CS422 - Programming Language Design

General Information and Introduction

Grigore Roşu

Department of Computer Science
University of Illinois at Urbana-Champaign

General Information

- Class Webpage:
  http://fsl.cs.illinois.edu/grosu (go to "Teaching")
- Lectures: Wednesday/Friday 14:00 - 15:15, 1304 Siebel Center
- Office hours: 2110 Siebel Center, by appointment
- Instructor: Grigore Roşu
  - Office: 2110 Siebel Center
  - Email: grosu@illinois.edu
  - WWW: http://fsl.cs.illinois.edu/grosu
- Prerequisites: CS421 or equivalent, or instructor's approval

- Textbooks

  No textbook required! Self contained lecture notes will be posted on class' webpage. The following may be useful:

  1) Friedman, Wand and Haynes, *Essentials of Programming Languages*, MIT Press, Second Edition, 2001
  2) Winskel. *The Formal Semantics of Programming Languages: An Introduction*, MIT Press, 1993

- Other sources
  - The K Framework: `http://kframework.org`
  - Proceedings of Conferences on Programming Languages
    * **POPL**, **PLDI**, **OOPSLA**

Grading

- Students registered for 4 units
  - Assignments (or MPs): **75%**
  - Individual project: **25%**
- Students registered for 3 units
  - Assignments: **100%**

The Homework Assignments

- This is a *labor intensive class*. The notions presented in class will be often backed by machine supported formalizations which you are supposed to modify or redo entirely as part of your assignments and as part of your project
- Assignments will be complete approximately every 4 lectures

The Unit Project

- The unit project will consist of designing a new programming language with specified features. This language will most likely be an extended version of an existing language. The design will be formalized and an interpreter will be provided, which we will test against many carefully selected programs.

No Final Exam!

Course Description

- Advanced course on principles of programming language design
- Major semantic approaches to programming languages will be introduced
- Major programming language design paradigms will be investigated and mathematically defined (or specified)
- Since the rigorous definitional framework will be *executable*, *interpreters* for the designed languages will be obtained *for free*
- Software analysis tools reasoning about programs in these languages will arise naturally
- Major theoretical models will be discussed

Tentative Subjects Covered in CS422

- Big-step and Small-step Structural operational semantics (SOS)
- Defining/designing a simple programming language using the formalisms above; discussing possible extensions of the simple language to reflect limitations of the formalisms above
- K, a tool-supported language definitional framework.
- Defining SIMPLE, a simple C-like imperative language with functions, together with type checker
- Defining KOOL, an object-oriented language; different method dispatch styles discussed, such as static versus dynamic method dispatch; defining typed and untyped variants of KOOL
- Defining FUN, a functional programming language; different binding styles discussed, such as static versus dynamic binding,

as well as different parametric passing styles, such as call-by-value, reference, name, need; defining static and dynamic type checkers, and a type inferencer for FUN

- Defining LOGIK, a simple logic programming language
- Other subjects covered include: continuation-passing style transformations, exceptions, concurrency, denotational semantics, lambda-calculus.

Course Objectives

- Present and define rigorously the major features and design concepts in programming languages
- Show how elegantly and easily one can design a programming language if one uses the right tools and framework
- Understand the significant theory of programming languages
- The practical objective of this course is *not* to implement programming languages, but rather to *specify*, or *define* them formally and modularly, on a feature by feature basis
  - Interpreters will, however, be obtained for free, because in the discussed framework one can *execute* specifications
  - For that reason, we take the freedom to interchangeably use the words *define* and *implement* in this course

Important Notes and Advice

- The lecture notes for this class will be all posted on the web and will be as detailed as needed. No other textbooks are necessary, though those of you interested in the covered topics may find it useful to check other books as auxiliary material in order to have a better understanding of the discussed concepts.
- We will *not* use Scheme, ML, OCAML or Haskell in this class as implementation languages! These are quite advanced programming languages; using them to *implement* interpreters hides some of the real important and interesting issues in specifying a programming language.

Collaboration and Other Policies

- You are free (and encouraged!) to discuss the assignments with other students. The focus of any such discussion should be limited to figuring the problem specification, not coming up with a solution. You may *not* jointly write or code any assignment. To do so will be considered cheating! All cheating will be penalized by automatically assigning a failing grade for the course and instigating further disciplinary action with the appropriate university disciplinary body.
- You should retain copies of your assignments until you receive your final grade. In the event of a discrepancy between your scores on assignments and those on the exams, you may be asked to explain any work you performed. Your grade may be adversely affected by an inability to explain your work or by failure to retain copies of it.