Paradigmi di Programmazione (Gruppo A)

Docente: Francesco Gavazzo
(francesco.gavazzo@unipi.it
francesco.gavazzo@gmail.com)

INTRODUCTION

PARADIGM = A collection of concepts and tools, together with a vocabulary to reason about them

PL PARADIGM = Defines what programs are
"
what we can compute

and
what is the underlying model of execution
"
how we compute

Examples

Imperative programming

- Programs = sequences of instructions (commands, statements, ...)
- Execution = state transformation

  $\langle \sigma, x := 1; skip \rangle \rightarrow \langle \sigma[x \mapsto 1], skip \rangle$

  Underlying computational model: Turing (von Neumann) machine

Functional programming

- Programs = (functional) expressions
- Execution = rewriting / reduction

  $(x+0) * 1 \rightarrow x * 1 \rightarrow x$

  Underlying computational model

  λ-Calculus / rewriting

Object-Oriented
- Programs = agents (internal state + behaviour)
- Execution = message passing / communication
  - Underlying computational model
  - Many: actors, chemical, concurrent-like, ...

Logic Programming
- Programs = logical theories
- Execution = (Inverse) deduction
  - Underlying computational model
  - First-order logic

Concurrent Programming

Bayesian "

Differentiable

⋮

➡ Modern languages are often **multiparadigm**

(Java, Python, Scala, OCaml, ...)

But not all (Haskell, ...)

How to study PL paradigms?

Not a linear process → learning is usually cyclic and dialectic

1. Study a new language (OCaml)
2. Understand its core features (Immutability, Higher-Order fun, Types, Pattern-matching, ...)

   Core part of a paradigm (but ≠ paradigm)

3. Understand its semantics
   - → Core calculus (λ-calculus, PCF, System F...)
   - → Operational semantics $(\lambda x.t)s \to_\beta t[s/x]$ — substitution model of evaluation
   - → Static semantics

   ⋮

4. Use all of that to implement languages
5. Repeat for different languages

Why is all of that useful?

- Two kinds of thinkers
  - (Hardcore) problem solvers: work hard to solve a problem using state-of-the art languages and tools
  - Theory/Language builders: to solve a (difficult) problem P, design a sufficiently abstract/high-level lang./theory inside which P has an easy solution

    There is no difficult problem, just inadequate languages

- Language of Thought hypothesis: The language we use determines the way we think

⟹ Crafting the right language/paradigm is probably the best way to have correct software

EXAMPLE

Need to write P : data → out s.t. { differentially private / information leakage

Approach: Design a language with a linear type system

P : $!_m$ data ⊸ output — static semantics

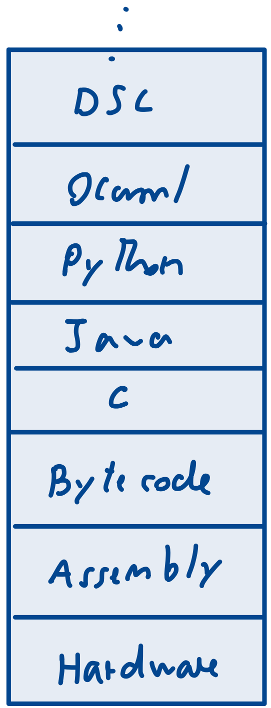
track data usage and resources (⇝ cf. RUST)

Theorem. If P is typable, then differentially private

↳ checked (at least partially) by compilers/interpreters

↳ Language-base correctness → This the reason why software is getting better (although still much to do...)

ASIDE. Main (meta) methodology in Computer Science is
Handling complexity through linguistic abstraction



Key principle: compositionality / modularity

1. Level n hides complexity of level $n-1$
2. To understand level n, it is enough to refer to level $n-1$ only.

TENTATIVE PLAN OF THE COURSE

Week 1, Week 2 } Intro to OCaml (functional programming)

Week 3-4 } FOUNDATIONS (λ-Calculus, Types)

Week 4-5-6 }

① From λ-Calculus to Core OCaml

- Introduction to PL semantics

(syntax) $e ::= x \mid \lambda x. e \mid e(e) \mid \text{let } x = e \text{ in } e \ \ldots$

(static sem.) $\Gamma \vdash e : \tau$

(dynamic sem.) $\Sigma \triangleright e \rightarrow e'$

MATHEMATICAL THEORY OF PLs

② Implementation of Core languages in OCaml

- Interpreter eval : exp $\rightarrow$ val

Week 7-8 } MAIN concepts of Object-Oriented programming (Java)

Exam

Written exam
- λ-Calculus
- Type systems
- Functional programming
- Interpretation of core lang.

(↓ increasing complexity)

Oral exam
- Object-oriented

How to prepare PdP ?

- A difficult course, but software is becoming socially critical: we need to be mathematically-oriented
- Slides & Lectures are NOT enough
  - You need to make your hand dirty
  - Write a lot of (functional) code; implement interpreters
  - Many sources on the web; use them!
    ↳ ChatGPT

READING MATERIAL

- Slides & Handouts of Modulo B on e-learning (register!)
  - ↳ same topics; different order
- Slides used in class
- Notebook with Youtube video
  - OCaml Programming: Correct + Beautiful + Efficient
- Selected Book chapters (?)