CSC501: Programming Language Semantics – An Introduction to Formal Methods.

Syllabus – Fall 2014

Time: Section 1 MWF 9-9:50, Location: Tyler Hall Rm 052
Webpage: http://homepage.cs.uri.edu/faculty/hamel/courses/2014/fall2014/csc501
Prerequisites: CSC301

Instructor:
Prof. Lutz Hamel
email: hamel@cs.uri.edu
office: Tyler Hall

Course Description

As processors become faster and computers become more interconnected our software systems become more complex to take advantage of these additional computational resources. This trend is also reflected in our programming languages where new features evolve to deal with the distributed nature of our computational resources and to deal with the architecture of modern processors (e.g. multi-core processors). With this added complexity it is often difficult to establish that a program or programming language system behaves correctly. One way to approach the correctness of programs and language systems is to use *formal methods*. In this approach we construct a model of the software system or programming language system in a rigorous mathematical formalisms such as first-order logic and then *prove* that the model exhibits the intended (correct) behavior.

The application of formal methods applied to programming languages has two interesting points:

1) We can formally establish that a programming language implementation is correct.
2) We obtain a mathematically precise description of the behavior of every feature in the programming language under consideration.

The latter is interesting in its own right from a language design perspective; once we have a formal description of every feature of a programming language we can study how features interact in a rigorous way; we can study if a particular feature has the intended behavior under special circumstances. Furthermore, we can achieve this without having to go through the trouble of building an actual implementation of the language. Thus, besides being able to establish correctness of software systems, in the case of programming languages formal methods are also a good design tool.

Formal methods applied to programming languages come in many different flavors under many different names such as “axiomatic semantics”, “operational semantics”, “denotational semantics”, *etc.* each name indicating a different mathematical formalisms used to model a programming language. Collectively the approach of applying formal methods to programming languages is referred to as *programming language semantics*.

In this course we will use two flavors of first-order logic as our formal systems. The first approach uses first-order logic in conjunction with a deduction system called natural deduction and is referred to as *natural semantics*. The second approach uses a specialized version of first-order logic called Horn Clause Logic together with a deduction system called the resolution principle. The advantage of this latter approach is that the deductions in this logic are machine executable. Being able to execute deductions mechanically allows us to complete much more complex proofs than otherwise

possible since the machine will handle many of the proof details. This approach is often referred to a *executable operational semantics*.

The aim of this course is to familiarize you with the basic techniques of applying formal methods to programming languages. This includes constructing models for programming languages and using these models to prove properties such as correctness and equivalence of programs. We will look at all major programming language constructs including assignments, loops, type systems, and procedure calls together with their models. Since many of our models are executable, we can test and prove properties of non-trivial programs. Finally, we also look at compiler correctness.

Required Texts

None.

Software

The main software we will be using in this course is the Prolog programming environment. This is public domain software. More details on the course website.

Grading

Homework, Quizzes, and Programming Assignments	40%
Midterm	30%
Final	30%

Policies

- Check the website (often)! I will try to keep the website as up-to-date as possible.
- Class **attendance, promptness, participation,** and **adequate preparation** for each class are expected. If you are absent, it is your responsibility to find out what you missed (e.g. handouts, announcements, assignments, new material, etc.)
- **Late assignments** will **not** be accepted.
- **Make-up quizzes** and **exams** will **not** be given without a valid excuse, such as illness. If you are unable to attend a scheduled examination due to valid reasons, please inform myself, or the department office in Tyler Hall, prior to the exam time. Under such circumstances, you are not to discuss the exam with any other class member until after a make-up exam has been completed.
- All work is to be the result of your own individual efforts unless explicitly stated otherwise. **Plagiarism, unauthorized cooperation or any form of cheating** will be brought to the attention of the Dean for disciplinary action. See the appropriate sections (8.27) of the University Manual.
- **Software piracy** will be dealt with exactly like stealing of university or departmental property. Any abuse of computer or software equipment will subject to disciplinary action.

Tentative Schedule

Week 1

Mathematical foundations
Syntax and Semantics

Week 2

Term Rewriting, normal forms, Grammars
Syntactic derivations

Week 3

Building models of programming languages – Natural Semantics
Induction
- Proof Principles
- Inductive Definitions

Week 4

Intro to Prolog
Semantic definitions with Prolog – Executable Operational Semantics

Week 5

Machine executable proofs
Program equivalence

Week 6

Type Systems

Week 7

Arrays

Week 8

Functions and recursion

Week 9

Loops

Week 9

Loop termination

Week 10

Code translation

Week 11

Compiler correctness