15-312: Foundations of Programming Languages

Lecture 1

Overview

Frank Pfenning
Carnegie Mellon University

August 27, 2002

http://www.cs.cmu.edu/~fp/courses/312/
Teaching Staff

- Frank Pfenning <fp@cs.cmu.edu>
- Office Hour: Wed 2:30-3:30, WeH 8117
- Joshua Dunfield <joshuad@cs.cmu.edu>
- Office Hour: TBA, WeH 1313
- Course web page
  
  http://www.cs.cmu.edu/~fp/courses/312/

- Blackboard area only for grade sheet, occasional announcement
Outline

• The Science of Programming Languages
• Our Approach
• Topic Overview
• Assignments and Exams
• Recitation
• Summary
Factors in Programmer Productivity

- Programmer productivity
  - Initial development time
  - Program correctness and robustness
  - Software maintainability

- Crucial factors
  - Programming language(s)
  - Development environment
  - Software engineering practices
Language Is Critical

- How do we implement data structures?
- How do we design and structure the code?
- How do we express assumptions and guarantees?
- How do we read and analyze a program?
Two Quotes

An ideal language allows us to express easily what is useful for the programming task and at the same time makes it difficult to write what leads to incomprehensible or incorrect programs.

—Nico Habermann

Good languages make it easier to establish, verify, and maintain the relationship between code and its properties.

—Robert Harper
Too Many Languages?

• In the last two years I have written code in at least the following languages:

  Standard ML  Emacs Lisp  Twelf
  TeX      Csh      C
  Perl     Java     CML
  HTML

• Different languages for different purposes

• Many are poorly designed
  – The authors did not take 15-312!
  – Your favorite mis-feature?
Language Evaluation Criteria

• Some objective criteria
  – Is the grammar LALR(1)?
  – Is the language type-safe?
  – Is the language dynamically or statically typed?
  – Is the language Turing-complete?
  – Is the language call-by-value, call-by-name, or call-by-need?
  – Is the language completely specified?
  – Does the language implementation require a heap?
  – Does the language require dynamic dispatch?

• A subjective statement
  – (I ((like Lisp)) (syntax))
When presented with something that might have several different interpretations, Perl uses the DWIM (that’s ”Do What I Mean”) principle to pick the most probable interpretation. This strategy is so successful that Perl programmers often do not suspect the ambivalence of what they write. But from time to time, Perl’s notions differ substantially from what the author honestly meant.
From the TeX manual

*Please don’t read this material until you’ve had plenty of experience with plain TeX.* After you have read and understood the secrets below, you’ll know all sort of devious combinations of TeX commands, and you will often be tempted to write inscrutable macros.

—Donald E. Knuth
Some Obfuscated \TeX Code

\let\catcode\'76\'A13\'F1\'j00\'P2jdefA71F\'7113jdefPALLFPA\'FwPA;\FPAZZFLaLPA//71F71iPAHHFLPAzzFenPASSFthP;A$$FevPA@@FfPARR717273F737271P;ADDFRgniPAWW71FPATTFvePA**FstRsamPAGGFRruoPAqq71.72.F717271PAYY7172F727171PA??Fi*LmPA&&71jfijfi71PAVVVFjbigskipRPWGAUU717273734 75,76Fjpar71727375Djifx:76jelse&U76jfifiPLAKK717271PAXX71FVLn0SeL71SLRyadR@oLRrhC?yLRurtKFeLPFovPgaTLtReRomLPABB71 72,73:Fjif.73.jelseB73:jifiXF71PU71 72,73:PWs;AMM71F71diPAJJFRdriPAQQFRsreLPAI71Fo71dPA!!FRgiePBt’el@1TLqdrYmu.Q.,Ke;vz vzLqpip.Q.,tz;Lql.IrsZ.eap,qn.i.i.eLlMaesLdRcna,;!;h htLqm.MRasZ.ilk,%s$;z zLqs’.ansZ.Ymi,/sx;LYegseZRyal,@@TLRlogdLrDsW,@@;GLcY1aDLbJsW,SWXJWree@rzchLhzsW,;WERcesInWqt.’oL.Rtrul;edOTsW,Wk;Rri@stW aHAHHFndZPpqar.tridgeLinZpe.LtYer.W,:jbye
Some Obfuscated C Code

- Prior $\TeX$ course in obf-tex.tex
- See obf-tex.pdf for result of pdftex obf-tex.tex
- Also see separate source obf-c.c
- See output obf-c.txt
There is an established science of programming languages. Among its first papers:


Some basic tools

- **Type theory**: Techniques for structuring languages to ensure safety and modularity of programs
- **Operational semantics**: Techniques for describing the execution behavior of programs, at various level of abstraction
- **Mathematical logic**: Techniques for specifying and verifying programs
Outline

- The Science of Programming Languages
- Our Approach
- Topic Overview
- Assignments and Exams
- Recitation
- Summary
Approach I: Vivisection

- Take one or several living languages, preferably widely used
- Analyze it or them in minute detail
  - Syntax: Grammar and parsing
  - Semantics: Type-checking and operational semantics
  - Pragmatics: Programming methodology and implementation strategies
- Practice dialectic mathematics
- Can be interesting and instructive
- Not our approach
Approach II: Autopsy

- Take one or several **dead** languages, preferably used
- Analyze it or them in minute detail
  - **Syntax:** Grammar and parsing
  - **Semantics:** Type-checking and operational semantics
  - **Pragmatics:** Programming methodology and implementation strategies
- Practice dialectic mathematics
- Can be interesting and instructive
- **Not** our approach
Approach III: Genesis

- Take a problem domain, preferably useful
- Design the ultimate language
  - **Syntax**: Grammar and parsing
  - **Semantics**: Type-checking and operational semantics
  - **Pragmatics**: Programming methodology and implementation strategies
- Practice dialectic mathematics
- Can be interesting and instructive
- **Not** our approach
Approach IV: Taxonomy

- Analyze many languages based on few criteria
- Create taxonomy of (living or dead) languages
- Can be interesting and instructive
- **Not** our approach
Approach V: Study Basic Concepts

- Ignore issues of syntax (largely)
- Isolate and investigate basic concepts, for example,
  - Functions, procedures, and variables
  - Classes, objects, and methods
  - Effect-free vs. imperative programming
  - Static vs. dynamic typing
  - Concrete vs. abstract types
  - Sequential vs. concurrent vs. parallel programming
- Emphasize mathematical tools
- **This is our approach!**
Our Approach and Goals

• Not bound by flaws or limitations in actual languages
• But can draw conclusions about actual languages
• After this course, you should be able to
  – confidently critique existing languages
  – define and analyze your own language
  – prove properties of languages
  – avoid common mistakes and pitfalls
  – reflect more deeply on programming style
  – write better programs(?)
  – carry out research on programming languages
Outline

- The Science of Programming Languages
- Our Approach
- Topic Overview
- Assignments and Exams
- Recitation
- Summary
Core Topics

• Mathematical foundations
  – Judgments and inductive definitions
  – Variable renaming and substitution
  – Structural induction

• Language description techniques
  – Concrete and abstract syntax
  – Static semantics via type systems
  – Dynamic semantics via abstract machines
  – Type safety and its consequences
Language Features (Tentative)

- Continuations
- Exceptions
- Mutable storage
- Monads
- Parallelism
- Polymorphism
- Data abstraction
- Laziness
- Dynamic typing
- Subtyping
- Inheritance
- Concurrency
- Storage management
- Refinement types
Course Reading

- Handouts will be provided, mostly from

  *Programming Languages: Theory and Practice.*

- Notes complement, but do not replace lecture!

- Supplementary reading

  *Types and Programming Languages.*
  *Benjamin C. Pierce.*
Outline

• The Science of Programming Languages
• Our Approach
• Topic Overview
• Assignments and Exams
• Recitation
• Summary
Written Assignments

- Alternating written (4) and programming (4) assignments
- Integral part of this course
- Schedule see web page
- Written assignments:
  - Total 200/1000 points (20%)
  - 1 week assignments
  - Hand in **before lecture** on due date
  - Graded on correctness and thoroughness
Programming Assignments

- Total 450/1000 points (45%)
- 2 week assignments
- Hand in by midnight on due date
- Graded for correctness, style, and documentation
- Implementation language is Standard ML
- Possibly using electronic hand-in pages (not Blackboard)
Assignment Policies

- 3 late days without penalty for each student
- Spread throughout the semester
- Can be used for written or programming assignments
- No other late hand-ins permitted
- No group projects—all work must be your own!
Examinations

- **Midterm**
  - Thursday Oct 17, in class
  - Closed book, one double-sided sheet of notes
  - Total 100/1000 points (10%)

- **Final**
  - Date and time TBA
  - Open book
  - Total 250/1000 points (25%)
Outline

- The Science of Programming Languages
- Our Approach
- Topic Overview
- Assignments and Exams
- Recitation
- Summary
Recitation

- Each Wednesday in two sections
- Practice technique from lectures
- Discuss assignments
- Occasionally covers new material
- See schedule on web page
Outline

• The Science of Programming Languages
• Our Approach
• Topic Overview
• Assignments and Exams
• Recitation
• Summary
Summary

- Language is critical for programmer productivity
- The good, the bad, and the ugly
- Rigorous study of programming languages with mathematical tools
  - Type theory
  - Operational semantics
  - Mathematical logic
- No vivisection, genesis, or taxonomy, but thorough investigation of basic concepts
- Combine theory (proof) with practice (implementation)