sinon :

| Si ($P.\text{util} = 1$) Alors : "Erren : double déclaration";

| | Sinon :

| | | a := P.@; P.util := 1;

| | Tant que ($a \neq 0$) faire :

| | | b := QUAD(a, 2); QUAD(a, 2) := qc; a := b;

| | F. Si Faut