
# CSC4251_252 : Introduction — Prolégomènes

Pascal Hennequin, J. Paul Gibson, Denis Conan

Novembre 2024

# Sommaire

# 1 Compilation

- **De l'algorithme à l'assembleur**
  - **Entrée** : un algorithme implanté sous forme d'un programme dans un langage de programmation « évolué »

COMPILATEUR

  - **Sortie** : un programme équivalent exécutable par un processeur

- Exemple : C avec gcc
  - Attention ! Ce n'est pas l'approche de JAVA avec `javac` et `java`

# 2 Interprétation

- **Autre possibilité**
  - **Entrée** : un algorithme implanté sous forme d'un programme dans un langage de programmation « évolué »

INTERPRÉTEUR

  - **Sortie** : exécution directe

- Exemple : Bash, Python

# 3 Méli-mélo

- **On peut mélanger**
  - **Entrée** : un algorithme implanté sous forme d'un programme dans un langage de programmation « évolué »

    COMPILATEUR

  - **Forme intermédiaire** : un programme équivalent dans un autre langage « moins évolué »

    INTERPRÉTEUR

  - **Sortie** : exécution de la forme intermédiaire par un programme interpréteur ou une machine virtuelle

- Exemple : JAVA avec `javac` puis `java`

# 4 Traduction

- **De façon plus large**
  - **Entrée** : un fichier définissant des objets informatiques (exécution, donnée...) en utilisant un format / une syntaxe / un langage

TRADUCTEUR

  - **Sortie** : un fichier équivalent (ou pas) utilisant un autre langage

- Exemples : traitement de texte, conversion d'images, impression de documents, etc., *pretty print*, etc.

# 5 Phases de compilation

# 5.1 Analyses lexicale et syntaxique

## I Analyse Lexicale

- Reconnaître les mots du langage en entrée

$\pi$R2 = alpha + 1

| $\pi$R2 | = | alpha | + | 1 |
|---|---|---|---|---|
| IDENT | AFF | IDENT | OPBIN | ENTIER |

## II Analyse syntaxique

- Reconnaître les phrases en entrée : construire et valider la structure grammaticale

Grammaire
R1 : ENTIER $\leftarrow$ exp
R2 : IDENT $\leftarrow$ exp
R3 : IDENT AFF exp $\leftarrow$ phrase
R4 : exp OPBIN exp $\leftarrow$ exp

R3 ( R4 ( R2 , R1 ) )
$\Longrightarrow$ phrase complète

# 5.2 Analyse sémantique

## III Analyse sémantique

- Établir la signification du fichier en entrée
- Construire les éléments de contexte :
  - Liaison entre les définitions et les utilisations des variables ou fonctions
  - Vérification des contraintes de typage, gestion du transtypage
  - Analyse des chemins d'exécutions, i.e code mort, return absent
  - ...

$\pi$R2 = alpha + 1

int alpha

double alpha

String alpha

?

| | |
|---|---|
| Addition entière | Résultat entier<br>ou transtypage avant diffusion |
| Addition flottante | Transtypage (double) 1, etc. |
| Concaténation<br>de chaînes | Transtypage 1.toString(),<br>Méthode String.append(), etc. |

# 5.3 Code intermédiaire

## IV Code intermédiaire

- Traduction de l'entrée dans un langage intermédiaire qui se veut :
  - Indépendant du langage en entrée et plus simple
  - Indépendant du langage en sortie ou du traitement à suivre
- Interface entre partie avant et partie arrière du compilateur

$\pi$R2 = alpha + 1

int $\pi$R2 ;
void F(int alpha) { $\pi$R2 = alpha + 1 ; }

Code à 3 adresses :
t2 := alpha PLUS 1
$\pi$R2 := t2

*Byte code* JAVA :
1 iload_1[arg0]
2 iconst_1
3 iadd
4 putfield Test2.$\pi$R2 : int [2]

# 5.3.1 Exemple de *byte code* JAVA

$\pi$R2 = alpha + 1

String $\pi$R2 ;
void F(String alpha) { $\pi$R2 = alpha + 1 ; }

*Byte code* JAVA (cas String) :
01 new java.lang.StringBuilder [2]
04 dup 05 invokespecial java.lang.StringBuilder() [3]
08 aload_1 [arg0]
09 invokevirtual java.lang.StringBuilder.append(java.lang.String) : java.lang.StringBuilder [4]
12 iconst_1
13 invokevirtual java.lang.StringBuilder.append(int) : java.lang.StringBuilder [5]
16 invokevirtual java.lang.StringBuilder.toString() : java.lang.String [6]
19 putfield Test2.$\pi$R2 : java.lang.String [7]

# 5.4 Génération de code

## V Génération de code

- Optimisation du code intermédiaire
- Génération du code cible
- Optimisation du code cible
- Production de l'exécutable et édition des liens

MIPS [1] :
```
move $t2, $a1
li $v1, 1
add $t2, $t2, $v1
sw $t2, 0($a0)
```

ELF (gcc) :
```
4004f7 : mov %rsp,%rbp
4004fa : mov -0x4(%rbp),%eax
4004fd : add $0x1,%eax
400500 : mov %eax,0x200b2a(%rip)
```

## V bis Interprétation

- Exécution directe du code intermédiaire

1. MIPS est utilisé avec le simulateur MARS dans ce module

# 6 Objectif du cours I

- Écrire un compilateur pour un sous-ensemble du langage JAVA vers le langage assembleur MIPS
  - Comprendre par la pratique les enjeux et les techniques des différentes phases de la compilation
- Trois objectifs sous-jacents
  - Analyse lexicale et syntaxique : maîtriser les concepts théoriques et leurs mises en œuvre dans les outils dits « *compiler-compiler* »
  - Architecture des processeurs : instructions, registres, structures algorithmiques, appel de fonction, gestion mémoire...
  - Algorithmique et programmation :
    - Dans l'écriture du compilateur (algorithmique d'arbre, etc.)
    - Dans la compréhension de la chaîne de compilation et de la façon dont les paradigmes usuels de programmation sont implantés pour l'exécution

# 6 Objectif du cours II

- Ne fait pas partie du cours
  - Analyse sémantique dynamique et analyse de flux
  - Aspects théoriques de l'analyse sémantique
  - Optimisation du code généré et transformation d'arbre
  - Optimisation du code du compilateur
  - Compilation séparée et édition des liens

# 7 Bibliographie I

- Mots clés
  - Compilation, théorie des langages, expressions régulières, automates finis, grammaires algébriques (*context-free*), compilateur de compilateur (*compiler-compiler*), analyse syntaxique ascendante (LR)
- Bibles
  - *Gödel, Escher, Bach : an Eternal Golden Braid*, aussi appelé « GEB », Douglas Hofstadter, 1979, Basic Books.
  - *Compilers : Principles, Techniques, and Tools*, aussi appelé « *Dragon book* », A. Aho, M. Lam, R. Sethi, J Ullman, 2nd ed., 2006, Pearson Education.
  - *Modern Compiler Design in Java, 2nd edition*, A.W. Appel, Cambridge, 2002.
  - Compilateurs, D. Grune, H.E. Bal, C.J.H. Jacobs, K.G. Langendoen, Dunod 2002.

# 7 Bibliographie II

- Références
  - *Introduction to Automata Theory, Langages and Computation, 3rd Edition*, John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, 2013, Pearson Education.
  - Langages Formels, Calculabilité et Complexité, 2ème édition, Olivier Carton, 2020, Vuibert.
  - http://www.regular-expressions.info
  - *Learning Perl*, R.L.Schwarz, T. Christiansen, 1997, O'Reilly.
  - *Mastering Regular Expressions, 3rd Edition*, Jeffrey Friedl, 2006, O'Reilly.