Foundations of Programming – Overview

- This course will cover calculi, languages and fundamental techniques of programing.
- 3 Streams:
  - Theory: Syntax and Semantics of Programming Languages
  - Applications: Core language which illustrates essential concepts: Funnel.
  - Practice: Programming Examples, Interpreters.

Goals of this course

- Better understanding of programming languages:
  - Which concepts are essential?
  - Which are ephemeral?
  - How can fundamental concepts encode derived ones?
- Better understanding of programming:
  - Fundamental composition principles
  - Language as a means of abstraction
  - Interpreters and compilers
- Better understanding of definitions of programming languages
  - Which things can/should be formalized?
  - What techniques are available for formalization?

$\Rightarrow$ Increased competence as a programmer, language implementer, library and language designer.

Language Definitions

- A programming language is defined by its *syntax* and *semantics*.
- Language: A set of strings over a given alphabet.
- Syntax: The set of rules which determines whether a given string is a member of the language (i.e. is legal according to the rules of the language).
- Syntax is usually split into
  - Context-free syntax – what can be described by a context-free grammar.
  - Context-dependent syntax – what can't. Examples: scope rules, type systems.
- Semantics tells us what the meaning of a legal program string is.

We have covered context-free syntax in Compiler Construction.

This course will be mainly concerned with semantics and also with context-dependent syntax.

Semantics

There are several different ways of assigning meaning to programs.

- Operational – by specifying evaluation rules. Two main flavors:
  - Abstract machine: Translate programs into an hypothetical machine and explain translation rules, and machine execution.
  - Rewrite or transition system: Rewrite the program itself.
- Denotational – by specifying a translation from program strings to some other domain, which is well understood.

  Example: Functions in a program $\rightarrow$ mathematical functions.
- Axiomatic – by stating laws that programs in the language satisfy.

We will learn about

- operational semantics of functional and concurrent programming
- axiomatic semantics of a simple imperative language
- denotational semantics by translation of derived constructs into our core languages.

Programming Paradigms

This course will cover a large spectrum of programming paradigms

- Functional programming
- Imperative programming
- Object-oriented programming
- Concurrency
- Logic programming

All 5 styles will all be expressed as *functional nets*, using our Funnel notation.

Organization of the Course

Two courses (interlinked)

- Doctoral school: 2 hours per week (theory)
- Full course for 2nd cycle: 6 hours per week (theory + applications + practice). (roughly 2 hours each).
- Proposed times:

Doctoral:	
Mondays	10.15 -12.00 (INR219), or
Tuesdays	10.15-12.00 (INM010)
Full course:	
Tuesdays	10.15 – 13.00 (INM010)
Thursdays	10.15 – 13.00 (INR322)

- Mondays, resp. Tuesdays: Theory and applications in INM010
- Thursdays: Applications and programming in INR 322

Material

- Web site: `http://lampwww.epfl.ch/courses/fondements01/`.

  This contains pointers to everything you need, including preprints of the transparencies (but they might be uploaded late because we are developing this course ”just in time”.)
- Papers: We’ll add references to the site as needed. For the first part of the course there is one paper that you should read:
  - An Overview of Functional Nets. M. Odersky. Gives an overview of our programming notation and the concepts behind it.

  This is available by http from our resources page (which is linked into root)

- Books: No book covers the course as a whole, but here are three books I recommend:
  - Structure and Interpretation of Computer Programs. Harold Jay Abelson and Gerald J. Sussmann with Julie Sussmann. MIT Press, 2nd Edition 1996.
  - Essentials of Programming Languages. David Friedman, Mitchell Wand and Christopher Haynes, Wand. MIT Press. 1992.
  - The Structure of Typed Programming Languages. David Schmidt. MIT Press 1994.

  If you want to read just one book, go with the first, it's worth it.

Questions?