Plan

I. Code intermédiaire

II. Code à trois adresses

1. Types d'énoncés
2. Instructions

III. Structures de données de sauvegarde de la représentation

1. Quadruples
2. Triplet
3. Exercices

IV. Traduction des expressions

1. Symboles des règles sémantiques
2. Traduction des expressions
3. Exercices

Plan

I. Code intermédiaire

II. Code à trois adresses

1. Types d'énoncés
2. Instructions

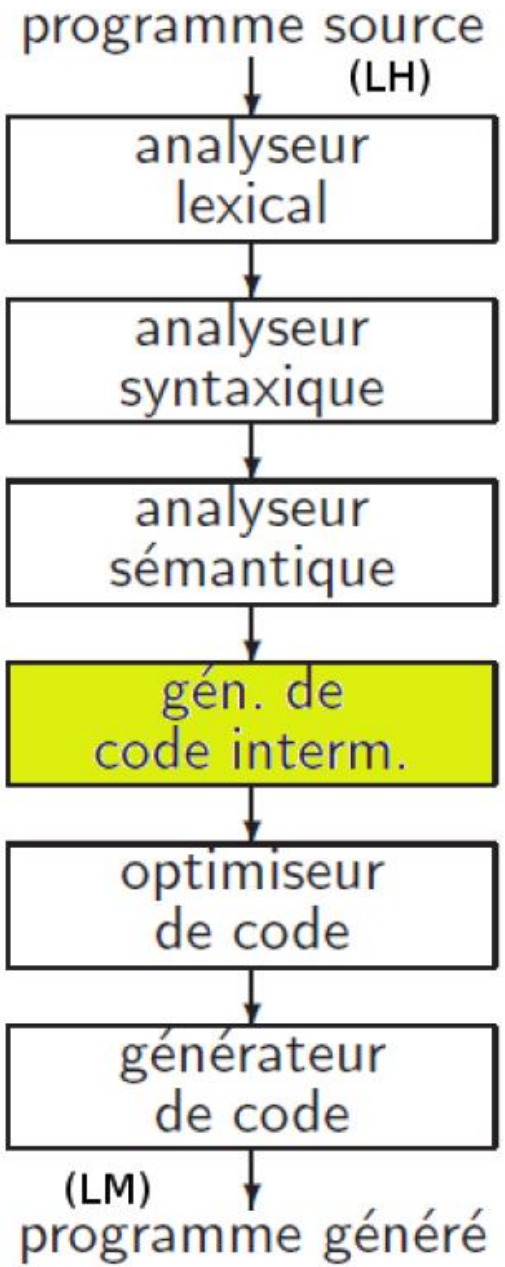
III. Structures de données de sauvegarde de la représentation

1. Quadruples
2. Triplet
3. Exercices

IV. Traduction des expressions

1. Symboles des règles sémantiques
2. Traduction des expressions
3. Exercices

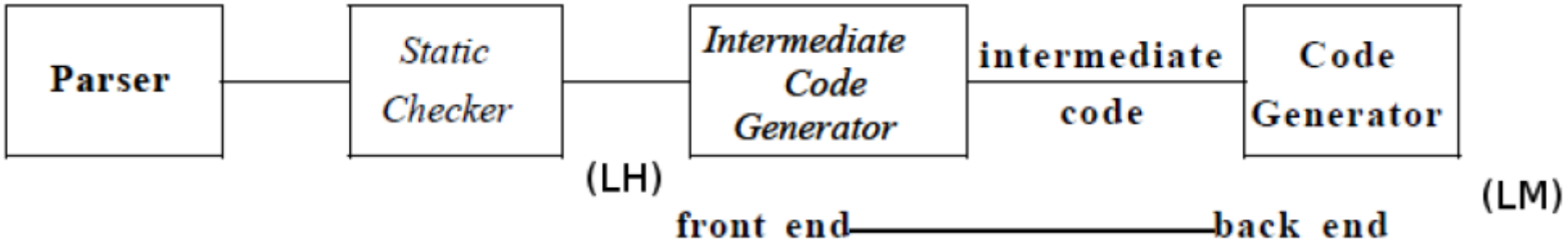
Rappel


programme source
(LH)
analyseur lexical
analyseur syntaxique
analyseur sémantique
gén. de code interm.
optimiseur de code
générateur de code
(LM)
programme généré

I- Code intermédiaire

- On peut convertir un langage haut-niveau directement vers un langage machine spécifique à une architecture matérielle sans passer par un code intermédiaire.
- Dans ce cas, s'il existe n langages de programmation de haut-niveau et m architectures matérielles différentes, il faut créer $n \times m$ conversions.
- Par contre, en passant par un code intermédiaire, il faut créer juste $n + m$ conversions.
- La représentation intermédiaire permet aussi d'appliquer des optimisations indépendantes du langage et de l'architecture cible.

I- Code intermédiaire

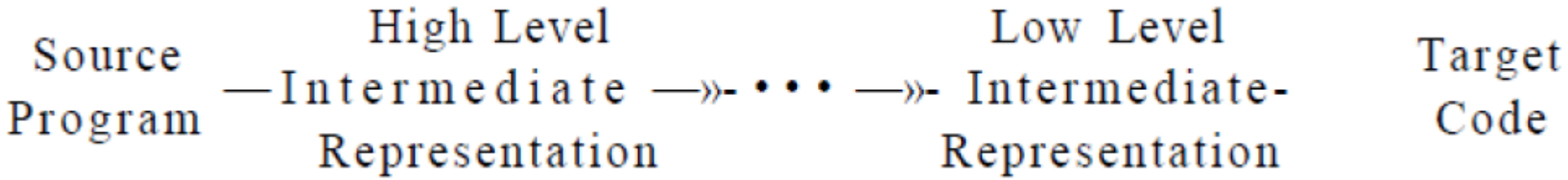


- **Front end**: du langage haut-niveau vers code intermédiaire
- **Back end**: du code intermédiaire vers langage machine

Il existe multiples représentations et différents niveaux de représentations du code intermédiaire.

I- Code intermédiaire

- Lors de la traduction d'un programme dans un langue haut-niveau donné en un code pour une machine cible donnée, un compilateur peut construire une séquence de représentations.



- Les représentations de haut niveau sont proches du langage source et les représentations de bas niveau sont proches de la machine cible.

- la représentation du **code à trois adresses** est une représentation qui peut aller du haut niveau au bas niveau dépendamment du choix des opérateurs utilisés.

Plan

I. Code intermédiaire

II. Code à trois adresses

1. Types d'énoncés
2. Instructions

III. Structures de données de sauvegarde de la représentation

1. Quadruples
2. Triplet
3. Exercices

IV. Traduction des expressions

1. Symboles des règles sémantiques
2. Traduction des expressions
3. Exercices

II- Code à trois adresses

- Le code à trois adresses est un langage très simple, utilisé comme langage intermédiaire.

- La restriction principale de ce langage est que chaque instruction ne concerne **au plus** que 3 variables : une variable de destination du résultat et deux variables dont la valeur est utilisée.

- il s'agit de séquences d'énoncés simples dont les plus courants sont de la forme "$x := y\ op\ z$" (2 opérandes + 1 receveur = 3 adresses);

II- Code à trois adresses

- Le découpage des expressions complexes du langage source en des séquences d'énoncés simples demande l'introduction de variables temporaires t_i.

- Exemple: représenter en code à trois adresses l'instruction suivante:

$$a := b + c + d$$

II- Code à trois adresses

- Exemple: représenter en code à trois adresses l'instruction suivante:

$$a := b + c + d$$

$t_1 = b + c$

$t_2 = t_1 + d$

$a = t2$

II- Code à trois adresses

<u>Exercice:</u>

1. Représenter en code à trois adresses l'instruction suivante:

$$a := (b * -c) + (b * d)$$

2. Donner l'instruction de la représentation en code à trois adresses suivante:

$t_1 = b - c$
$t_2 = a * t_1$
$t_3 = a + t_2$
$t_4 = t_1 * d$
$t_5 = t_3 + t_4$

II- Code à trois adresses

Exercice:

1. Représenter en code à trois adresses l'instruction suivante:

$a := (b * -c) + (b * d)$

$t_1 = -c$
$t_2 = b * t_1$
$t_3 = b * d$
$t_4 = t_2 + t_3$
$a = t_4$

2. Donner l'instruction de la représentation en code à trois adresses suivante:

$t_1 = b - c$
$t_2 = a * t_1$
$t_3 = a + t_2$
$t_4 = t_1 * d$
$t_5 = t_3 + t_4$

$a * (b - c) + (b - c) * d$

II- Code à trois adresses

Le choix du jeu d'instructions de la représentation intermédiaire doit être fait judicieusement.

- Il doit être assez riche pour pouvoir exprimer tous les calculs possibles dans le langage source.
- S'il est trop étendu, le générateur de code devient alors lourd; tout changement de langage cible est coûteux.
- S'il est trop restreint, les séquences de code intermédiaire à générer deviennent plus longues et la réalisation des optimisations subséquentes est plus difficile.

Plan

I. Code intermédiaire

II. Code à trois adresses

1. Types d'énoncés
2. Instructions

III. Structures de données de sauvegarde de la représentation

1. Quadruples
2. Triplet
3. Exercices

IV. Traduction des expressions

1. Symboles des règles sémantiques
2. Traduction des expressions
3. Exercices

II-1. Types d'énoncés

Les types d'énoncés du code à trois adresses :

- *x := y op z* où *op* est un opérateur binaire (add, etc.) et x, y et z sont des adresses;
- *x := op y* où *op* est un opérateur unaire (not, negatif, etc.);
- *x := y ;*
- *goto L* où *L* est l'étiquette d'un énoncé dans la séquence;
- *if x relop y goto L* où *relop* est un opérateur de comparaison comme ≤, >, ≠, …
- *param x* et *call p, n* pour la préparation des appels de procédures où *n* indique le nombre de valeurs passées en argument;

II-1. Types d'énoncés

Les types d'énoncés du code à trois adresses :

- *return [y]* pour le retour des appels de procédures;
- *x := y[i]* et *x[i] := y* met dans *x* la valeur située à i cases mémoire de *y* (adresse de *y*);
- *x := &y, x := *y* et **x := y* pour la manipulation des pointeurs;
- *L : pseudo-énoncé* déclare l'étiquette *L*, il marque la position de l'énoncé suivant.

Plan

I. Code intermédiaire

II. Code à trois adresses

1. Types d'énoncés
2. Instructions

III. Structures de données de sauvegarde de la représentation

1. Quadruples
2. Triplet
3. Exercices

IV. Traduction des expressions

1. Symboles des règles sémantiques
2. Traduction des expressions
3. Exercices

II-2. Instructions

Exercice:

Représenter en code à trois adresses les instructions suivantes:

```
do
    i = i + 1;
while( a[i] < v);
```

II-2. Instructions

Solution Exercice: **Étiquettes symboliques**

La représenter en code à trois adresses des instructions *do i = i + 1; while(a[i] < v);* :

L : $t_1 = i + 1$
$i = t_1$
$t_2 = i * 8$ (8 = type.width)
$t_3 = a[t_2]$
if $t_3 < v$ goto L

II-2. Instructions

Solution Exercice: **Numéros de position**

La représenter en code à trois adresses des instructions *do i = i + 1; while(a[i] < v);* :

100 : $t_1 = i + 1$
101 : $i = t_1$
102 : $t_2 = i * 8$ *(8 = type.width)*
103 : $t_3 = a[t_2]$
104 : *if* $t_3 < v$ *goto 100*

Plan

I. Code intermédiaire

II. Code à trois adresses

1. Types d'énoncés
2. Instructions

III. Structures de données de sauvegarde de la représentation

1. Quadruples
2. Triplet
3. Exercices

IV. Traduction des expressions

1. Symboles des règles sémantiques
2. Traduction des expressions
3. Exercices

III- Structures de données de sauvegarde de la représentation

Pour sauvegarder la représentation, on peut utiliser :

- **Quadruple**: quatre champs: op, arg1, arg2 et résultat;
- **Triplet**: le champ des résultats n'est pas utilisé. On utilise directement les références aux instructions;

Plan

I. Code intermédiaire

II. Code à trois adresses

1. Types d'énoncés
2. Instructions

III. Structures de données de sauvegarde de la représentation

1. Quadruples
2. Triplet
3. Exercices

IV. Traduction des expressions

1. Symboles des règles sémantiques
2. Traduction des expressions
3. Exercices

III-1. Quadruple

Exemple:

a := b * −c + b * −c

```
t1 : = -c
t2 := b * t1
t3 : = - c
t4 := b * t3
t5 := t2 + t4
a  := t5
```

	Op	Arg1	Arg2	result
(0)	uminus	c		t1
(1)	*	b	t1	t2
(2)	uminus	c		t3
(3)	*	b	t3	t4
(4)	+	t2	t4	t5
(5)	:=	t5		a

Plan

I. Code intermédiaire

II. Code à trois adresses

1. Types d'énoncés
2. Instructions

III. Structures de données de sauvegarde de la représentation

1. Quadruples
2. Triplet
3. Exercices

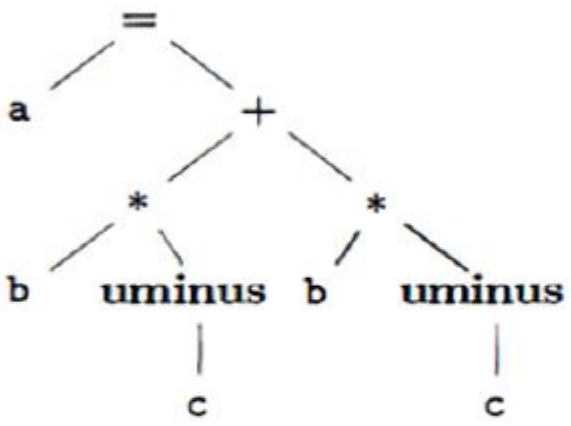
IV. Traduction des expressions

1. Symboles des règles sémantiques
2. Traduction des expressions
3. Exercices

III-2. Triplet

Exemple:

a := b * −c + b * −c



Syntax tree

	Op	Arg1	Arg2
(0)	uminus	c	
(1)	*	b	(0)
(2)	uminus	c	
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	:=	a	(4)

Plan

I. Code intermédiaire

II. Code à trois adresses

1. Types d'énoncés
2. Instructions

III. Structures de données de sauvegarde de la représentation

1. Quadruples
2. Triplet
3. Exercices

IV. Traduction des expressions

1. Symboles des règles sémantiques
2. Traduction des expressions
3. Exercices

III-3. Exercices

Traduire l'expression arithmétique

$$a + -(b + c)$$

en:

a) Arbre syntaxique.

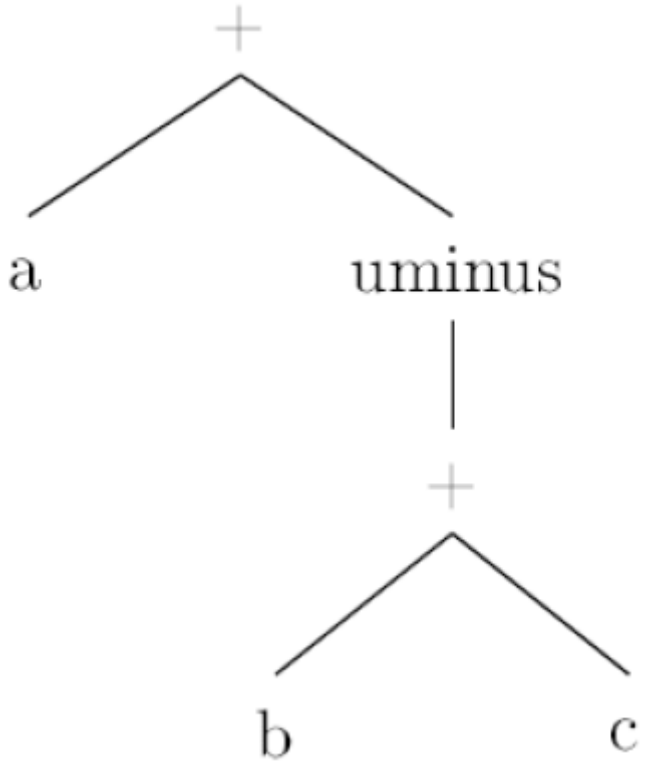
b) Quadruples.

c) Triplets.

III-3. Exercices

Traduire l'expression arithmétique a + −(b + c) en:

a) Arbre syntaxique.



III-3. Exercices

Traduire l'expression arithmétique a + −(b + c) en:

b) Quadruples.

	Op	Arg 1	Arg 2	Résultat
0	+	b	c	t_1
1	uminus	t_1		t_2
2	+	a	t_2	t_3

III-3. Exercices

Traduire l'expression arithmétique a + −(b + c) en:

c) Triplets

	Op	Arg 1	Arg 2
0	+	b	c
1	uminus	(0)	
2	+	a	(1)

Plan

I. Code intermédiaire

II. Code à trois adresses

1. Types d'énoncés
2. Instructions

III. Structures de données de sauvegarde de la représentation

1. Quadruples
2. Triplet
3. Exercices

IV. Traduction des expressions

1. Symboles des règles sémantiques
2. Traduction des expressions
3. Exercices

IV- Traduction des expressions

- Le but est de traduire le code (LH) en un code intermédiaire (code à trois adresses) qui ne dépend pas des détails matériels de la machine cible;
- Cette traduction est faite en se basant sur une grammaire de construction de code à trois adresses;
- Dans cette grammaire, un attribut (chaque symbole utilisés sauf les terminaux) possède : un **Code**, une **Place**-**Adresse** et une **Valeur**.
- Sur ces attributs, on applique des **Règles sémantiques** pour générer du code à trois adresses.

Plan

I. Code intermédiaire

II. Code à trois adresses

1. Types d'énoncés
2. Instructions

III. Structures de données de sauvegarde de la représentation

1. Quadruples
2. Triplet
3. Exercices

IV. Traduction des expressions

1. Symboles des règles sémantiques
2. Traduction des expressions
3. Exercices

IV-1. Symboles des règles sémantiques

- → ⇔ Produit ;
- || ⇔ Ensuite (concaténation);
- gen ⇔ Générer code à trois adresses;
- lexeme ⇔ Tel quel;
- ‘ ‘ ⇔ litéral;
- newTemp() ⇔ Créer une adresse temporaire;
- ( ) ⇔ Ne pas créer une adresse temporaire, copier l’adresse et le code directement;

Plan

I. Code intermédiaire

II. Code à trois adresses

1. Types d'énoncés
2. Instructions

III. Structures de données de sauvegarde de la représentation

1. Quadruples
2. Triplet
3. Exercices

IV. Traduction des expressions

1. Symboles des règles sémantiques
2. Traduction des expressions
3. Exercices

IV-2. Traduction des expressions

Production	Semantic Rules
S → id := E;	S.code = E.code || gen (top.get(id.lexeme) '=' E.address
E → E1 + E2	E.addr = new Temp() E.code = E1.code || E2.code || gen (E.addr '=' E1.addr '+' E2.addr)
E → - E1	E.addr = new Temp() E.code = E1.code || gen (E.addr '=' 'uminus' E1.addr)
E → E1 * E2	E.addr = new Temp() E.code = E1.code || E2.code || gen (E.addr '=' E1.addr '*' E2.addr)
E → (E1)	E.addr = E1. addr E.code = E1.code
E → id	E.addr = top.get(id.lexeme) E.code = ' '

Plan

I. Code intermédiaire

II. Code à trois adresses

1. Types d'énoncés
2. Instructions

III. Structures de données de sauvegarde de la représentation

1. Quadruples
2. Triplet
3. Exercices

IV. Traduction des expressions

1. Symboles des règles sémantiques
2. Traduction des expressions
3. Exercices

IV-3. Exercices

Donner l'arbre syntaxique et la table de symboles ainsi que le code généré (attributs et variables temporaires) pour l'instruction suivante :

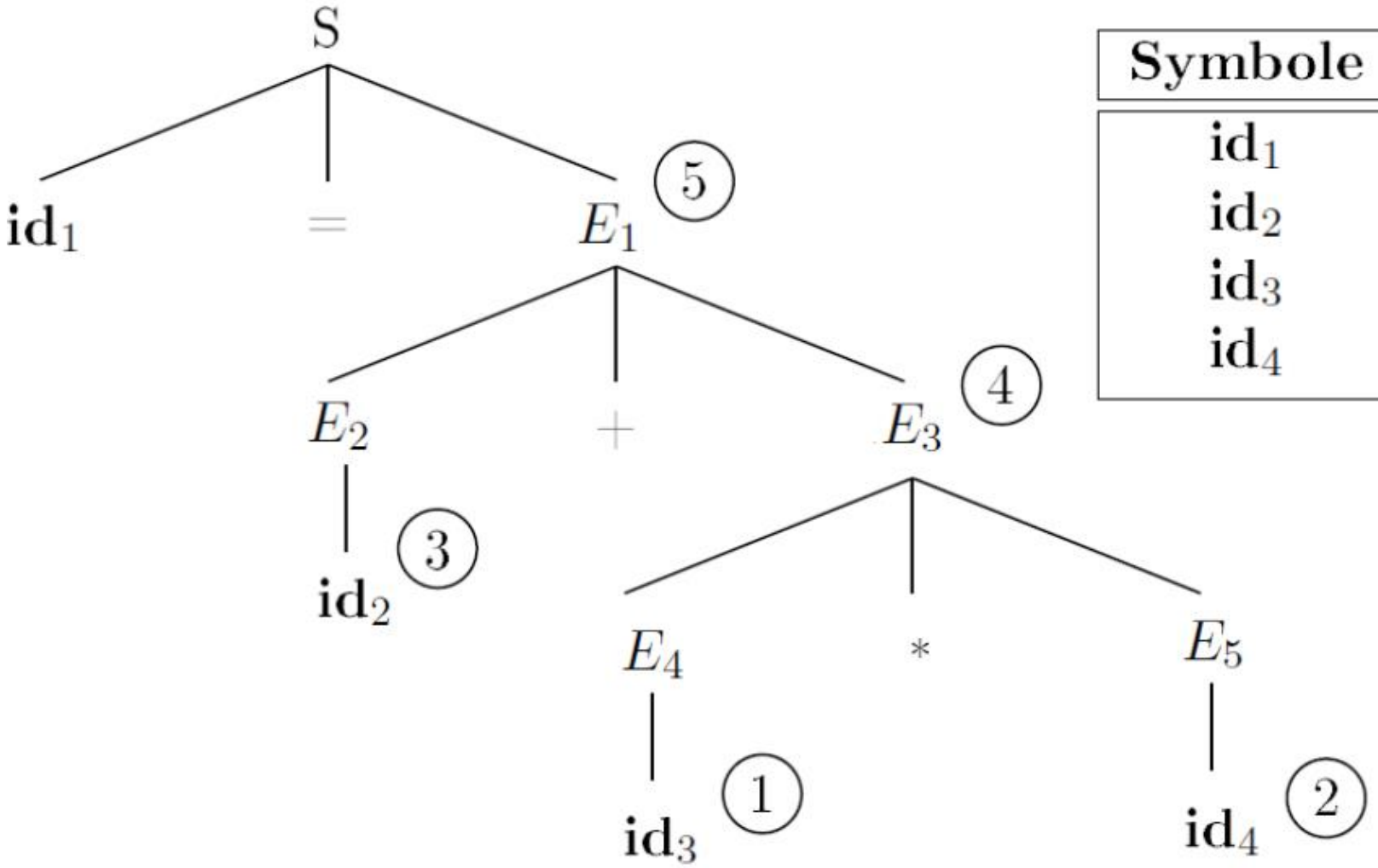
$$a = b + c * d$$

<u>La grammaire :</u>

$S \rightarrow id := E;$
$E \rightarrow E1 + E2$
$E \rightarrow -E1$
$E \rightarrow E1 * E2$
$E \rightarrow (E)$
$E \rightarrow id$

IV-3. Exercices

Donner l'arbre syntaxique et la table de symboles ainsi que le code généré (attributs et variables temporaires) pour l'instruction suivante : $a = b + c * d$



Symbole	Nom
$\mathbf{id}_1$	a
$\mathbf{id}_2$	b
$\mathbf{id}_3$	c
$\mathbf{id}_4$	d

IV-3. Exercices

Donner l'arbre syntaxique et la table de symboles ainsi que le code généré (attributs et variables temporaires) pour l'instruction suivante : $a = b + c * d$

Étape	Attributs et temporaires	Code généré
① $E_4 \rightarrow id_3$	E_4.addr = top.get(id_3. lexeme) E_4. code = ' ' **E_4.addr = id_3.place**	
② $E_5 \rightarrow id_4$	E_5.addr = top.get(id_4. lexeme) E_5. code = ' ' **E_5.addr = id_4.place**	
③ $E_2 \rightarrow id_2$	E_2.addr = top.get(id_2. lexeme) E_2. code = ' ' **E_2.addr = id_2.place**	
④ $E_3 \rightarrow E_4 * E_5$	E_3.addr = new Temp() = t_2 E_3.code = E_4.code || E_5.code || gen(E_3.addr '=' E_4.addr '*' E_5.addr)	$t_2 = c * d$
⑤ $E_1 \rightarrow E_2 * E_3$	E_1.addr = new Temp() = t_1 E_1.code = E_2.code || E_3.code || gen(E_1.addr '=' E_2.addr '+' E_3.addr)	$t_1 = b + t_2$
$S \rightarrow id_1 := E_1$	S.code = E_1.code || gen(top.get(id . lexeme) '=' E_1.addr)	$a = t_1$

Traduction "if"

PRODUCTION	RÈGLES SÉMANTIQUES
$S \rightarrow$ **if** E **then** S_1 **else** S_2	$S.L_{else}$:= **new** Label $S.L_{sortie}$:= **new** Label $S.code := E.code \parallel$ $gen(\text{'}\textbf{if}\text{'}\ E.place\ \text{'}=\text{'}\ \text{'}0\text{'}\ \text{'}\textbf{goto}\text{'}\ S.L_{else}) \parallel$ $S_1.code \parallel gen(\text{'}\textbf{goto}\text{'} S.L_{sortie}) \parallel$ $gen(S.L_{else}\ \text{':'}) \parallel S_2.code \parallel gen(S.L_{sortie}\ \text{':'})$

Allure du code généré:

```
E.code
if E.place = 0 goto Lelse
S1.code
goto Lsortie

Lelse:
S2.code

Lsortie:
```

Déclaration

1. Donner une grammaire qui permet les déclarations des variables et des tableaux (considérer juste les entiers et les réels).

2. Donner les règles sémantiques de génération de code sachant que chaque variable possède deux attributs : **type** et **width**
   pour un entier : le type est integer et le width est 4
   pour un réel : le type est real et le width est 8

Déclaration

Donner une grammaire qui permet les déclarations des variables et des tableaux (considérer juste les entiers et les réels).

D → D ; D
D → T id
D → ε
T → B C
B → integer | real
C → ε
C → [num] C

Déclaration

- Le calcul des adresses des variables est effectué par des règles sémantiques reliées au code à trois adresses.

- Il est basé sur : **Type**, **Width** et **Offset**.

P → D	Déclaration
enter	Crée une nouvelle entrée dans la table de symboles

Déclaration

Production	Règles sémantiques
P → D	{offset = 0}
D → D ; D	
D → T id	{enter (id.name, T.type, offset); offset = offset + T.width;}
D → ε	
T → B C	
B → integer	T.type = integer; T.width = 4;
B → real	T.type = real; T.width = 8;
C → ε	
C → [num] C	T.type = array(num.val, B.type); T.width = num * B.type;

D → D ; D
D → T id
D → ε
T → B C
B → integer | real
C → ε
C → [num] C