

Programming Languages, CPSC 604—Slides 1

Introduction

Jaakko Järvi

January 19, 2009

Outline

- 1 Organization
- 2 Course Overview

604: Programming Languages

- Evaluation
 - Homework assignments, quizzes 40%
 - Projects and/or exams 60%
 - Lack of attendance can affect negatively up to 10%
- Course homepage:
<http://courses.cs.tamu.edu/jarvi/2009/CPSC-604/>
 - Secret questions may be needed in some parts of the cite: user:
programming password: languages

Material

- Lectures mostly based on book:

Benjamin Pierce: Types and Programming Languages, MIT Press.

- The book is [required](#), a few additional handouts
- The main focus of the book is static typing, but it serves to introduce the formal side of programming language research
- We'll hop around some in the book, details along the way

Assignments

- There will be some number of assignments
 - Solutions to be submitted through CSNET, or in paper, to be instructed later
 - There will be a deadline for each assignment
 - Some rules, such as late work penalties to be described on the web pages and/or in the assignment
- Code of conduct
 - See Aggie honor code!
 - If pairs are allowed, state partner in the answer (only one submission per pair)

Software

- We'll be implementing some evaluators and type-checkers
- The language we use is Haskell
 - See: <http://haskell.org>
- Learning Haskell is not the main focus of the course, but a nice bonus.
- One possibility for editing Haskell is with Emacs. Emacs provides “haskell-mode” and “haskell-indent-mode”.
- The de facto compiler/interpreter is the **Glasgow Haskell Compiler (GHC)**: <http://haskell.org/ghc/>
 - Packaged to Linux distributions, MacPorts, etc.
- Also, using \LaTeX for typesetting your assignments is highly recommended. More guidance on this provided later.

Questions etc.

- Please do ask questions during the lectures
 - repeat an explanation?
 - give a better explanation?
 - give an example?
- Please say when things go too slow or fast!
- Please answer my questions, even if you think they are so obvious that everyone knows the answer. Only if you do not know the answer, you have the permission not to answer !!!

Outline

1 Organization

2 Course Overview

10/50 Questions for a Prospective Language Designer

Source: Scott McKay, based on Usenet thread

- What need are you trying to fill?
- What is the metaphor (OO, lazy functional, ...)?
- Is high performance an issue?
- Debuggability?
- Catch type errors early or late?
- Explicit type annotations or not?
- What kind of a module system?
- Pure/imperative/...?
- OO or not? Class or prototype based?

... continued

- Reflection capabilities?
- Genericity, continuations, threads, ...
- Memory management (GC or not)?
- Order of evaluation of expressions?
- Real syntax that must be parsed, or just use S-expressions? (Or directly operate on the AST).
- Syntactic extensions (macros, hygienic/cpp like)?
- ...

This was already more than 10, and there's really much more than 50 questions. \Rightarrow PL is a very large topic, not all can be covered

Aim of the course

- Knowledge of
 - the fundamental concepts of programming languages
 - mechanism for specifying the semantics (and syntax) of programming languages.
 - formal reasoning on properties of languages
- Focus on the fundamentals, not on particular language
- Also, focus not too much on language processing (parsing, semantic analysis, code generation, ...).
- Not a compilers course. You should take that too!
- Not an overview and comparison of languages or programming paradigms course.

... Aim of the course

- Many languages have been designed with good craftsmanship, engineering, hacking, committees, semi-formal reasoning, etc.
 - and very successfully in deed
 - C++, Perl, Python, ... also Java, C#, ..., even Haskell
- In fact, very few languages arise directly from mathematical ideals (ML?)
- Climate is changing, however. Java generics and wild-cards, C# generics are backed up with heavy formal reasoning. Similarly, many features in Haskell. And in C++ to an extent too.
 - Siek, Taha: formal treatment of C++ templates (in ECOOP06)
 - Tip et al. formal treatment of C++ multiple inheritance (in OOPSLA06)
- Major component of programming language research today is modeling languages as mathematical objects and reasoning formally on those models

... Aim of the course

- Programming language research is experiencing very active and exiting times.
- You want to be part of these activities, understand them and contribute!
- This requires background, and that's why you are here.

A quote on formally reasoning about languages

*Q: Why bother doing proofs about programming languages?
They are almost always boring if the definitions are right.*

A quote on formally reasoning about languages

*Q: Why bother doing proofs about programming languages?
They are almost always boring if the definitions are right.*

*A: The definitions are almost always wrong. [Benjamin Pierce
2002]*

Course outline

- Basic skills
 - Specifying semantics of programming languages (and a bit about syntax too)
 - Proof techniques for programming language research
 - Lambda calculus
 - Specifying type systems
 - Formal study of properties of type systems and languages (such as type safety)
- Using the basic skill set
 - Studying more ambitious language constructs, such as
 - Polymorphism and generics
 - Constrained generics (bounded quantification)
 - Modeling a (subset of a) real language (Java) formally
- Recent interesting developments in programming languages