Computer Science / Linguistics 066
Mathematical and Computational Foundations of Linguistics
Spring 2006

Instructor & Text

Instructor: Kim Bruce
Lectures: MWF 10:00 - 10:50 a.m., Millikan 213
Lab: Windows Lab, Andrews 253
Office: 212B Millikan, x7-1866
E-mail: kim@cs.pomona.edu
Office Hours: MThF 1:15-2:15 p.m., W 9:00 a.m. - 9:50 a.m. & by appointment


Course web page: www.cs.pomona.edu/classes/cs066/
Instructor’s web page: www.cs.pomona.edu/~kim/

Recommended Logic Text:


Recommended Texts for ML:


Recommended Math of Linguistics:

Overview

This course is concerned with representational and computational issues relevant to the formal study of natural languages. A solid understanding of logic, lattices, and the lambda calculus will help us understand how to represent linguistic phenomena and to compute the meaning of discourse in natural languages. Logic provides an important first step, but a compositional understanding of language requires more powerful computational tools like the lambda calculus that will allow us to build up the meaning of phrases. Some programming will be taught so that students can understand computational models of languages. Previous programming experience or experience in linguistics is not required.

Some of you come in with background in Computer Science, some with background in Linguistics, some with neither, and some with both. I will attempt to reach out to all of you and hope that you will work together with people whose backgrounds are different from your own. In particular, the course will have a final project, and I would prefer to see students work together in pairs which include different backgrounds and strengths.

The goal is to introduce both the technical tools and linguistics, though the focus will be on how the tools make it possible to gain a deeper understanding of formal linguistics. The topics covered in this course are often characterized under the umbrella of computational semantics, though the subject area includes many topics that we will not have a chance to cover.

Lectures and Readings

Because this is the first time this course is being offered, and similar courses do not really exist at other schools, the syllabus at this point is simply a guess on my part. What we cover and how fast we cover it will depend on the class. You should go to the class web page before each class for detailed information on what will be covered in the next class.

All reading assignments are from the required and recommended texts. The notation “BB” stands for the book by Blackburn and Bos, while “G” stands for the logic book by Gamut. I have also included some reading from Paulson’s ML for the Working Programmer in case you have access. These readings will be marked with “ML”. Students should come to class having completed the indicated readings for the day.
<table>
<thead>
<tr>
<th>Lecture</th>
<th>Date</th>
<th>Topic</th>
<th>Reading</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Jan. 18</td>
<td>Introduction</td>
<td>BB Introduction</td>
</tr>
<tr>
<td>2.</td>
<td>Jan. 20</td>
<td>ML programming</td>
<td>ML 1-2.9</td>
</tr>
<tr>
<td>3.</td>
<td>Jan. 23</td>
<td>ML programming</td>
<td>ML 2.10-2.18</td>
</tr>
<tr>
<td>5.</td>
<td>Jan. 30</td>
<td>ML programming</td>
<td>ML 2.19-4.13</td>
</tr>
<tr>
<td>6.</td>
<td>Feb 1</td>
<td>ML programming</td>
<td>ML 2.19-4.13</td>
</tr>
<tr>
<td>7.</td>
<td>Feb 1</td>
<td>First-order logic review</td>
<td>BB 1</td>
</tr>
<tr>
<td>8.</td>
<td>Feb 3</td>
<td>Defining truth</td>
<td>BB 1</td>
</tr>
<tr>
<td>9.</td>
<td>Feb 6</td>
<td>Model checker in ML</td>
<td>BB 1</td>
</tr>
<tr>
<td>10.</td>
<td>Feb. 8</td>
<td>Extending first order logic</td>
<td>BB 1</td>
</tr>
<tr>
<td>11-15.</td>
<td>Feb. 10 - Feb 20</td>
<td>Lambda Calculus</td>
<td>BB 2</td>
</tr>
<tr>
<td>16-21.</td>
<td>Feb. 22 - March 6</td>
<td>Intensional propositional logic</td>
<td>G 1, 2</td>
</tr>
<tr>
<td>22-27.</td>
<td>Mar. 8-27</td>
<td>Intensional Predicate Logic</td>
<td>G 3</td>
</tr>
<tr>
<td></td>
<td>Mar. 13-17</td>
<td><strong>Spring Break</strong></td>
<td></td>
</tr>
<tr>
<td>28-29.</td>
<td>Mar. 29 - April 3</td>
<td>Lattices</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mar. 31</td>
<td><strong>Holiday - Chavez Day</strong></td>
<td></td>
</tr>
<tr>
<td>30-36.</td>
<td>April 5-19</td>
<td>Quantifiers &amp; Scope Ambiguities</td>
<td>BB 3</td>
</tr>
<tr>
<td>37-41.</td>
<td>April 21 - May 1</td>
<td>Final Projects</td>
<td></td>
</tr>
<tr>
<td>42.</td>
<td>May 3</td>
<td>Summary</td>
<td></td>
</tr>
</tbody>
</table>

**ML Programs from class**

This page includes links to ML programs shown in class. They are an essential part of the course and may be used as part of your homework solutions.

- Dictionary.sml Code from “ML for the working programmer” that is used in basis.sml.
• basis.sml Code taken from “ML for the working programmer” that is used in building software for class. Needs Dictionary.sml to run.

• logic.sml Code providing representation of formulas of first-order logic, parser for formulas, and pretty printer for formulas. Formula.formula is the type. ParseFormula.read takes a list of constants and a string representation of a formula and returns an element of type Formula.formula. DisplayFormula.pr takes an element of type Formula.formula and prints it in a human readable form. See signatures FORMULA, DISPLAY_FORMULA, and DISPLAY_FORMULA for details.

• model.sml Code providing representation of models of first-order logic and functions to determine satisfaction. Model.modelType is the representation of models. Model.satisfy takes a model, a formula, and an environment (represented as a list of pairs) and returns a boolean indicating whether the formula is true in the model. See signature MODEL for details.

• sampleModels.sml Code providing examples of how to (parse and) build formulas, models, and determine satisfaction.

Assignments

There will be several kinds of assignments in this course. They include:

• Computer programs (15%)
• Written homework (20%)
• Final project (25%)

The numbers in parentheses indicate their tentative weight in computing the final course grade. Computer programs and/or written homework will be due at least once per week. Final projects, which will be described in more detail later, will be due on the last day of class, with oral presentations being given during the last two weeks of class.

There will also be a midterm and a final exam, each worth 20% of your final grade. The midterm will be take-home, while the format of the final has not yet been determined.

<table>
<thead>
<tr>
<th>Due Date</th>
<th>Homework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan. 30</td>
<td>Hmwk 1</td>
</tr>
<tr>
<td>Feb 3</td>
<td>Hmwk 2</td>
</tr>
</tbody>
</table>

Documentation

The following documents provide information on the SML and its compilers, and are available via the online version of this document:

• Downloading SML
• Using SML mode in emacs
• on-line ML tutorial
• A gentle introduction to ML
• Standard ML basis library documentation

• Emacs tutorials:
  – A tutorial intro to emacs
  – The official GNU emacs tutorial is available from within emacs by typing `M-x help-with-tutorial`

The main things necessary to run sml in Emacs are the following:

• Remember that the “meta” key on Windows is the “Alt” key.

• To enter sml mode either load (“C-x C-f file-name”) a file with the “.sml” suffix or type “M-x sml-mode”.

• To run sml on a buffer, type “C-c C-b”. The first time you do this you might also have to hit the return key if “sml” shows up in the mini-buffer.

• I strongly urge you to use separate windows for your program and for the sml process. Choose “New Frame” under the file menu and then select “*sml*” from the “Buffers” menu.

Collaboration & Academic Honesty Policy

I highly encourage students to get together in small groups to go over material from the lectures and text, work problems from the text, study for exams, and to discuss the general ideas and approaches to laboratory assignments. However, work to be turned in, including programming assignments, must generally be done independently, unless credit is given for all collaboration or assistance. All students should include a statement on all homework detailing all collaboration or assistance given or received. The statement must detail the problems involved, the type of collaboration or help, and the persons involved. If I believe any student is depending too much on others, I will inform them that they must receive less help in the future, but no points will be deducted from homework turned in to that point. All solutions must be written up independently. Copying another person’s solution is always a violation of the academic honesty standards.

Of course exams must always represent only your own work. Before an exam is distributed I will give you careful instructions on what sources, if any, may be consulted during the exam.

Failure to abide by these rules is considered plagiarism, and will result in severe penalties. Violations will be dealt with promptly. The first offense typically results in failure in the course. A second offense is automatically referred to the College’s Board of Academic Discipline. See the Academic Honesty Policy in the Student Handbook for further information. Please do not put me, yourself, or anyone else in this unpleasant situation.

Philosophy behind our approach

The following abstract from a talk by Gerry Sussman at OOPSLA 2005 summarizes my view of the use of mathematical and computational models. While Sussman’s work has primarily been in physics, the sentiments are, if anything, more applicable in linguistics.

Why Programming is a Good Medium for Expressing Poorly Understood and Sloppily Formulated Ideas
We have been programming universal computers for about 50 years. Programming provides us with new tools to express ourselves. We now have intellectual tools to describe “how to” as well as “what is”. This is a profound transformation: it is a revolution in the way we think and in the way we express what we think.

For example, one often hears a student or teacher complain that the student knows the “theory” of some subject but cannot effectively solve problems. We should not be surprised: the student has no formal way to learn technique. We expect the student to learn to solve problems by an inefficient process: the student watches the teacher solve a few problems, hoping to abstract the general procedures from the teacher’s behavior on particular examples. The student is never given any instructions on how to abstract from examples, nor is the student given any language for expressing what has been learned. It is hard to learn what one cannot express. But now we can express it!

Expressing methodology in a computer language forces it to be unambiguous and computationally effective. The task of formulating a method as a computer-executable program and debugging that program is a powerful exercise in the learning process. The programmer expresses his/her poorly understood or sloppily formulated idea in a precise way, so that it becomes clear what is poorly understood or sloppily formulated. Also, once formalized procedurally, a mathematical idea becomes a tool that can be used directly to compute results.

Gerry Sussman, MIT, October, 2005