كلـيـــــــــــة العلوم
السملالية - مراكش
FACULTE DES SCIENCES
SEMLALIA-MARRAKECH

MARRAKECH
جامعة القاضي عياض
UNIVERSITÉ CADI AYYAD

Département Informatique

Licence SMI – S6

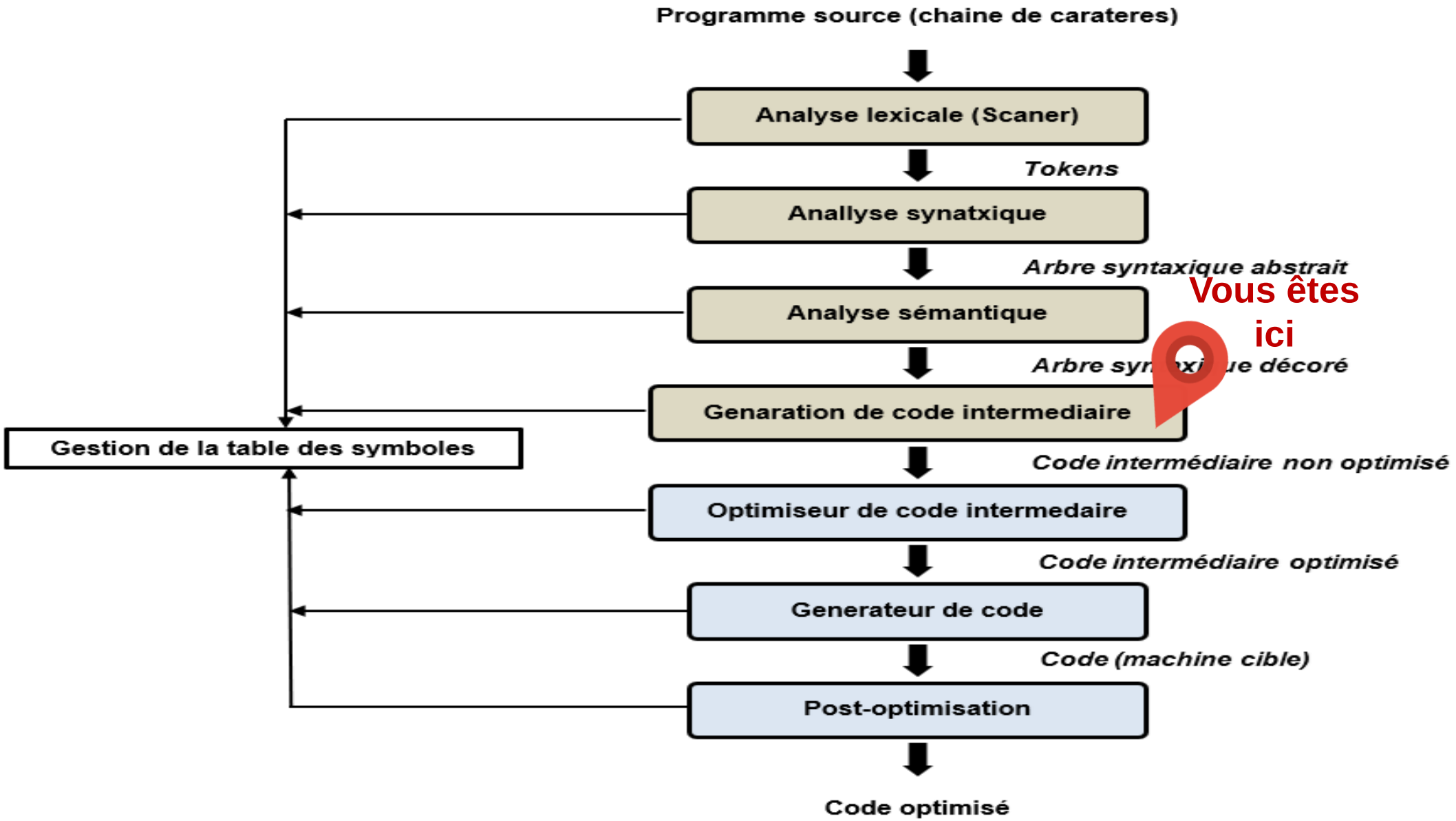
Compilation

Pr. Aimad QAZDAR

aimad.qazdar@uca.ac.ma

7 mai 2021

Rappel : Phases d'un compilateur



Chapitre 5 :

Génération du code intermédiaire

Niveaux de langages

- Langages spécifiques (ex. : Matlab, Halide, Shell scripts...)
  - Langages spécialisés disposant de constructions (données et/ou contrôle) spécifiques à un domaine en particulier
- Langage de haut niveau (ex. : Java, C#, C/C++...)
  - Langages généralistes adaptés aux algorithmes et données évolués indépendamment de l'architecture cible
- Langages intermédiaires (ex. : GIMPLE, Bytecode Java...)
  - Langages internes aux compilateurs, communs à tous les langages supportés et adaptés à leur optimisation
- Langages d'assemblage (ex. : assembleur x86 ou ARM...)
  - Langage proche du langage machine mais où instructions et adresses sont remplacées par des noms lisibles
- Langages machine (ex. : x86, ARM...)
  - Suite binaire directement interprétable par l'architecture

Machines à registres et Machines à pile

- Les langages évolués permettent l'écriture d'expressions en utilisant la notation algébrique à laquelle nous sommes habitués,
- Exemple : A = B + C, cette formule signifiant « ajoutez le contenu de B à celui de C et rangez le résultat dans A »
- A, B et C correspondent à des emplacements dans la mémoire de l'ordinateur.
- Une telle expression est trop compliquée pour le processeur, il faut la décomposer en des instructions plus simples.
- La nature de ces instructions plus simples dépend du type de machine dont on dispose.
- Relativement à la manière dont les opérations sont exprimées, il y a deux grandes familles de machines :
  - Les *machines à registres*
  - Les *machines à pile*

Les machines à registres

- Possèdent un certain nombre de registres, notés ici R1, R2, etc., qui sont les seuls composants susceptibles d'intervenir dans une opération (autre qu'un transfert de mémoire à registre ou réciproquement) à titre d'opérandes ou de résultats.
- Inversement, n'importe quel registre peut intervenir dans une opération arithmétique ou autre ;
- Par conséquent, les instructions qui expriment ces opérations doivent spécifier leurs opérandes.

Les machines à registres (2)

- Si on vise une telle machine, l'affectation *A*= B + C devra être traduite en quelque chose comme :

```
MOVE @B,R1 // déplace la valeur de B dans R1
MOVE @C,R2 // déplace la valeur de C dans R2
ADD R1,R2 // ajoute R1 à R2
MOVE R1,@A // déplace la valeur de R1 dans A
```

Les machines à pile

- Disposent d'une pile (qui peut être la pile des variables locales) au sommet de laquelle se font toutes les opérations.
- Pour une telle machine, le code *A*= B + C se traduira donc ainsi :
  - les opérandes d'un opérateur binaire sont toujours les deux valeurs au sommet de la pile ; l'exécution d'une opération binaire $\otimes$ consiste toujours à dépiler deux valeurs B et C et à empiler le résultat B $\otimes$ C de l'opération ;
  - l'opérande d'un opérateur unaire est la valeur au sommet de la pile ; l'exécution d'une opération unaire $\triangleright$consiste toujours à dépiler une valeur A et à empiler le résultat $\triangleright$A de l'opération.

```
PUSH @B // met la val. de B au sommet de la pile
PUSH @C // met la val. de C au sommet de la pile
ADD // remplace les 2 val. au sommet de la pile par leur somme
POP @A // enlève la val. au sommet de la pile et la range dans A
```

Interprétation et compilation

- Les langages informatiques peuvent grossièrement se classer en deux catégories :
  - les **langages interprétés**
  - les **langages compilés**.

Langage interprété

- Un langage informatique est par définition différent du langage machine.
- Il faut donc le traduire pour le rendre intelligible du point de vue du processeur.
- Un programme écrit dans un langage interprété a besoin d'un programme auxiliaire (l'interpréteur) pour traduire au fur et à mesure les instructions du programme.

Langage compilé

- Un programme écrit dans un langage dit « **compilé** » va être traduit une fois pour toutes par un programme annexe, appelé **compilateur**, afin de générer un nouveau fichier qui sera autonome,
- C'est-à-dire qui n'aura plus besoin d'un programme autre que lui pour s'exécuter; on dit d'ailleurs que ce fichier est **exécutable**.
- Un programme écrit dans un langage compilé a comme avantage de ne plus avoir besoin, une fois compilé, de programme annexe pour s'exécuter.

Langage compilé (2)

- De plus, la traduction étant faite une fois pour toute, il est plus rapide à l'exécution.
- Toutefois il est moins souple qu'un programme écrit avec un langage interprété
  - car à chaque modification du fichier source (fichier intelligible par l'homme: celui qui va être compilé) il faudra recompiler le programme pour que les modifications prennent effet.
- D'autre part, un programme compilé a pour avantage de garantir la sécurité du code source.
  - En effet, un langage interprété, étant directement intelligible (lisible), permet à n'importe qui de connaître les secrets de fabrication d'un programme et donc de copier le code voire de le modifier.
  - Il y a donc risque de non-respect des droits d'auteur.
  - D'autre part, certaines applications sécurisées nécessitent la confidentialité du code pour éviter le piratage (transaction bancaire, paiement en ligne, communications sécurisées, ...).

Langages intermédiaires

- Certains langages appartiennent en quelque sorte aux deux catégories (LISP, Java, Python, ..)
- Car le programme écrit avec ces langages peut dans certaines conditions subir une phase de compilation intermédiaire vers un fichier écrit dans un langage qui
  - n'est pas intelligible (donc différent du fichier source)
  - et non exécutable (nécessité d'un interpréteur).
- Les applets Java, petits programmes insérés parfois dans les pages Web, sont des fichiers qui sont compilés mais que l'on ne peut exécuter qu'à partir d'un navigateur internet (ce sont des fichiers dont l'extension est .class).
- Passer par un langage intermédiaire est la méthode moderne pour écrire un compilateur.

Langages intermédiaires (2)

- Plusieurs types de code intermédiaire:
  - **L'arbre syntaxiques** (abstrait)
  - **La notation postfixée**, ou **préfixées** introduits dans le chapitre précédent
  - **Codes à 3 adresses** (quadruplets et triplets)
  - **Langage C** (**C--**)
- Le **Codes à 3 adresses** est une version simplifiée de l'assembleur où on ne se préoccupe pas ni de l'adressage des variables, ni des registres, ni du protocole des appels de fonction, ni d'autres subtilités de l'assembleur, etc.
- Il s'agit simplement de "découper" un code simple en morceaux de taille (à peu près) trois : deux arguments, un résultat.

Code à trois adresses

- C’est une séquence d’instruction de la forme générale : x := y op z
  - où op est un opérateur (arithmétique ou logique). Les trois adresses x, y et z. sont celles d’identificateurs du programme , des constantes ou des variables temporaires générées par le compilateurs.
- Proche d'un langage d'assemblage
- Utilise des noms symboliques (ex. add ou +) et des sauts
- Remarques:
  - Les instructions complexes sont fragmentées en instructions simples
  - Les variables temporaires contiennent les résultats intermédiaires.

Code à trois adresses (2)

- L’expression x + y * z pourra être traduite par la séquence suivante :
  - T1 := y*z
  - T2:= x+ T1

T1 et T2 sont des variables temporaires produites par le compilateur

- Le code à trois adresses est une structure intermédiaire bien adaptée à la complexité des problèmes dus aux expressions arithmétique et aux structures de contrôles imbriquées.

➔C’est une représentation linéarisée d’un arbre abstrait.

Code à trois adresses

Soit l'expression : A := (B+C) *D

Son code intermédiaire équivalent est :

T1 := B+ C
T2:= T1 * D

Exo: Chercher le code intermédiaire de l'expression:

A := - (A/B) * (C/D) + (A+B-C)

T1:= A/B
T2:= - T1
T3:= C/D
T4:= T2 * T3
T5 := A + B
T6 := T5 – C
T7:= T4 + T6
A := T7

Liste des instructions à trois adresses

- Instructions d'affectation :
  - x := y op z x := op y x:=y
  - Où op est un opérateur binaire ou unaire , arithmétique ou logique
- Instructions de saut
  - inconditionnel goto L
  - conditionnel if x oprel y goto L
    - « si oprel est vraie alors exécuter l'instruction à 3 adresses étiquetée par L »
- Les appels de procédures

Liste des instructions à trois adresses

- Les expressions sont évaluées de gauche à droite en tenant compte des priorités des opérateurs.
- Les valeurs vraie ou faux sont donnés aux expressions booléennes. 0 représente faux, 1 représente vrai

Code source : A or not B and C

Code intermédiaire:

T1:= not B

T2:= T1 and C

T 3 := A or T2

Code source : A > B

Code intermédiaire:

100 if A > B Goto 103

101 T1:= 0

102 Goto 104

103 T1:= 1

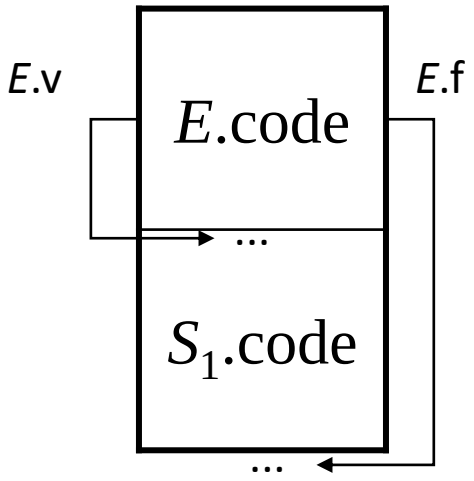
104 …

Liste des instructions à trois adresses

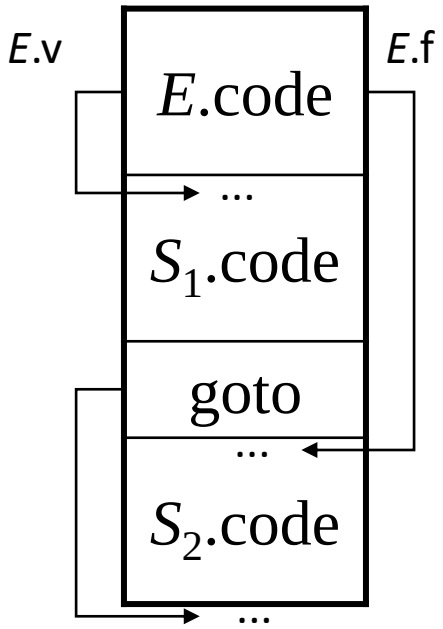
Soit une grammaire : S--> **if (** *E* **)** *S* | **if (** *E* **)** *S* **else** *S* | **while (** *E* **)** *S*

Donner le code intermédiaire des codes source:
if (E) then S1 else S2 et de While (E) S1

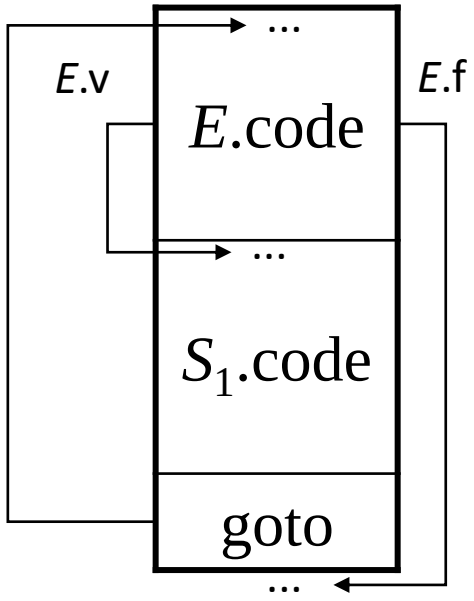
si-alors



si-alors-sinon



tant que



Liste des instructions à trois adresses

- Dans l'exemple , une boucle stocke les carrés des nombres entre 0 et 9:

```
for (i = 0; i < 10; ++i) {
     b[i] = i*i;
 }
```

```
      t1 := 0;              Initialisation de i
L1: if t1 >= 10 goto L2;    Saut conditionnel
      t2 := t1 * t1;        Calcul du carré de i
      t3 := t1 * 4;         Définition de l'adresse
      t4 := b + t3;         Adresse pour stocker i au carré
      *t4 := t2;            Stockage de l'information via le pointeur
      t1 := t1 + 1;         Incrémentation de i
      goto L1;              Répétition de la boucle
L2:
```

Code source

```
Algorithme Somme
   Var A, B, S: Entier
   Debut
      A=2
      B=3
      S=A+B
      Ecrire (S)
   Fin
```

Langage intermédiaire

```
Void main(){
   int A, B, S;
   A=2;
   B=3
   S=A+B
   printf("%d", S)
}
```