Functional Programming Languages

Luca Abeni

`luca.abeni@santannapisa.it`

Functional Programming

- Programming paradigm
  - No mutable state
  - Pure functions
  - Recursion, not iteration!
  - Function invocation, not sequences of commands
- Currying
- High-order functions
- Algebraic datatypes / pattern matching

Functional Programming Languages

- Not only support the functional programming paradigm...
- ...But also try to enforce it!
- Different kinds of functional languages
  - Pure functional programming languages: really no side-effect
    - What about I/O and similar?
  - Impure functional programming languages: allow some side-effects (for example, for I/O)

Pure Functional Programming: Haskell

- No compromises: no mutable state, functions are pure, etc...
    - I/O performed through "*actions*" (see the I/O monad!)
- Syntactic sugar: allow to write programs that *look* imperative (see the `do` notation)
- Lazy evaluation of functions; eager evaluation of actions
    - Remember that `bind` serializes!

Non Pure Functional Programming

- Lots of functional programming languages providing impure I/O functions
  - ML family
  - Lisp family
  - ...
- Must evaluate eagerly at least the I/O functions...
- Different amounts of compromises on how much impure the language can be...
- Still based on functional reduction (no sequence of commands, generally no mutable variables, ...)

Multi-Paradigm Languages

- Imperative or OO languages with some (or many!) functional features
    - High-order functions
    - Lambda expressions
    - Currying
    - Maybe algebraic data types / pattern matching...
- They do not force to use the functional paradigm
- Examples: C++, Scala (and even modern Java)