



**UNIVERSITÉ  
DE GENÈVE**

**FACULTÉ DES SCIENCES**



# Compilateurs & Interprètes

Implantation des langages informatiques

Jacques Menu & Didier Buchs  
repris par Guido Bologna

`Guido.Bologna@unige.ch`

Cours 13x007 – Centre Universitaire d'Informatique

# 1 Prologue

But et trame du cours :

- présenter les techniques classiques d'implantation des langages informatiques, suite du cours sémantique des langages
- l'étudiant-e réalisera un projet de construction d'un petit compilateur (en 3 parties)

## 2 Introduction

Nous présentons dans ce chapitre le contexte dans lequel nous nous plaçons pour ce cours :

- langages et évaluation
- notion de grammaire formelle
- automates a piles
- analyse descendante
- analyse ascendante
- compiler et interpréter

## 2.1 Compiler

Etymologiquement selon : <http://www.cnrtl.fr/etymologie/compiler> :

COMPILER, verbe trans.

Étymol. et Hist. Ca 1265 spéc. en parlant d'un écrit « fait d'extraits d'ouvrages différents » (Brunet Latin, Trésor, 1 ds T.-L.) ; xiiies.  
« rassembler en un tout » (Li Epistle Saint Bernard a Mont Deu, ms. Verdun 72, fo115 rods Gdf.) ; 1758 péj. (Voltaire, Le Pauvre diable ds Littré). Empr. au lat. class. compilare « piller quelque chose, dépouiller quelqu'un, plagier (un auteur) » puis « composer, écrire » en lat. médiév. (viiies., Nierm.).

Informatiquement, c'est :

- ▷ **analyser** une description d'informations
- ▷ **synthétiser** une autre forme de celles-ci,  
*mieux adaptée à ce que l'on veut en faire,*  
typiquement une **exécution**
- ▷ tout en maintenant la **sémantique invariante**

## 2.1 Compiler, suite (2)

Un compilateur traduit automatiquement une **forme source** (*source form*) en une **forme objet** (*object form*, terme antérieur à l'orientation objets) :



Figure 1 – Compilation

Vue classique :

traduction d'un langage procédural  
en du code d'un processeur du marché

## 2.2 Interpréter

Etymologiquement :

expliquer, donner un sens, une **signification** (une **sémantique**),  
à quelque chose

Informatiquement :

**parcourir un graphe** (une structure de données chaînée)  
dont les nœuds sont appelés des **instructions**,  
jusqu'à ce que l'une d'elles indique la terminaison

## 2.2 Interpréter, suite (2)

Un **interprète** est un programme réalisant une interprétation :



Figure 2 – Interprétation

Vue classique :



langages implantés par une **machine virtuelle**, comme Java

Il y a tout de même pratiquement toujours *aussi*  
une compilation dans ce cas !

Une **machine informatique** est la réunion  
d'une mémoire de code et d'un interprète de ce code

# 3 Terminologie et exemples

Un langage informatique est un formalisme de **représentation d'informations**

Dans ce chapitre :

- ▷ nous introduisons les **notions fondamentales** dans le domaine des langages et de la compilation
- ▷ nous les illustrons par différents exemples
- ▷ en particulier, nous précisons les termes de **lexique**, **syntaxe** et **sémantique**

Nous utilisons le terme d'**implantation** de langages plutôt que l'anglicisme "implémentation"



## 3.1 Caractéristiques des langages informatiques

Un langage **comble un fossé sémantique** (*semantic gap*) entre :

- ▷ le niveau d'abstraction du problème à résoudre
- ▷ les opérations exécutables par une machine informatique

Plus le fossé est large,  
plus le langage est dit **de haut niveau** (*high level*)



Un langage informatique **unique** ne peut pas exister :  
il serait soumis à trop d'exigences contradictoires  
Sans cela, les cours de compilation  
perdraient d'ailleurs singulièrement de leur intérêt...

## 3.1 Caractéristiques des langages informatiques, (2)

Les souhaits pour un langage peuvent être par exemple :

- ▷ simplicité et ressemblance avec les **langues naturelles** en SQL
- ▷ écriture **postfixée** du code source pour PostScript, qui est utilisé pour programmer des imprimantes
- ▷ **notation et structure similaires** pour le code et les données en Lisp (tout est des listes)
- ▷ facilités de **traitement des chaînes de caractères** pour des langages de script comme Perl et Python
- ▷ **orientation objets** dans Smalltalk, Objective-C, C++ et Java

## 3.2 Lexique

Le lexique d'un langage définit les **mots** qui le composent :

- ▷ de manière **explicite**, en extension, comme :

```
(  procedure    if    >=    .    }    ;
```

- ▷ de manière **générique**, en compréhension, comme :

- ▷ identificateur
- ▷ chaîne de caractères
- ▷ constante numérique

Les langues naturelles ont par exemple des règles génériques pour le **nombre** (singulier-pluriel) et le **genre** (masculin-féminin)

En informatique, on parle de **symboles terminaux**, ou simplement "**terminaux**", pour les mots du langage

## 3.2 Lexique, suite (2)

La justification du terme “**terminal**” se verra sur les arbres de dérivation

Remarque importante :

Les terminaux génériques sont en fait des classes de terminaux, définis par des règles de bonne forme

Nous verrons que cela conduit souvent à des **grammaires à deux niveaux** :

1. le premier est la grammaire définissant les **classes de terminaux**
2. le second est le niveau syntaxique lui-même

## 3.3 Syntaxe

La syntaxe du langage régit la **forme** des phrases :

- ▷ les phrases acceptables au vu de la définition syntaxique **appartiennent** au langage
- ▷ les autres n'y appartiennent pas

La définition syntaxique d'un langage s'appuie sur :

- ▷ les **terminaux**, définis au niveau lexical
- ▷ des **règles de bonne forme** pour des **séquences** de terminaux appelées **notions non-terminales**

Là encore, la justification du terme “non-terminal” se verra sur les arbres de dérivation

## 3.3 Syntaxe, suite (2)

Toute description grammaticale textuelle d'un langage s'appuie sur une **syntaxe spécifique**, comme par exemple :

Bison :

Expression:

Expression PLUS Terme;

*non-terminal*

*"PLUS" est un terminal*

Dans cet exemple, on décrit un fragment de la syntaxe d'expressions algébriques usuelles au moyen d'une spécification écrite

## 3.4 Sémantique

La sémantique d'un langage est  
la **signification** véhiculée par les phrases de ce langage

L'intérêt des langages informatiques est ce rôle de “**véhicule**”,  
les aspects lexicaux et syntaxiques n'étant  
que des **maux nécessaires** pour y parvenir

Exemple en Fortran :

```
READ 100, FORMAT
```

```
... ..
```

```
100 FORMAT F10.3
```

*'FORMAT' désigne ici une variable réelle*

*'FORMAT' est ici un mot clé*

## 3.5 Notion de sur-langage

Un sur-langage d'un langage donné  
contient toutes les phrases de ce dernier

Cela permet :

- ▷ de réutiliser les analyseurs lexical et syntaxique du sur-langage
- ▷ dans un premier temps, d'accepter des phrases du sur-langage, supposé plus facile à analyser
- ▷ ensuite seulement,  
de refuser celles qui ne sont pas spécifiquement  
dans le (sous-)langage qui nous intéresse



## 3.6 Statique ou dynamique

Ces termes ont la sémantique suivante :

Caractéristique	En anglais	Signification
Statique	At compile-time	A la compilation du programme
Dynamique	At runtime	A l'exécution du programme

Ils s'appliquent en particulier au typage

Ainsi en Lisp, une variable peut recevoir dynamiquement une valeur de n'importe quel type :

```
(setq i '33)
(setq j 'toto)
(+ i j)
> Error: Argument "toto" is not of type NUMBER.
> While executing: +      erreur de sémantique dynamique
```

```
(setq j 19)
(+ i j)
```

## 3.6 Statique ou dynamique ?, suite (2)

Autres exemples :

- ▷ **évaluation** d'expressions  
“ $i+10*j$ ” peut être **évaluée dès la compilation**  
si on connaît à ce moment-là les valeurs de “ $i$ ” et “ $j$ ”
- ▷ **liens entre blocs d'activation** dans l'environnement d'exécution
  - ▷ un lien statique reflète  
**l'imbrication des déclarations de fonctions**,  
connue dès la compilation
  - ▷ un lien dynamique reflète. . .  
la dynamique des **appels de fonctions**

## 3.7 Empilement des machines

C'est dans un cas comme **PostScript** que l'on parle usuellement de langage “interprété” ou “pseudo-compilé” :

- ▷ il y a un empilement de plusieurs machines informatiques
- ▷ chacune exécute (interprète) un programme **implantant** celle qui se trouve au-dessus d'elle
- ▷ il y a une machine **réelle** au bas de cet empilement, les autres étant “**virtuelles**”

## 3.7 Empilement des machines, suite (2)

Synoptique d'implantation de PostScript – l'interprète est souvent en C :

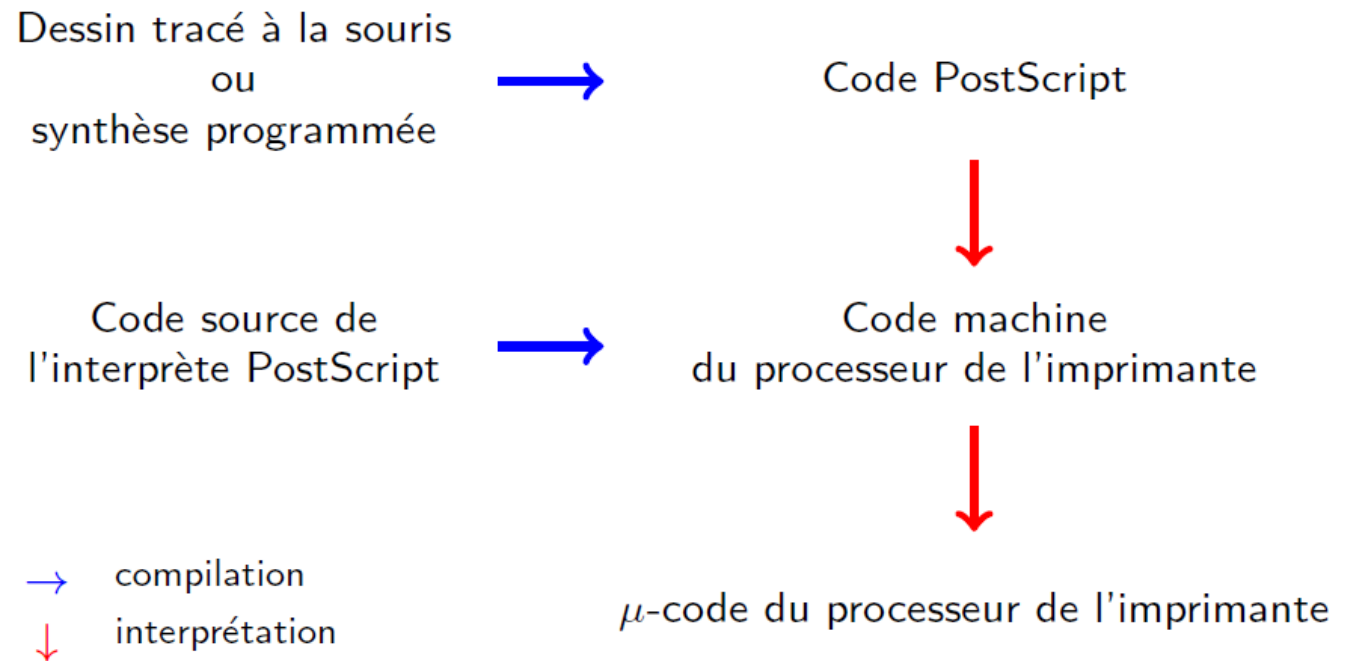


Figure 5 – Implantation de PostScript™

## 3.8 Analyse et synthèse, passes de compilation

Pour les tâches d'analyse et de synthèse,  
un compilateur **construit**  
une **description du code source** compilé

Dans le cas fréquent où la forme source est un fichier de caractères,  
on trouve trois tâches d'analyse typiques :

Analyse	Rôle
Lexicale	Lit les caractères du source, et détermine la <b>séquence des terminaux</b> le composant
Syntaxique	Vérifie que la <b>structure</b> de cette séquence est conforme à la syntaxe du langage
Sémantique	Contrôle la <b>signification</b> du code source

## 3.8 Analyse et synthèse, passes de compilation, (2)

Selon le langage, on mène toutes les tâches de compilation :

Tâches...	En...	Détails
De front	Une passe	On ne fait qu'un passage sur le texte source
Successivement	Plusieurs passes	On fait des passes successives sur des formes intermédiaires représentant le code source Certaines créent des structures de données utilisées par des passes ultérieures On peut ainsi faire une analyse sémantique plus fine et du meilleur code objet

## 3.8 Analyse et synthèse, passes de compilation, (3)

Exemple en Pascal :

```
program exemple;  
  var  
    i : integer;  
begin  
  write ('Veuillez fournir un entier: ' );  
  readln (i);  
  writeln ('Le carré de ', i, ' est ', i * i)  
end.
```

Identificateurs présents :

```
exemple  
i  
integer  
write  
readln  
writeln
```

## 3.9 Ordre d'évaluation et notation postfixée

L'ordre d'évaluation de " $f(3) + f(i)$ " est :

évaluer $f(3)$	ou	évaluer $f(i)$
évaluer $f(i)$		évaluer $f(3)$
faire l'addition		faire l'addition

mais l'addition se fait toujours **en dernier** !

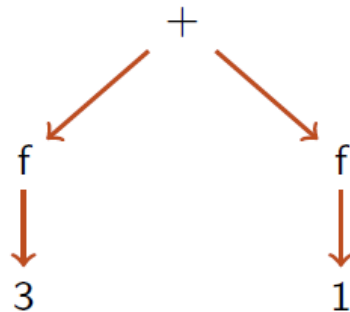


Figure 7 – Graphe sémantique



## 3.9 Ordre d'évaluation et notation postfixée, suite (2)

La notation postfixée est **incontournable** !

Elle n'est qu'une **écriture linéaire d'un graphe**,  
qui décrit l'**ordre d'évaluation** des opérandes et opérateurs

Que fait-on dans l'exemple ci-dessus  
de la valeur résultant de l'évaluation du premier opérande  
pendant l'évaluation du second ?

➡ On sauvegarde dans une **pile**  
les opérandes **en attente d'être consommés**  
par l'opération qui les utilise

On parle aussi parfois de **notation polonaise inverse**,  
en mémoire du mathématicien polonais **Lukacievitz**  
Cela se dit... "*odwrócona polska notacja*" en notation polonaise inverse

## 3.10 Fonctions strictes

Une fonction stricte :

- ▷ évalue **toujours tous** ses arguments d'appel
- ▷ ne peut donc être évaluée  
si l'un des arguments d'appel ne peut pas l'être

Un exemple typique de fonction **non stricte** est la conditionnelle **“Si”** :

- ▷ elle évalue toujours son premier argument d'appel,  
la **condition**
- ▷ elle n'évalue que l'un **ou** (exclusif) l'autre  
de ses deuxième et troisième arguments,  
selon la valeur de la condition

## 3.11 Compilation indépendante ou séparée

Les versions initiales du langage Fortran ne faisaient que de la compilation **indépendante** :

- ▷ on pouvait définir une fonction à deux paramètres formels. . .
- ▷ et l'appeler avec zéro, un, deux argument(s) ou plus depuis un autre fichier
- ▷ les compilateurs ne signalaient **pas d'erreur** dans ce cas : chaque fichier était compilé *sans aucune connaissance* des autres dans cette approche

## 3.11 Compilation indépendante ou séparée, suite (2)

Certains langages permettent de compiler des fichiers formant un tout, comme les fichiers “.h” et “.cp” en C++

On parle de compilation **séparée** dans ce cas :  
cette subdivision en plusieurs fichiers  
cela n'empêche pas les contrôles sémantiques  
effectués en tenant compte des fichiers importés

La **tendance moderne**  
est de contrôler le plus de choses possible **statiquement**, à la compilation,  
à part pour certains langages de scripts

L'exception qui confirme la règle est Ruby

Ada intègre même la gestion des bibliothèques compilées séparément  
dans la norme du langage, pour assurer la portabilité

## 3.11 Compilation indépendante ou séparée, suite (3)

En C, il peut être nécessaire d'appeler une fonction avec un nombre variable d'arguments, comme `printf ()`

Depuis C ANSI, ce point sensible est traité de manière "officielle" avec l'emploi de `...` comme dernier paramètre formel



Les contrôles faits à la compilation  
sont des **garde-fous protégeant le développeur !**

## 3.12 Auto-interprétation

Un **auto-interprète** ou **interprète méta-circulaire**  
est écrit dans le langage dont il peut interpréter les programmes  
En anglais : “*meta-circular interpreter*”

Cas classique : Lisp, avec ses fonctions “**eval**” et “**apply**”

Prolog est tellement puissant que l’auto-interprète tient en **4 clauses**,  
expliquant au passage comment une conclusion peut être démontrée

## 3.14 Autocompilation

Un **autocompileur** est écrit dans le langage qu'il peut compiler

Cela pose le problème du “**bootstrap**” ou amorçage de la pompe :

- ▷ en anglais : *tirer sur ses lacets* pour s'aider à monter un escalier
- ▷ on s'appuie souvent sur un langage de bootstrap disponible, par exemple un langage d'assemblage

## 3.14 Autocompilation, suite (2)

### Comment obtenir un auto-compilateur ?

- si on veut écrire le compilateur dans son propre langage  $S$ 
  - écrire d'abord un compilateur d'un sous-ensemble  $S'$  de  $S$  en  $E$ , et le compiler
  - écrire ensuite en  $S'$  le compilateur de  $S$
  - c'est ce qu'on appelle un *bootstrap* du compilateur



## 3.15 Générateurs de compilateurs

Ces outils réalisent la **synthèse automatique de compilateurs** d'après une spécification de tout ou partie du langage

Le terme anglais “*compiler compiler*” (compilateur de compilateurs) est en fait très mal choisi :

c'est une “**grammaire**” que l'on compile pour obtenir un compilateur, et non pas un autre compilateur !

Il serait plus approprié de parler de “*grammar compiler*” ou de “*compiler generator*”