Week 2

David Baelde
Postdoctoral Fellow
IT University of Copenhagen
Denmark

Alwen Tiu
Research Fellow
Australian National University
Canberra
alwen.tiu@anu.edu.au

Title of the course: Specification and Prototyping Using Higher-Order Logic

Section: Logic and computation
Level: Introductory

Content of the course:

This course introduces a logical approach to formalizing, prototyping and reasoning about computational systems. We shall consider systems from various contexts, including typing and evaluation notions for programming languages, logics and proof-theoretic principles, formally specified software and concurrency. A feature common to many of these systems is that they are presented via recursive rule-based descriptions. Those rules usually involve syntactic structures featuring variable binding, a notion that has been surprisingly tricky to handle in specification and reasoning tasks. We will present a natural and effective way of formalizing such computational systems, based on relational specifications and a higher-order approach to representing syntax. Further, we will show how these specifications can be used for prototyping through logic programming, and for reasoning formally about represented systems through logics of definitions. Several tools will be used to develop various concrete applications of this methodology.

The lectures will be divided into roughly two parts. The first part will provide an exposure to the proof-theoretic principles of logic programming and to its higher-order extension, with a focus on helping the student understand the use of logic-based techniques for formalizing systems with complex binding structures. A particular emphasis in this part will be to provide a working knowledge of how such descriptions can be transformed into programs in the language λProlog. The second part will focus on logics that support the (meta-level) capability of reasoning about such specifications, and on computer-based systems that provide practical realizations of such logics. Two key logical notions are needed to reflect the intended meaning of our specifications: fixed-point definitions are used to reflect the inductive or coinductive nature of rule-based specifications, and generic quantification ∇ (pronounced as “nabla”) is used to formalize binding. At a practical level, we will show how an enhanced logic programming interpreter, with facilities to unfold fixed points, can be used to automate reasoning about the finite behavior of systems encoded in the logic; the Bedwyr prover will be used to demonstrate practical applications of this idea. We will then discuss a different approach to
the meta-reasoning task: instead of directly encoding a specification in the meta-logic, we will realize its encoding by embedding the entire specification logic itself in the meta-logic. This approach has the advantages of allowing specifications to be used also for prototyping, and leveraging properties of the specification logic when reasoning about formalized systems. The benefits of this “two-level logic” approach will be examined and the Abella theorem prover will be used to provide an in-depth exposure to its use through a collection of extended case studies.

At an intellectual level, this introductory course will both treat foundational issues and provide a hands-on experience with the practice of formalizing computation-oriented systems and reasoning about their meta-theory. There has been a recent resurgence of interest in such capabilities that are useful, for instance, in characterizing logical systems, in establishing properties of programming languages and in verifying software. The desired outcome of attending this course is that the student will have a sensitivity to the logical issues relevant to these areas and the broad preparation that is necessary to begin participating in research on these topics.

Tentative outline

- Lectures 1 and 2: **Structural Operational Semantics, Higher-Order Abstract Syntax, Logic Programming.** These two lectures will introduce the connections between SOS-style presentations of formal systems and relational styles of specification. Higher-order approaches to treating binding structure will be discussed and proof-theory based enhancements will be made to logic programming to support such treatments. Representation and specification concepts will be illustrated through actual examples such as the formalization of λ-calculi, the π-calculus and logics presented in natural deduction and sequent styles. Issues such as goal-directed search and unification relevant to animating specifications will be discussed. The λProlog language and its implementation in the Teyjus system will be exposed.

- Lecture 3: **Logics of Definitions and Reasoning about Finite Behavior.** This lecture will introduce fixed-point definitions and their use in encoding relational specifications, examine the nature of implication and universal quantification in this setting, and motivate the introduction of generic quantification to deal with binding structure. The resulting logic will be used to perform simple reasoning about the meta-theory of specified systems. In particular, we will present a simple extension to logic programming, justified by our new logical concepts, that can be used to reason about the finite behaviors of our systems. Examples will be provided using the Bedwyr prover.

- Lectures 4 and 5: **Reasoning about Infinite Behavior, Two Level Logic Approach, Case Studies.** The logic of definitions will be enhanced with treatments of least and greatest fixed points to realize inductive and co-inductive styles of reasoning, definitions will be extended to allow generic properties to be encoded in inductive contexts, the two-level logic approach to meta-theoretic reasoning about computations will be presented, proof-search issues will be discussed and implementation techniques will be considered. Several case studies relating to the meta-theory of programming languages, logics and the λ-calculus will be used to illustrate the approach. The Abella system will provide the means for presenting such examples.
References

The first two lectures of the course will draw on material presented in the book entitled *Programming with Higher-Order Logic* by Dale Miller and Gopalan Nadathur, scheduled for publication by Cambridge University Press in May 2012. A complete assimilation of the topics covered in the course will require experimenting with λProlog programs. This involves using the Teyjus system that is being distributed through the web. The URL for this system is http://teyjus.cs.umn.edu/.

The remaining three lectures will draw on material contained in the following collection of papers and theses:


A smaller, more focused list will be compiled from this collection about two months before the course. The instructors will also develop targetted lecture notes that will be made available to the registered course participants during the summer school. Finally, this part of the course will use the Bedwyr system which has a web page at

http://slimmer.gforge.inria.fr/bedwyr/

and the Abella system whose web page is at


Prerequisites

This course falls under the rubric of “introductory courses” in the ESSLLI roster.
We expect participants of this course to have had a previous exposure to logic. A background in logic programming may also be useful. Our presentation of the specification and computational capabilities deriving from higher-order logic will often be based on using logic programs in an extended language; someone who does not already understand logic programming may find it difficult to assimilate such programs at the expected pace. The course has logic content, all of which will be covered in the lectures. However, sensitivities to logical issues will be needed again to understand the concepts being treated sufficiently quickly.