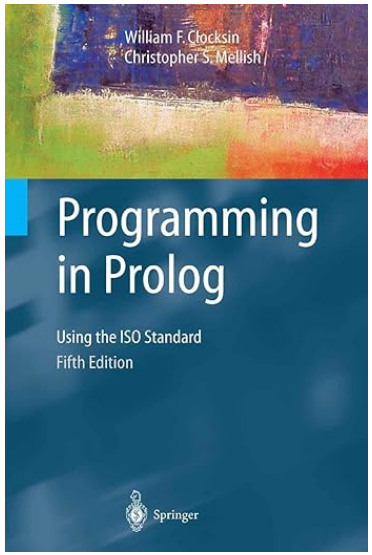
CSC 501 – Semantics of Programming Languages

- Subtitle: An Introduction to Formal Methods.
- Instructor: Dr. Lutz Hamel
- Email: lutzhamel@uri.edu
- Office: Tyler, Rm 251

Books



For the majority of the course we will rely on Prolog skills. We will use “Programming in Prolog: Using The Iso Standard”, 5th Edition, by Clocksin and Mellish.

Course Objectives

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.
- Look at all major programming language constructs including assignments, loops, type systems, and function calls together with their models.
- Introduce mechanical theorem proving assistants so that we can test and prove properties of non-trivial programs.

Semantics

Programming Language Semantics:

- Programming language semantics refers to the rules and principles that define the **meaning and behavior** of a programming language.
- It provides a formal specification of how programs written in a particular programming language should be executed or interpreted.
- Semantics help programmers and programming language implementers understand how code should behave, allowing for correct and predictable program execution.

Formal Methods

- **Formal methods** refer to a set of mathematical techniques and tools used in computer science and software engineering to specify, design, and verify software and hardware systems.
- These methods employ rigorous and precise mathematical models to describe system behavior and properties, aiming to improve the reliability, correctness, and safety of complex systems.
- Formal methods are particularly valuable in critical systems where errors can have severe consequences, such as in aerospace, healthcare, and automotive industries.

Formal Methods

Key components of formal methods include:

- **Formal Specification:** This involves the use of **formal systems** to precisely define the requirements and behavior of a system. Formal specifications eliminate ambiguity and provide a clear, unambiguous representation of system functionality.
- **Modeling:** Formal methods often use mathematical models, such as finite state machines, logic, and set theory, to represent the behavior and structure of a system. Models enable the analysis of system properties, such as correctness and safety.
- **Verification:** Formal methods allow for the rigorous verification of system properties and requirements. This can include formal proofs of correctness, model checking, and theorem proving to ensure that a system adheres to its specified behavior and constraints.

Formal Systems

- A **formal system** consists of a *formal language* and a set of *inference rules*. The formal language is composed of primitive symbols that make up well formed formulas and the inference rules are used to derive expressions from other expressions within the formal system.
- These systems are employed in various fields, including mathematics, computer science, logic, linguistics, and philosophy, to provide a precise and rigorous means of reasoning and constructing proofs.
- In order to be truly useful in computer science, we require our formal systems to be *machine executable*.

Observations

When programming we can observe two mental activities:

- We construct *correct looking* programs - *syntactically* correct programs.
- We construct *models* of the intended computation in our minds. Consider,

```
x := 1
while (x <= 10) do
     writeln(x)
     x := x + 1
end whiledo
```

Any person with some familiarity of programming immediately has a mental picture that this program will generate a list of integers from 1 through 10.

Programming Language Definitions

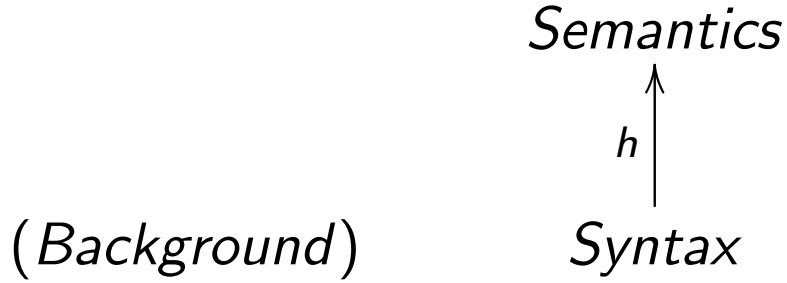
Mirroring our intuition, language definitions consist of two parts:

- **Syntax:** The formal description of the **structure** of well-formed expressions, phrases, programs, etc.
- **Semantics:** The formal description of the **meaning** of the syntactic features of a programming language usually understood in terms of the runtime **behavior** each syntactic construct evokes. The formal description of the behavior of all the syntactic features of a language is considered a **model** of the language.

Evaluation/Interpretation

Syntax and semantics of a programming language are usually related via an *evaluation relation* or *interpretation*, say h. Then we say that the interpretation h takes each syntactic element and maps it into the appropriate semantic construct.

We often represent this with the diagram



Note: In order for the interpretation h to make any sense we will have to define the syntax and semantics in terms of sets.
Note: Syntax is often evaluated in the context of some background like machine state.

Formal Systems and Programs

The formal systems we will be using in this course are:

- First-order logic extended with **natural deduction** – natural semantics.
- Horn Clause Logic as implemented by Prolog, a subset of first-order logic.

Readings

- Read Chapter 0 in "Denotational Semantics" by David Schmidt (available from the course website).
- Read Sections 2.1 and 2.2 in "Denotational Semantics" by David Schmidt.