Logiciel Libre
Cours 2 — Fondements: Programmation

Stefano Zacchiroli
zack@pps.univ-paris-diderot.fr

Laboratoire PPS, Université Paris Diderot

2013–2014

Outline

Contenu de ce cours

- Qu'est-ce que c'est un logiciel ?
- Langages de programmation
- Code source, compilation, et code exécutable

Outline

Logiciels

- Un logiciel consiste en des instructions qui peuvent être exécutées par une machine (a priori dans un sens très large).
- Les instructions sont écrites dans un certain langage de programmation, suivant des règles appelées la syntaxe du langage.
- Il y a énormément de langages qui peuvent être utilisés pour écrire un logiciel.
- L'exécution des instructions par une machine suit des règles strictes (la semantique du langage) il n'y a pas d'ambiguité dans le langage.
- Programme : le texte des instructions, contrairement à la représentation dans l'électronique d'une machine.

Exemple d'un langage : Logo (1)

- Développé pour enseigner la programmation, date : 1967.
- Instructions pour turtle graphics : on fait avancer une tortue (virtuelle) sur un canevas qui peut laisser une trace en se déplaçant.
- Quelques exemples d'instructions (ceci est un extrait de la définition exacte de la syntaxe du langage)

  `FORWARD` *n* où *n* est un entier [1]
  `TURNLEFT` *n* où *n* est un entier
  `TURNRIGHT` *n* où *n* est un entier
  `PENUP`
  `PENDOWN`

1. il faut aussi préciser comment noter des entiers

Exemple sémantique

- Sémantique du langage : on suppose qu'on dispose d'un canevas non borné. La tortue peut avoir le stylo posé sur le canevas (donc elle fait une trace quand elle bouge) ou le style levé (donc elle ne fait pas de trace quand elle bouge).
- Initialement : le stylo est posé, la tortue est orienté vers le nord.
- Sémantique de `FORWARD` n : avancer la tortue par n millimètres
- Sémantique de `TURNLEFT` n : tourner la tortue par n degrés dans le sens inverse des aiguilles d'une montre.
- Sémantique de `TURNRIGHT` n : ...
- Sémantique de `PENUP` : lever le stylo
- Sémantique de `PENDOWN` : poser le stylo

Exemple sémantique

- Sémantique du langage : on suppose qu'on dispose d'un canevas non borné. La tortue peut avoir le stylo posé sur le canevas (donc elle fait une trace quand elle bouge) ou le style levé (donc elle ne fait pas de trace quand elle bouge).
- Initialement : le stylo est posé, la tortue est orienté vers le nord.
- Sémantique de `FORWARD` n : avancer la tortue par n millimètres
- Sémantique de `TURNLEFT` n : tourner la tortue par n degrés dans le sens inverse des aiguilles d'une montre.
- Sémantique de `TURNRIGHT` n : ...
- Sémantique de `PENUP` : lever le stylo
- Sémantique de `PENDOWN` : poser le stylo

Est-ce une sémantique non ambiguë ?

Exemple sémantique

- Sémantique du langage : on suppose qu'on dispose d'un canevas non borné. La tortue peut avoir le stylo posé sur le canevas (donc elle fait une trace quand elle bouge) ou le style levé (donc elle ne fait pas de trace quand elle bouge).
- Initialement : le stylo est posé, la tortue est orienté vers le nord.
- Sémantique de `FORWARD` n : avancer la tortue par n millimètres
- Sémantique de `TURNLEFT` n : tourner la tortue par n degrés dans le sens inverse des aiguilles d'une montre.
- Sémantique de `TURNRIGHT` n : ...
- Sémantique de `PENUP` : lever le stylo
- Sémantique de `PENDOWN` : poser le stylo

Est-ce une sémantique non ambiguë ?
Non, p.ex. ça ne dit pas qu'est-ce qu'il se passe quand on fait PENUP et le stylo est déjà levé (rien, ou erreur ?)

Exemple d'un programme

```
FORWARD 50
TURNLEFT 90
FORWARD 50
TURNLEFT 90
FORWARD 50
TURNLEFT 90
FORWARD 50
TURNLEFT 90
```

Qu'est-ce que ce programme fait ?

Exemple d'un programme

```
FORWARD 50
TURNLEFT 90
FORWARD 50
TURNLEFT 90
FORWARD 50
TURNLEFT 90
FORWARD 50
TURNLEFT 90
```

Qu'est-ce que ce programme fait ?

Demo

(demo1.turtle)

L'état de l'exécution d'un programme

- L'effet de l'exécution d'une instruction dépend de l'*état* actuel de la machine qui exécute l'instruction.
- Exemple : **FORWARD** 20 fait un trait quand le stylo est posé, et ne fait pas de trait quand le stylo est levé.
- Un état de notre machine pour l'exécution de Logo consiste en :
  - Un design sur le canevas
  - La position de la tortue
  - L'orientation de la tortue
  - La position du stylo (posé ou levé)
- Sémantique d'une instruction : transformation d'un état dans un autre état.

Instructions composées

- Construction des instructions composées à partir des instructions plus simple
- Exemple d'une instruction composée : **REPEAT** n { l } où *n* est un entier, *l* est une séquence d'instructions qui sont séparées par des espaces.
- Exemple d'un programme :
  REPEAT 4 { **FORWARD** 100 **TURNLEFT** 90 }
- Les instructions dans la liste *l* peuvent être des instructions de base, ou encore des instructions composées.

Procédures

- Déclarer une procédure : donner un nom à un bout de code
- Syntaxe : LEARN *nom* { *l* }
  où *nom* est un nom de votre choix, *l* une liste d'instructions.
- Utiliser une procédure : écrire son nom

Exemple d'une procédure

```
LEARN SQUARE {
   REPEAT 4 {
      FORWARD 50
      TURNLEFT 90
   }
}

SQUARE
FORWARD 50
SQUARE
```

Qu'est-ce que ce programme fait ?

Exemple d'une procédure

```
LEARN SQUARE {
   REPEAT 4 {
      FORWARD 50
      TURNLEFT 90
   }
}

SQUARE
FORWARD 50
SQUARE
```

Qu'est-ce que ce programme fait ?

Demo

(demo3.turtle)

Abstraction

- Écrire du code générique (aussi général que possible)
- Exemple : procédure dont certaines valeurs restent abstraites
- Faire ces valeurs concrètes au moment de l'appel de la fonction
- On parle des *paramètres* d'une procédure

Exemple procédure avec paramètres

```
LEARN VARSQUARE $size {
   REPEAT 4 {
      FORWARD $size
      TURNLEFT 90
   }
}

VARSQUARE 40
VARSQUARE 60
VARSQUARE 80
```

Exemple procédure avec paramètres

```
LEARN VARSQUARE $size {
   REPEAT 4 {
      FORWARD $size
      TURNLEFT 90
   }
}

VARSQUARE 40
VARSQUARE 60
VARSQUARE 80
```

Demo

(demo4.turtle)

Étendre l'état de l'exécution

- Variable : un nom qui réfère à une valeur
- Cette valeur peut être modifiée : affectations
- Syntaxe en Logo : *$nom = valeur*
- La valeur actuelle de la variable fait partie de l'état d'exécution

Exemple avec affectation

```
LEARN SQUARE $size {
   REPEAT 4 {
      FORWARD $size
      TURNLEFT 90
   }
}
LEARN MANYSQUARES $number {
   $size = 30
   REPEAT number {
      SQUARE $size
      $size = $size + 10
   }
}
MANYSQUARES 8
```

Exemple avec affectation

```
LEARN SQUARE $size {
   REPEAT 4 {
      FORWARD $size
      TURNLEFT 90
   }
}
LEARN MANYSQUARES $number {
   $size = 30
   REPEAT number {
      SQUARE $size
      $size = $size + 10
   }
}
MANYSQUARES 8
```

Demo

(demo5.turtle)

Des langages de programmation réalistes

- type de données de base : nombre entiers, nombre flottant, mots (chaînes de caractères)
- type de données structurés : enregistrements, tableaux, arbres, ...
- structures de contrôle plus sophistiqués : procédures d'ordre supérieures, exceptions, objets, ...
- instructions pour l'interaction avec la machine : lecture et écriture de fichiers, clavier, accès réseau, ...
- Quelques exemples de langages : C, Java, Python, C++, Cobol, OCaml, Haskell, Pascal, Lisp, ...

Outline

Le choix d'un langage de programmation

- En principe, tous les vrais langages de programmation sont équivalents : Si on peut résoudre un problème dans un langage de programmation on peut aussi le résoudre dans n'importe autre langage.
- Un langage peut être orienté vers un certain domaine d'applications (par exemple math, traitement de langues, ...)
- Expressivité des structures de contrôle
- Conception d'un langage de programmation : problème difficile
- Le choix du langage de programmation a des conséquences pour la qualité du logiciel produit.

Le langage Pascal

- Développé par Nicolas Wirth, 1970, Suisse (Turing Award 1984)
- Nom : hommage à *Blaise Pascal*
- Approche de la programmation structurée (novateur à l'époque)
- Initialement conçu pour enseigner la programmation
- Langage tout à fait adapté pour des petits projets de programmation réels

Exemple d'un programme Pascal

```
program mine(output);
  var i: integer;
  procedure print(var j: integer);
    function next(k: integer): integer;
    begin
      next:= k + 1
    end;
  begin
    writeln('The total is: ', j);
    j:= next(j)
  end;

begin
  i:= 1;
  while i <= 10 do print(i)
end.
```

Le langage C

- Développé 1972 par Dennis Richie (Turing Award 1983), Bell Telephone Labs, États-Unis
- Très lié au système d'exploitation UNIX (voir plus tard)
- Un objectif principal : efficacité de l'exécution
- À l'époque considéré comme un langage d'un haut niveau d'abstraction
- Syntaxe très concise

Exemple d'un programme C (début)

```c
#include <stdio.h>

main() {
        int c;
        int i;
        for (i = 0; i < 128; i++)
                printf('%c', i);
        printf('\n');
        i = 0;

        while ((c = getchar()) != EOF & (c != 0200)) {
                i++;
                switch(c) {
                        case ' ': if (i > 100) i = 0;
        /* ... */
}
```

Le langage COBOL

- **CO**mmon **B**usiness-**O**riented **L**anguage
- 1959, consortium d'entreprises, et l'armée américaine
- Orienté à des applications en bureautique
- Syntaxe très loquace
- Toujours utilisé (voir plus tard : les logiciels ont toute une vie)

Exemple d'un programme COBOL

```
MULTIPLY B BY B GIVING B-SQUARED.
MULTIPLY 4 BY A GIVING FOUR-A.
MULTIPLY FOUR-A BY C GIVING FOUR-A-C.
SUBTRACT FOUR-A-C FROM B-SQUARED GIVING RESULT-1.
COMPUTE RESULT-2 = RESULT-1 ** .5.
SUBTRACT B FROM RESULT-2 GIVING NUMERATOR.
MULTIPLY 2 BY A GIVING DENOMINATOR.
DIVIDE NUMERATOR BY DENOMINATOR GIVING X.
```

pour calculer $x = \frac{-b-\sqrt{b^2-4ac}}{2a}$

Le langage LISP

- John Mc Carty, 1958, États-Unis
- Le deuxième plus ancien langage de haut niveau
- À la base : un calcul mathématique développé dans les années 1930 pour décrire le calcul : λ-calcul
- Le code de programme est considéré comme une valeur (!)
- Grande influence sur le développement des langages de programmation
- Des variantes de LISP sont toujours utilisées (p.ex. Emacs Lisp)
- Le seule élément syntaxique est une liste

Exemple d'un programme LISP

```
(defun unify (x y &optional (bindings no-bindings))
  (cond ((eq bindings fail) fail)
    ((eql x y) bindings)
    ((variable-p x) (unify-variable x y bindings))
    ((variable-p y) (unify-variable y x bindings))
    ((and (consp x) (consp y))
     (unify (rest x) (rest y)
     (unify (first x) (first y) bindings)))
    (t fail)))
```

- dont la blague : *Lots of Irritating Sequences of Parentheses*

Le langage Objective CAML

- Développé par Xavier Leroy et son une équipe à INRIA depuis les années 1990
- LISP est une des racines, puis la famille de langages ML (Robin Miller, années 1970, encore un Turing Award)
- Plusieurs paradigmes de programmation
- Types de données structurées
- Un système de typage fort : analyse du code qui permet de trouver beaucoup d'erreurs de programmation avant l'exécution du programme
- Efficacité comparable à C

Exemple d'un programme OCaml

```
exception Argument_negatif

let rec factorielle x =
    if x < 0
    then raise Argument_negatif
    else
        match x with
        | 0 -> 1
        | n -> n * (factorielle (n - 1))
```

pour définir la fonction qui calcule la factorielle.

$$factorielle(4) = 4 * 3 * 2 * 1 = 24$$

Les « guerres » des langages de programmation

- Le choix du bon langage de programmation est un sujet de débat inépuisable pour les programmeurs.
- C'est un choix crucial au début d'un projet de programmation qui a des conséquences importantes.
- Il y a plusieurs critères à prendre en considération :
  - Expressivité pour la tache à résoudre
  - Facilité pour les programmeurs (mais ça dépend des programmeurs !)
  - Efficacité d'exécution
  - Portabilité
  - ...

Les « guerres » des lang. de programmation (cont.)

> *If I were chained to a bench and Perl was the only thing that could open the lock, I'd probably cut my hand off.*
>
> — *Gerald Penn, université de Toronto*

Perl est un langage très utilisé mais considéré par beaucoup d'informaticiens comme mal conçu et difficile à maintenir.

Les « guerres » des lang. de programmation (cont.)

The use of COBOL cripples the mind ; its teaching should, therefore, be regarded as a criminal offense.

– Edsger Dijkstra, Turing Award 1972

Les « guerres » des lang. de programmation (cont.)

There are two types of programming languages ; the ones that people bitch about and the ones that no one uses.

— Bjarne Stroustrup, créateur du langage C++

Outline

Exécution d'un programme

- A priori un programme peut être exécuté à la main
- Il est aussi en principe concevable de construire un circuit électronique qui exécute directement les programmes du genre vu jusqu'à maintenant
- Mais ce n'est pas fait en pratique :
  - Les langage de programmation vu au-dessus sont conçu pour le programmeur (langages de haut niveau).
  - Les circuits pour les exécuter seraient beaucoup trop compliqués.
  - Il faudrait un circuit par langage de programmation, et construire un nouveau circuit pour toute petite modification d'un langage, ou pour tout nouveau langage.

Programmes machine

- Les programmes qui peuvent être exécutés par une machine (CPU) sont écrit dans un langage extrêmement simple qui est conçu pour l'efficacité et simplicité.
- Le langage précis dépend du type de processeur (Pentium, Alpha, AMD64, Powerbook, ...)
- Langage machine (simplement des séquences de bits) et langage assembleur (représentation [plus] lisible du langage machine)
- Instructions pour opérations arithmétiques, adresser une case mémoire, communiquer avec des périphériques (disques, réseau, etc.) mais sur un niveau d'abstraction très bas.

Exemple d'un programme en assembleur amd64

```
movl    -4(%rbp), %edi
movl    $0, %edx
movl    $1232, %esi
movl    $0, %eax
call    lseek
leaq    -16(%rbp), %rsi
movl    -4(%rbp), %edi
movl    $8, %edx
movl    $0, %eax
call    read
cmpl    $-1, %eax
jne     .L4
movl    $2, %edi
call    exit
```

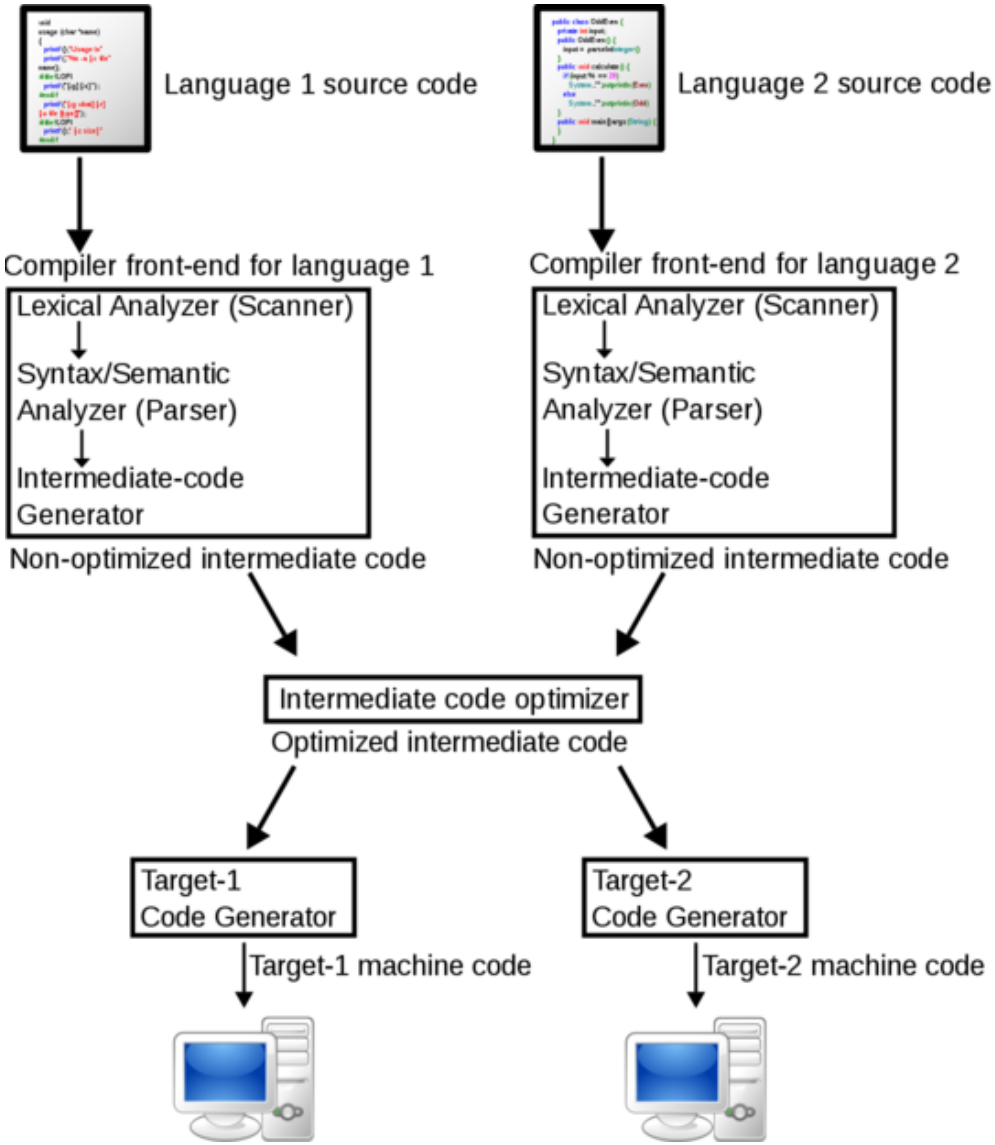
Le rôle d'un compilateur

- En peut en principe écrire directement des programmes en assembleur
- Compilateur : un programme qui lit un programme, puis
  - vérifie qu'il est conforme à la syntaxe du langage de programmation
  - le traduit en langage machine pour une certaine architecture, selon la sémantique du langage
  - Traduction en plusieurs étapes : langage de programmation → code intermédiaire → code assembleur / code machine
  - Construction de compilateurs : un sujet très complexe au coeur de l'informatique

Code source le programme dans le langage choisi par le programmeur

Exécutable le programme machine qui normalement est le résultat d'une compilation

Architecture d'un compilateur moderne



La poule et de l'oeuf

Dans quel langage écrire le compilateur ?

- On peut écrire le compilateur directement en code machine (difficile au mieux)
- On peut écrire le compilateur dans un autre langage de programmation pour lequel on a déjà un compilateur
- Bootstrapping : arriver à écrire le compilateur pour un langage *X* dans le langage *X* lui-même (!)
  Comment compiler le compilateur ?
  Le compilateur est organisé en plusieurs couches :
  - La première couche (pour un noyau du langage) écrit par exemple en Assembleur
  - Toute couche est écrite dans le sous-langage compris par la couche précédente.
  - Compilation couche par couche, jusqu'au langage complet.