Programming language design and analysis

Introduction

Marius Minea

29 September 2010

Why this course ?

Programming languages are fundamental
and one of the oldest CS fields

Language design is an important current issue
mainstream languages still appear and evolve (Java, C#, ...)
+ lots of domain-specific languages

Language design impacts software design (polymorphism, reflection, ...), security (type safety), efficiency (compilation ...), etc.

Analysis is needed in verification, testing, parallelization, certification, performance estimation, ...

Course goals

SIGPLAN motto: "To explore programming language concepts and tools focusing on design, implementation and efficient use."

Know the landscape of programming languages

Understand language features and impact of design decisions

Learn language/program analysis techniques (semantics, reasoning)

Introduction to current programming language research

Words of wisdom

"a programming language is a tool which should assist the programmer in the most difficult aspects of his art, namely program design, documentation, and debugging"

[Hoare, Hints on programming language design, 1973]

Potential papers

Main programming language conferences (ACM SIGPLAN)

PoPL: Principles of Programming Languages

PLDI: Programming Language Design and Analysis

OOPSLA: Object-Oriented Programming, Languages, Systems and Applications (now: SPLASH)

All of them have "most influential paper award" (10 years later)
+ best paper award (current year)
+ 20 years of PLDI (1979-1999)

Other potential topics

symbolic computation
lazy evaluation, closures, higher-order functions and continuations,
concurrency, inter-process communication and synchronization,
active objects and mobile agents,
object views, directed interfaces, and dynamic type systems,
reflection and introspection
persistent object systems and garbage collection,
error management, assertions and declarative debugging,
aspect-oriented programming,
generative programming,
constraint imperative programming,
staged compilation and virtual machines

For a start: small is beautiful

Functional programming

  simple mathematical foundation: lambda calculus (possibly typed)

  in pure form avoids *state* and *mutable data*

"The determined Real Programmer can write functional programs in any language"

(paraphrasing Ed Post)

Exercise 1: program without state and variables in C

Exercise 2: simulate state and an interpreter in ML

Keywords to continue

paradigms concepts first-class functional closures continuations lambda calculus reductions eager lazy evaluation binding

What is programming ?

Programming encompasses three things:

1. a computation model:
   a formal system that defines a *language* and how it is *executed on an abstract machine*

2. a set of *programming techniques* and *design principles*
   used to write programs in that language

3. a set of *reasoning techniques* for reasoning about programs and calculating their efficiency

[vanRoy & Haridi, Concepts, Techniques and Models of Computer Programming]

Paradigms and concepts

programming paradigm = approach to programming based on a mathematical theory or a coherent set of principles

many languages

⇒ fewer paradigms

⇒ still fewer concepts

Key concepts form a paradigm's *core (kernel) language*

Functional paradigm

Evaluate an expression and use the value for something

Discipline and idea:
  Mathematics and the theory of functions

Values produced are non-mutable
  Impossible to change part of a composite value
  But can make a revised copy of composite value

Atemporal: no matter when done, computation produces same value
  pure functional programming is side-effect free

Applicative: all computations done by applying (calling) functions

Natural abstraction: the function
  abstracts expression to a function which can be evaluated as an expression

Functions are first class values: full-fledged data just like numbers, lists, ..

Computations driven by needs

after K. Normark, course, Aalborg U.

First-class objects

A first-class object is one that can be:
passed as an argument
returned as a value, and
stored in a data structure.

What is first-class influences your choices of abstraction:

In languages where functions are first-class, can represent data as procedures.

Example: represent *environment*
2 constructors: empty environment, enlarge environment with $(symbol, value)$ pair
1 observer: give value of symbol in environment

Advantages of Simplicity

Functional / declarative operations are:

independent (do not depend on any external execution state)
stateless (no internal execution state remembered between calls)
deterministic (same result when given same arguments)

Why is functional programming important ?

Declarative programs are compositional

Reasoning about declarative programs is simple [van Roy & Haridi]

Learn by interpreting

"This book brings you face-to-face with the most fundamental idea in computer programming:

The interpreter for a computer language is just another program"

Hal Abelson

foreword to Friedman, Wand & Haynes, *Essentials of Programming Languages*

makes you think about fundamental *concepts*

defines the meaning of programs: *semantics*