Computer Science 190 (Spring Term, 2008)
Semantics of Programming Languages

Course instructor: Harry Mairson (mairson@brandeis.edu), Volen 257, phone (781) 736-2724. Office hours Monday and Wednesday, 1–2.30pm and by arrangement. Don't be afraid to bug me in my office—I'm around most of the time. I especially encourage you to communicate with me via electronic mail, for fastest and most reliable responses to your questions. I try to read e-mail every 5 minutes, 24 hours a day.

Teaching assistant: David Van Horn (dvanhorn@brandeis.edu), Volen 136, phone (781) 736-2728. Office hours Tuesday and Friday 12-1.30pm and by arrangement.

Time and place: Tuesday and Friday, 10.40am–12.00pm, Volen 105.

What is this course about?

The main subject of this course is *semantics of programming languages*—how is it that we can give a precise, mathematical meaning to a programming language. We address this issue from the perspective of what's often called the *Curry-Howard correspondence*, named after two of its major proponents, Haskell Curry [1900–1982] and William Howard [1926–]. The fundamental idea is that data types are theorems, and (typed) programs are proofs of theorems. What this correspondence suggests is that programming is a good medium for understanding logic, and *vice versa.*

In contrast, the study of programming language semantics has for the most part been *denotational*, following Gottlob Frege [1848–1925] (and before him, George Boole [1815–1864]). The semantics of logical expressions (or even natural language) is often modelled this way: a piece of syntax denotes some semantic object which is invariant under "equivalent" rewritings of the syntax. An example from computer science is the language (syntax) of *regular expressions*, each of which denotes a *regular set*; that $x^* = \epsilon \cup xx^*$ means each side of the equation denotes the same regular set. This *denotational semantics* associates programs (including higher-order ones) with functions on well-specified domains—typically (infinite) sets with an underlying ordering on its elements. Recursion is interpreted as a limit (in the naive calculus sense) on these sets.

Another sense of semantics had a proponent in Arend Heyting [1898–1980], who wanted not to model the *denotation*, but the *proofs*. For example, though $27 \times 37 = 999$ has a denotational sense, there is a finite computation which *shows* (proves) that the denotations are the same. In other words, study the finitary dynamics. When we look at logical expressions, rather than ask (denotationally) "is ϕ *true?*" we should ask "what is the *proof* of ϕ?" Writes Jean-Yves Girard in the introduction to **Proofs and Types**, one of the books we'll use,

> By *proof* we understand not the syntactic formal transcript, but the inherent object of which the written form gives only a shadowy reflection [like in Plato's cave]. We take the view that what we *write* as a proof is merely a description of something which is *already* a **process** in itself.

λ-calculus: The programming language we'll use as a *lingua franca* in this course is the λ-calculus, which is just a kind of mathematically idealized Scheme. (It's the blueprint

prototype from which functional programming languages were built.) We'll examine its *confluence* (the *Church-Rosser theorem*, that evaluation order doesn't really affect returned answers, like in "normal" mathematical calculi), the *standardization* of reductions (that anything you could compute by any evaluation order can also be computed by a canonical, "standard" reduction), and look at the *Krivine machine*, a λ-calculus interpreter that is call-by-name and computes *head reductions*. This machine will be important later because it is connected to the idea of *linearity*.

Sequent calculus: Next, we'll look at a logic, Gerhard Gentzen's *sequent calculus* for classical logic. This calculus is an abstraction of logical reasoning via formal proofs, and we'll prove Gentzen's *cut-elimination theorem*—that the so-called *cut rule*, which logically represents much of what we think of as a *procedure call*, can be *eliminated* from the calculus without changing theoremhood (what is a theorem). If a proof is a program, then cut-elimination is its intepreter. We observe, however, that classical cut-elmination is not *confluent* in the sense mentioned above—a bit like simplifying an arithmetic expression, and getting two different answers. Two ways of restoring confluence are by restricting our focus on so-called *intuitionistic logic*, which replaces the multiple conclusions of a classical logic proof with a single conclusion, or by passing to *linear logic*, which pays special attention to the logical rules of *contraction* and *weakening*.

Simply-typed λ-calculus: The simply-typed λ-calculus has a type system with variables, products, and functions. Following the Curry-Howard correspondence, λ-terms in this language represent proofs in a logic with propositional variables, conjunction, and implication; the normalization of terms corresponds to cut elimination on the analogous proofs. We will examine the complexity of deciding if two terms have the same normal form, and study two theorems demonstrating that all typed terms have a normal form. The first, *weak normalization theorem* identifies a specific reduction strategy which always yields a normal form; the second *strong normalization theorem* shows that any evaluation strategy must produce a normal form. The proof of the latter theorem is interesting in that it presents a general technique which can be generalized to more complex type systems.

Polymorphically typed λ-calculus (System F): This typed λ-calculus generalizes the simply-typed system by allowing quantification over type variables, which facilitates a certain *type polymorphism* in programming. The Curry-Howard correspondence is to a logic allowing universal quantification over propositional variables, or (equivalently) second-order quantification over a first-order universe.

This calculus allows a very straightforward *coding* of many familiar data types and algorithms for integers, lists, trees, and so on. We'll prove a *strong normalization* theorem for System F, using a clever variant of the technique used in the simply-typed case, Girard's *candidats de reductibilité*. (Via Curry-Howard, this also proves Gentzen's cut-elimination theorem for second-order logic.) We further show how second-order *proofs* that functional, equational specifications define *total functions* can then be mechanically analyzed, extracting (strongly normalizing) System F programs which realize the specifications. A corollary of this realizability, and the strong normalization theorem, is a Gödel-style incompleteness theorem that the strong normalization theorem cannot itself be proven in second-order logic, and (by the Curry-Howard correspondence) that a System F interpreter cannot be coded as a System F term. (Compare, in contrast, the metacircular evaluator for Scheme, coded itself in Scheme.)

Linear logic: We then pass to the second option for avoiding the nonconfluence of proofs in cut-elimination for classical logic, namely a *resource conscious logic*, called *linear logic*. Linear logic constricts the use of *contraction* (that two hypotheses A are the same as one) and *weakening* (that hypotheses can be added at discretion). To recover the effect of contraction and weakening, an *exponential modality* $!A$ is introduced which makes their use explicit in the logical formulas. We'll look at sequent calculus for linear logic, as well as a variant formulation called *proofnets*, examine the complexity of normalization and of parsing (the so-called *correctness criterion*), show how λ-calculus is coded in this logic (again, following Curry-Howard intuitions), and introduce a semantics of proofs, called the *geometry of interaction*, that describes paths in proofs which are crucial in cut-elimination.

Classical logic intuitionistically interpreted: A famous result of Gödel (also attributed to Kolmogorov, Kuroda, and others) shows how to compile classical sequent formulas into intuitionistic ones, such that theoremhood is preserved. This compilation is often called the *double-negation embedding*, as certain subformulas ϕ are replaced by $\neg\neg\phi$; note that while $A \vee \neg A$ is classically but not intuitionistically provable, $\neg\neg(A \vee \neg A)$ is intuitionistically provable. When the Curry-Howard perspective is used to understand this result, it turns out that the proofs produced by the compiler correspond to programs that have been converted using *continuation-passing style*. Further, the introduction of *control operators* (like `call/cc`) can be logically interpreted as the *elimination* of double-negation, i.e., $\neg\neg A \to A$.

Games: There is yet another logical tradition of understanding logic as a game between a *prover* and a *skeptic* who interact via a protocol regarding the provability of formulas; a formula is a theorem when there is a winning strategy for the prover. These games have intuitionistic and classical variants, with different protocols. A more recent rendition on these games is through the Curry-Howard correspondence, where the prover and skeptic are, effectively, a program and its contextual environment. Language features such as state and control operators can be explained by different game protocols. These games have been used to build models of programming languages, and address the issue of so-called *full abstraction.*

Differential λ-calculus: Finally, we'll conclude the course with a discussion of a differential λ-calculus. Derivatives are linear approximations of functions, and linear logic explains what linearity means in the context of computation. Then a *partial derivative* at a point can be modelled by a *linear substitution* of one term in another. This approach leads to a *chain rule* just as in the usual differential calculus, a *Taylor theorem* where λ-terms are approximated by infinite sums of iterated derivatives, and the terms in the sum can be understood as explaining computations made by the Krivine machine which we earlier introduced.

How hard will this course be?

The course will require no programming, but a certain mathematical maturity—it's not for the faint-hearted. Students should have completed (or have a strong grasp of the ideas in) CS21b (Structure and Interpretation of Computer Programs) and CS30a (Introduction to the Theory of Computation). A course in mathematical logic (PHIL 106b) or algebra (MATH 30a) would also be good preparation.

Grading and homework policy

The work for the course will consist of writing a 15-20 page expository paper on a subject from or connected to the course (in consultation with the instructor), and giving a lecture on one of the topics in the lecture list (again, in consultation with the instructor). Understanding

the readings will require *real mathematical maturity*, so this presentation is no easy task. Both the paper and the presentation will be evaluated on the basis of technical accuracy, but especially on the intuitions that motivate understanding of the subject. There may also be an occasional problem set. Class attendance is important.

Tentative syllabus

24 lectures overall. (Observe $H, D, M, ?$ mark lectures by Harry, David Van Horn, Matt Goldfield, or a class member.)

Introduction and survey

January 15:^{H} Introduction, administrivia, survey of course material.

λ-calculus (4 lectures)

January 18:^{H} λ-calculus basics, representation of data types and recursive functions; programming via iteration (on inductive data) and unbounded recursion, Church-Rosser theorem.

J. Roger Hindley and Jonathan P. Seldin, **Introduction to Combinators and Lambda Calculus**, Cambridge University Press, 1986, pp. 1–19 and 313–322.

January 22:^{H} Böhm's theorem.

Jean-Louis Krivine, **λ-calculus, Types and Models**, Ellis Horwood, 1993, pp. 67-72.

Class notes.

January 25:$^?$ Standardization theorem.

Hendrik Barendregt, **The Lambda Calculus**, North-Holland, 1984, pp. 296–301.

January 29:$^?$ Krivine machine and head reduction.

Jean-Louis Krivine, *A call-by-name lambda calculus machine,* **Higher-Order and Symbolic Computation** 20:3 (2007), pp. 199–207.

Mitchell Wand, *On the correctness of the Krivine machine,* **Higher-Order and Symbolic Computation** 20:3 (2007), pp. 231-235.

Sequent calculus and Gentzen's cut-elimination theorem (2 lectures)

February 1:^{H} Sequent calculus, cut-elimination theorem, non-confluence of classical logic.

Jean-Yves Girard, Yves Lafont, and Paul Taylor, **Proofs and Types,** Cambridge University Press, 1989, pp. 28–32, 104–112, and 149–151.

February 5:^{H} Hauptsatz in more detail, with bounds on normalization.

Jean-Yves Girard, **Proof Theory and Logical Complexity,** Bibliopolis, 1987, pp. 105–112.

Class notes.

Simply-typed λ-calculus (3 lectures)

February 8:[?] Terms, types, reduction. Curry-Howard correspondence, weak normalization theorem (ω^2 ordinal proof).

Morton Heine Sørensen and Pawel Urzyczyn, **Lectures on the Curry-Howard Isomorphism,** North-Holland, 2006, pp. 55-67 and 67–69.

Proofs and Types, pp. 14-21.

February 12:[?] Statman's theorem on decidability of equality of normal forms.

Harry Mairson, *A simple proof of a theorem of Statman,* **Theoretical Computer Science** 103:2 (1992), pp. 387–394.

February 15:[H] Strong normalization (Tait reducibility method).

Introduction to Combinators and Lambda Calculus, pp. 323–327.

Proofs and Types, pp. 22–27.

System F: λ-calculus with polymorphic types (4 lectures)

February 26:[M] Terms, types, reduction, representation of recursive functions.

Proofs and Types, pp. 81–93.

February 29:[?] Weak normalization.

Andre Scedrov, *Normalization revisited,* **Categories in Computer Science and Logic** (American Mathematical Society Contemporary Mathematics, vol. 92), pp. 357–370.

March 4:[H] Strong normalization: Girard's *candidats de reductibilité.*

Lectures on the Curry-Howard Isomorphism, pp. 287–290.

March 7:[D] Realizability of recursive functions (Curry-Howard), *Dialectica* interpretation, and a Gödel-style incompleteness.

Daniel Leivant, *Contracting proofs to programs*, **Logic and Computer Science** (ed. P. Odifreddi), Academic Press, 1990, pp. 279–328.

Linear logic (4 lectures)

March 11:[H] Introduction to linear logic.

Jean-Yves Girard, *Linear logic: its syntax and semantics*
Proceedings of the Workshop on Advances in Linear Logic, Cambridge University Press, 1995, pp. 1–42. *(Read first half—the syntax part.)*

March 14:[H] Proofnets for linear logic. Danos-Regnier correctness criterion for proofnets. Complexity of deciding correctness.

Yves Lafont, *From proofnets to interaction nets*, **Proceedings of the Workshop on Advances in Linear Logic**, Cambridge University Press, 1995, pp. 225–247.

March 18:^{H} Multiplicative linear logic and circuits, complexity of normalization.

Harry Mairson, *MLL normalization and transitive closure: circuits, complexity, and Euler tours,* **GEOCAL (Geometry of Calculation): Implicit Computational Complexity**, 2006. (Slides, paper copies not distributed.)

March 25:^{H} Decomposition of function space construction, coding of lambda calculus, Geometry of Interaction.

Harry Mairson. *From Hilbert space to Dilbert space: context semantics made simple* **Conference on Foundations of Software Technology and Theoretical Computer Science**, Kanpur, 2002. Lecture Notes in Computer Science 2556, pp. 2–17.

Classical logic intuitionistically interpreted, and control operators (2 lectures)

March 28:$^{?}$ Double-negation translations.

Dirk van Dalen, **Logic and Structure** (third edition), Springer, 1991, pp. 155–164.

April 1:^{D} Classical logic and control operators. $\lambda\mu$-calculus.

Lectures on the Curry-Howard Isomorphism, pp. 127–144.

Harry Mairson and David Van Horn.
Proofnets and paths in constructive classical logic: too old, too new. **GEOCAL (Geometry of Calculation): Geometry of Interaction**, 2006. (Slides, paper copies not distributed.)

Games (4 lectures)

April 4:$^{?}$ Intuitionistic and classical prover-skeptic dialogues.

Lectures on the Curry-Howard Isomorphism, pp. 89–96 and 144–150.

April 11:^{H} Lorenzen dialogues.

Lectures on the Curry-Howard Isomorphism, pp. 181–194.

April 15, 18:$^{?M}$ Game semantics for programming with computable functions.

Samson Abramsky and Guy McCusker, *Game semantics*, **Computational Logic** 165 (NATO Science Series, Series F: Computer and Systems Sciences), Springer 1999, pp. 1–55. *(Read sections 1 and 2.)*

Differential λ-calculus (1 lecture)

May 2:^{H} Differential λ-calculus, linear approximations, and Taylor's theorem reinterpreted.

Thomas Ehrhard and Laurent Regnier, *The Differential Lambda Calculus.* **Theoretical Computer Science** 309:1, pp. 1–41.

Reading

Samson Abramsky and Guy McCusker
Game semantics.
In **Computational Logic**, vol. 165, pp. 1–55.
NATO Science Series, Series F: Computer and Systems Sciences.
Springer-Verlag, Berlin, Germany, 1999.

Hendrik Barendregt
The Lambda Calculus.
North-Holland, 1984.

Dirk van Dalen
Logic and Structure. (Third edition)
Springer, 1991.

Thomas Ehrhard and Laurent Regnier
The Differential Lambda Calculus.
Theoretical Computer Science, vol. 309, no. 1, pp. 1–41.
Elsevier Science Publishers Ltd., Essex, UK, 2003.

Jean-Yves Girard
Proof Theory and Logical Complexity.
Bibliopolis, 1987.

Jean-Yves Girard, Yves Lafont, and Paul Taylor
Proofs and Types.
Cambridge University Press, 1989.
`http://www.monad.me.uk/stable/Proofs+Types.html`
`http://iml.univ-mrs.fr/~lafont/pub/prot.pdf.gz`

Jean-Yves Girard
Linear logic: its syntax and semantics.
Proceedings of the workshop on Advances in linear logic, pp. 1–42.
Cambridge University Press, 1995.

J. Roger Hindley and Jonathan P. Seldin
Introduction to Combinators and Lambda Calculus.
Cambridge University Press, New York, NY, 1986.

Jean-Louis Krivine
λ-calculus, Types and Models.
Ellis Horwood, 1993

Jean-Louis Krivine
A call-by-name lambda calculus machine.
Higher-Order and Symbolic Computation 20:3 (2007), pp. 199–207.

Yves Lafont
From proofnets to interaction nets.
Proceedings of the workshop on Advances in linear logic, pp. 225–247. Cambridge University Press, New York, NY, 1995.

Daniel Leivant
Contracting proofs to programs.
Logic and Computer Science (ed. P. Odifreddi), Academic Press, 1990, pp. 279–328.

Harry Mairson
From Hilbert space to Dilbert space: context semantics made simple.
Foundations of Software Technology and Theoretical Computer Science, Kanpur, 2002. Lecture Notes in Computer Science 2556, pp. 2–17.

Harry Mairson
A simple proof of a theorem of Statman.
Theoretical Computer Science 103:2 (1992), pp. 387–394.

Andre Scedrov
Normalization revisited.
Categories in Computer Science and Logic (American Mathematical Society Contemporary Mathematics, vol. 92), pp. 357–370.

Morton Heine Sørensen and Pawel Urzyczyn
Lectures on the Curry-Howard Isomorphism.
North-Holland, 2006.

Mitchell Wand
On the correctness of the Krivine machine.
Higher-Order and Symbolic Computation 20:3 (2007), pp. 231-235.