CSC501: Programming Language Semantics

An Introduction to Formal Methods.

Syllabus – Spring 2022

Time: MoWeFr 11-11:50, Location: Davis 109
Webpage: https://brightspace.uri.edu/d2l/home/174251 and https://lutzhamel.github.io/CSC501
Prerequisites: CSC301

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

Course Description

Ever wondered what a **mathematical foundation of programming languages** might look like? Ever wondered what a mathematical foundation of programming languages might accomplish?

It is often difficult to establish that a program behaves correctly under all possible circumstances. One way to approach the correctness of programs is to use *formal methods*. In this approach we construct a model of the program in a rigorous mathematical formalism 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 obtain a **mathematical description of the behavior of every feature** in the programming language under consideration.
2) We can formally **establish that a program behaves correctly,** that is, it behaves according to a given specification under all circumstances.

From a language design perspective, the former is interesting in its own right; 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. The latter is the holy grail of programming in that **we can prove that our programs are correct under all circumstances.**

In this course we will use two flavors of first-order logic as our formal systems for specifying semantics. The first approach uses first-order logic in conjunction with a deduction system called natural deduction and is usually 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 as embodied by Prolog**. We will be using Prolog to automate our proofs.** 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.

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 use 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 our models are executable, we can test and prove properties of non-trivial programs. We also explore the topic of compiler correctness.

The expectation is that you have a good background in discrete mathematics and are familiar with programming languages and programming. You should be able to write a program and execute it in your favorite language The course is open to undergraduates and graduates from all mathematical sciences and engineering disciplines.

Required Texts

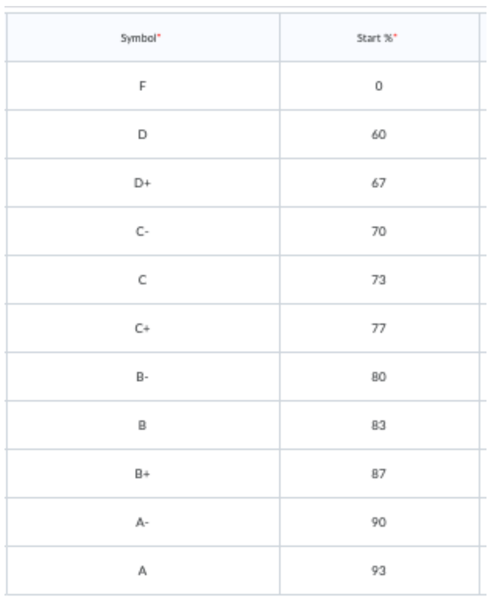
None.

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

Assignments	40%
Midterm	30%
Final	30%



Symbol*	Start %*
F	0
D	60
D+	67
C-	70
C	73
C+	77
B-	80
B	83
B+	87
A-	90
A	93

Policies

- Check BrightSpace!
- **Email Policy:** Emails sent to me on **Monday through Friday between 8am and 5pm** will usually be answered within an hour or two. For emails sent to me outside of these time you can expect an answer on the next business day.
- Class **attendance** is mandatory and will be checked.
- Class **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 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 other form of cheating will be reported to the Dean**. See the appropriate sections (8.27) of the University Manual.
- Any student with a documented disability is welcome to contact me as early in the semester as possible so that we may arrange reasonable accommodations. As part of this process, please be in touch with Disability Services for Students Office at 302 Memorial Union, Phone 401-874-2098.

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