Programming Paradigms
Introduction (Part 3)

Prof. Dr. Michael Pradel
Software Lab, University of Stuttgart
Summer 2020
Overview

■ Motivation
  □ What the course is about
  □ Why it is interesting
  □ How it can help you

■ Organization
  □ Exercises
  □ Grading

■ Introduction to Programming Languages
  □ History, paradigms, compilation, interpretation
First electronic computers: Programmed in **machine language**

- Sequence of bits
- Example: Calculate greatest common divisor

```
55 89 e5 53 83 ec 04 83 e4 f0 e8 31 00 00 00 89 c3 e8 2a 00
00 00 39 c3 74 10 8d b6 00 00 00 00 39 c3 7e 13 29 c3 39 c3
75 f6 89 1c 24 e8 6e 00 00 00 8b 5d fc c9 c3 29 d8 eb eb 90
```
History: From Bits ...

First electronic computers: Programmed in machine language

- Sequence of bits
- Example: Calculate greatest common divisor

```
55 89 e5 53 83 ec 04 83 e4 f0 e8 31 00 00 00 89 c3 e8 2a 00
00 00 39 c3 74 10 8d b6 00 00 00 00 39 c3 7e 13 29 c3 39 c3
75 f6 89 1c 24 e8 6e 00 00 00 8b 5d fc c9 c3 29 d8 eb eb 90
```

Machine time more valuable than developer time
... over Assembly ...

Human-readable abbreviations for machine language instructions

- Less error-prone, but still very machine-centered
- Each new machine: Different assembly language
- Developer thinks in terms of low-level operations
... over Assembly ...

Greatest common divisor in x86:

```
pushl %ebp
movl %esp, %ebp
pushl %ebx
subl $4, %esp
andl $-16, %esp
call getint
movl %eax, %ebx
call getint
cmpl %eax, %ebx
je C
A: cmpl %eax, %ebx

jle D
subl %eax, %ebx
B: cmpl %eax, %ebx
jne A
C: movl %ebx, (%esp)
call putint
movl -4(%ebp), %ebx
leave
ret
D: subl %ebx, %eax
jmp B
```
... to High-level Languages

- 1950s: First high-level languages
  - Fortran, Lisp, Algol
- Developer thinks in mathematical and logical abstractions
Greatest common divisor in Fortran:

```fortran
subroutine gcd_iter(value, u, v)
  integer, intent(out) :: value
  integer, intent(inout) :: u, v
  integer :: t

  do while( v /= 0 )
    t = u
    u = v
    v = mod(t, v)
  enddo
  value = abs(u)
end subroutine gcd_iter
```
Today: 1000s of Languages

- New languages gain traction regularly
- Some long-term survivors
  - Fortran, Cobol, C
Today: 1000s of Languages

- New languages gain traction regularly
- Some long-term survivors
  - Fortran, Cobol, C

Poll (in Ilias):
Your favorite programming language?
What Makes a PL Successful?

- **Expressive power**
  - But: All PLs are Turing-complete
- **Ease of learning** (e.g., Basic, Python)
- **Open source**
- **Standardization**: Ensure portability across platforms
- **Excellent compilers**
- **Economics**
  - E.g., C# by Microsoft, Objective-C by Apple
PL Spectrum

- **Broad classification**
  - Declarative ("what to compute"):
    
    E.g., Haskell, SQL, spreadsheets
  - Imperative ("how to compute it"):
    
    E.g., C, Java, Perl

- **Various PL paradigms**:  
  - Functional
  - Logic
  - Statically typed
  - Dynamically typed
  - Dataflow
  - Sequential
  - Distributed-memory parallel
  - Shared-memory parallel

- **Most languages combine multiple paradigms**
Example: Imperative PL

C implementation for GCD:

```c
int gcd(int a, int b) {
    while (a != b) {
        if (a > b) a = a - b;
        else b = b - a;
    }
    return a;
}
```
Example: Imperative PL

C implementation for GCD:

```c
int gcd(int a, int b) {
    while (a != b) {
        if (a > b) a = a - b;
        else b = b - a;
    }
    return a;
}
```

Statements that influence subsequent statements
Example: Imperative PL

C implementation for GCD:

```c
int gcd(int a, int b) {
    while (a != b) {
        if (a > b) a = a - b;
        else b = b - a;
    }
    return a;
}
```

Statements that influence subsequent statements

Assignments with side effect of changing memory
Example: Functional PL

OCaml implementation of GCD

```ocaml
let rec gcd a b =
  if a = b then a
  else if a > b then gcd b (a - b)
  else gcd a (b - a)
```
Example: Functional PL

OCaml implementation of GCD

```ocaml
let rec gcd a b =
  if a = b then a
  else if a > b then gcd b (a - b)
  else gcd a (b - a)
```

Recursive function with two arguments
Example: Functional PL

OCaml implementation of GCD

```ocaml
let rec gcd a b =
  if a = b then a
  else if a > b then gcd b (a - b)
  else gcd a (b - a)
```

Recursive function with two arguments

Focus on mathematical relationship between inputs and outputs
Example: Logic PL

Prolog implementation of GCD

\[
gcd(A, B, G) \, : \, \begin{cases} 
A = B, & G = A. \\
A > B, & C \text{ is } A-B, \, gcd(C, B, G). \\
B > A, & C \text{ is } B-A, \, gcd(C, A, G). 
\end{cases}
\]
Example: Logic PL

Prolog implementation of GCD

gcd(A, B, G) :- A = B, G = A.
gcd(A, B, G) :- A > B, C is A-B, gcd(C, B, G).
gcd(A, B, G) :- B > A, C is B-A, gcd(C, A, G).

Facts and rules
Example: Logic PL

Prolog implementation of GCD

\[
gcd(A, B, G) :- A = B, \ G = A.
gcd(A, B, G) :- A > B, \ C \ is \ A-B, \ gcd(C, B, G).
gcd(A, B, G) :- B > A, \ C \ is \ B-A, \ gcd(C, A, G).
\]
Compilation and Interpretation

Different ways of executing a program

■ Pure compilation
■ Pure interpretation
■ Mixing compilation and interpretation
  □ Virtual machines
  □ Just-in-time compilation
Compilation

Source program

Compiler

Input → Target program → Output
Interpretation

Source program

Inputs \rightarrow \text{Interpreter} \rightarrow \text{Output}
Just-in-compilation

Source program

Compiler

(Input)

[Bytecode interpreter] <-> [JIT compiler]

Virtual machine

(Output)
PL Design vs. Implementation

- Some PLs are easier to compile than others
- E.g., runtime code generation
  - Code to execute: Unknown at compile time
  - Hard to compile
  - Easy to interpret
Other Tools

- Linkers
- Preprocessors
- Source-to-source compilers
Linking

Source program

Compiler

Target program

Linker

Libraries

Complete target program
Preprocessors

Source program

Preprocessor

Modified source program

Compiler

Target program

E.g., macro expansion in C
Source-to-source compiler

Source program

Compiler 1

Alternative source program (e.g., in C)

Compiler 2

Target program