

1 Introduction

L'architecture d'un compilateur est classiquement composé de :

- Analyse lexicale
- Analyse syntaxique
- Analyse sémantique
- Génération de code intermédiaire
- Optimisation de code
- Génération de code cible.

1.1 Un langage, sa grammaire

```
Program      ::= Block
Block        ::= begin Declaration_list ; Instruction_list end
Declaration_list ::= Declaration_list ; Declaration | Declaration
Declaration  ::= Idf := Expression : Type
Type         ::= integer
Instruction_list ::= Instruction_List ; Instruction | Instruction
Instruction   ::= Idf := Expression | skip |
                if Expression then Instruction else Instruction endif |
                if Expression then Instruction endif |
                while Expression do Instruction done
Expression   ::= Disjonction
Disjonction  ::= Conjunction | Disjonction or Conjunction
Conjunction  ::= Comparison | Conjunction and Comparison
Comparison   ::= Relation | Relation = Relation
Relation     ::= Sum | Sum < Sum
Sum          ::= Term | Sum + Term | Sum - Term
Term         ::= Factor | Factor * Term
Factor       ::= Not Factor | Denotation | Idf | '('Expression)'
```

1.2 Grammaire abstraite

```
S ::= x := a | skip | S ; S | if b then S else S | while b do S
a ::= n | a + a | a - a | a * a | x
b ::= true | false | a = a | a ≤ a | ¬b | b ∧ b
```

2 Exercice I :Analyse lexicale

Un analyseur lexical a pour entrée une chaîne de caractères et en sortie un couple classe lexicale, élément de la classe.

Question Repérez les classes lexicales et donnez-en une spécification sous forme d'automate ou d'expression régulière.

3 Exercice 2 : Analyse syntaxique

Un analyseur syntaxique a pour entrée un flot de classes lexicales et en sortie un arbre abstrait, décrit par la grammaire abstraite.

Question Construire les arbres syntaxiques et les arbres abstraits pour les expressions suivantes :

a+b-c
a-b+c
a+b*c

4 Exercice 3 : Vérification de types

La vérification de types se fait sur l'arbre abstrait.

Question Donnez informellement les règles pour vérifier les types.

5 Exercice 4 : Génération de code

En entrée de la génération, on a un arbre abstrait, en sortie une séquence d'instructions. En considérant la syntaxe abstraite du paragraphe 1.2, on considère 3 fonctions de génération :

- Une fonction qui prend une instruction S et qui a pour résultat une séquence d'instructions machine,
- Une fonction qui prend une expression arithmétique a et qui a pour résultat une séquence d'instructions machine et le registre qui contient le résultat,
- Une fonction qui prend une expression booléenne b et qui a pour résultat une séquence d'instructions machine et le registre qui contient le résultat.

5.1 Machine M

Nous décrivons, dans ce qui suit, une machine à registres. Les opérations arithmétiques et logiques mettent à jour les codes condition. Les opérations arithmétiques et logiques, notées OPER, sont ADD, SUB, AND, OR.

Les registres généraux sont notés Ri. Le registre CP désigne le compteur programme, le registre FP la base de l'environnement local et le registre SP le sommet de pile.

Les instructions et adresses sont codées sur 4 octets, le pointeur de pile désigne le dernier emplacement occupé, on empile en décrémentant de 4 le registre SP et on dépile en incrémentant de 4 le registre SP. Les entiers sont codés sur 4 octets.

A chaque variable d'un programme, on associe une adresse de la forme $FP - \text{depl}$, où depl est calculé et mis dans la table des symboles.

Dans cet exercice, on ne s'intéresse qu'aux opérations élémentaires, dont la syntaxe est donnée ci-dessous. val désigne une valeur constante. Nous verrons plus tard, les opérations de branchement et d'appel de procédure.

OPER Ri,Rj,Rk Ri est le registre destination
 OPER Ri, Rk, val Ri est le registre destination
 LD Ri, [adr]
 ST Ri, [adr]

Les adresses sont décrites de la manière suivante :

$\text{adr} := \text{Ri} + \text{Rj} \mid \text{Ri} + \text{val} \mid \text{Ri} \mid \text{val}$

Question Donner la séquence de code correspondant aux expressions et séquences d'instructions ci-dessous, en supposant la table des symboles suivante :

identificateur	déplacement
x	4
y	8
z	12
u	16
v	20
w	24

- $x - y + z$
- $x := 3$
- $x := 3; y := x + 2;$
- $x := 3; y := 2; z := x + y$
- $x := 3; y := 2; z := x + y; u := x + y + z; v := z + y; w := y + v;$

5.2 Exercice 5 : Optimisations locales

Minimisez le nombre de registres utilisés dans les séquences de codes précédentes. Pour cela, on peut s'intéresser :

- au nombre minimal de registres pour évaluer une expression,
- à l'analyse de variables actives pour minimiser les LD et ST inutile,
- à une borne du nombre total de registres et à la politique de sauvegarde.

6 Exercice 6 : Optimisations globales indépendantes de la machine

Soit le programme écrit en forme intermédiaire suivant :

```
(1) a := 1
(2) c := a+b
(3) if c>0 goto 7
(4) d := a+b
(5) c := c+2
(6) goto 9
(7) f := a+b
(8) c := c-1
(9) g := d+c
```

Questions :

1. Construction du graphe de flot de contrôle,
2. Suppression des calculs redondants,
3. Suppression des affectations inutiles en supposant :
 - qu’aucune variable n’est utilisée à l’extérieur de ce programme,
 - que g est utilisée à l’extérieur,
4. Propagation des constantes.