

CS 502: Compiling and Programming Systems

Handout 1

August 27, 2008

Instructors

Lecturer

Prof. Suresh Jagannathan
Room LWSN 3154J
Ph: 49-40971
suresh@cs.purdue.edu

Teaching Assistant

Mr. Yu Zhang
Room HAAS 145
Ph. 49-69444
zhang106@purdue.edu

Office Hours:

Tu, Th. 11 - 12.

Office Hours:

Tu. 4:15 - 6:15.

Course Overview

This course will study both foundational and pragmatic issues in the design and implementation of language translators for modern programming languages. The course is divided into roughly three parts, with the first two parts forming the bulk of the course material.

1. The first part of the course will study high-level intermediate representations such as CPS (continuation-passing style) and A-normal form that provide a canonical representation of a program's control-flow. While our focus will be mostly on functional programs (e.g., Scheme, ML, Haskell, Erlang, etc.), we will also study intermediate representations suitable for other language paradigms (e.g., object-oriented languages).
2. The second part of the course will discuss inter- and intra-procedural program optimizations. Besides examining program representations like SSA for describing local control-flow, we will also study more general inter-procedural flow analysis frameworks, and related optimizations and analyses. A significant focus will be placed on program analysis frameworks (e.g., monotone dataflow frameworks), and the theory underlying their definition.
3. The third part of the class will explore backend translation issues, calling conventions, register allocation, virtual machine and runtime design. Time permitting, we will also study advanced techniques such as type-based compilation, compiling concurrency and distributed programming primitives, etc.

Prerequisites

It is assumed that students taking this class have already taken an undergraduate compiler class (equivalent to CS 352), and are comfortable with basic programming language theory (e.g., parsing, interpreters, etc.) Students should also be fluent in basic graph and set theory, and should be able to translate high-level algorithmic descriptions to efficient code.

Students who have not taken an undergraduate compiler class are *strongly discouraged* from taking this class. It is also expected that students have previously programmed a large application (on the order of 5K lines of code, comprising 25-30 modules). Students unsure about their qualifications should see the instructor.

Project

There is a significant programming project associated with this class. The project will be to implement a subset of ML (called Mini-ML) using Standard ML as the implementation language. Both SML and Mini-ML are mostly-functional strongly-typed higher-order programming languages. Students who are unfamiliar with SML will need to quickly master its salient features to successfully complete the course.

Supporting handouts will provide details about the project, and there will be time devoted in class to discuss relevant details.

The grade you receive for this project will consist of three factors:

1. **Correctness:** The compiler must generate verifiably correct code;
2. **Documentation:** You must comment your code copiously, explaining the intended role of every non-trivial procedure, its expected inputs and outputs, and exceptional conditions it handles. If we are unable to understand what your code is supposed to do, it will not be possible to give credit for it even if it appears to produce the correct answers;
3. **Test cases:** You must supply test cases which will convince us of the correctness of your implementation. You should take care to also test exceptional conditions to ensure that the implementation correctly handles error conditions.

Grading

Grading for this class will be based as follows:

Midterm	30%
Final	30%
Project	40%

Texts and Papers

There is no required text for this class. Instead, we will use a combination of recommended textbooks (listed below), and a reading list comprised of both classic papers in the area, and more recent research contributions. The reading list will be updated throughout the semester and will be available from the course web page. While you are expected to read the papers listed, you will only be tested on material presented in class.

The following recommended texts are on reserve in the Math Library. Except for the first, none of the material found in the other texts will be directly referenced in the lectures.

- **Principles of Program Analysis**, Flemming Nielson, Hanne Riis Nielson, and Chris Hankin, Springer-Verlag, 2005.
- **Modern Compiler Implementation in ML**, Andrew Appel, Cambridge University Press, 1998.
- **Engineering a Compiler**, Keith Cooper and Linda Torczon, Morgan-Kaufmann, 2003.
- **Advanced Compiler Design and Implementation**, Stephen Muchnick, Morgan-Kaufmann, 1997.
- **Building an Optimizing Compiler**, Robert Morgan, Digital Press, 1998.
- **Compiling with Continuations**, Andrew Appel, Cambridge University Press, 1992.
- **Essentials of Programming Languages**, Daniel Frideman, Mitchell Wand, and Christopher Haynes, MIT Press 2001.

Resources

There are numerous resources available to help you gain greater familiarity and fluency with SML and compilation techniques for functional languages; some of the ones you might find particularly useful are listed below:

1. <http://www.dcs.napier.ac.uk/course-notes/sml> and <http://www.dcs.ed.ac.uk/home/stg/NOTES> are online tutorials on SML.
2. Details about Standard ML of New Jersey (SML/NJ), the SML implementation supported on the department servers can be accessed at <http://www.smlnj.org>.
3. The SML basis library contains signatures for a number of SML structures that are part of the SML/NJ release. It can be found at: <http://www.standardml.org/Basis>.
4. Robert Harper at CMU has extensive notes on programming in Standard ML; you can find his lecture notes at www-2.cs.cmu.edu/~rwh/smlbook/online.pdf.
5. Peter Lee has some useful notes on using SML/NJ at: <http://www-2.cs.cmu.edu/~petel/smlguide/smlnj.htm>.
6. MLton is a whole program optimizing compiler for Standard ML whose compiler structure closely mirrors the one we will use for our project. Details about MLton can be found at <http://www.mlton.org>.
7. You are encouraged to review past proceedings of the ACM Principles of Programming Languages (POPL), ACM Programming Language Design and Implementation (PLDI), and the ACM International Conference on Functional Programming (ICFP). These conferences all publish articles related to compilation.

Web Pages and Newsgroups

Copies of the lectures and handouts will be available through the course web page:

<http://www.cs.purdue.edu/homes/suresh/502-Fall2008/index.html>.

There is also a mailing list for the class that will be used to post last-minute announcements, bug fixes, etc. Students may also use this list to post questions or raise issues that may be of interest to the class as a whole. Students should subscribe to 502-Fall2008 to join this list.

Tentative Syllabus

<i>Lectures</i>	<i>Topics</i>
1 – 10	Introduction and Course Overview SML and Mini-ML Pattern-matching compilation Type checking and inference Continuation-passing style and Static Single Assignment
11 – 23	Closure and environment representations Dataflow analysis: foundations and optimizations Shape analysis, Points-to and alias analysis
24 – 30	Virtual machine design Data layout and representations, garbage-collection Register allocation and code generation Type-based compilation, intensional polymorphism Course review