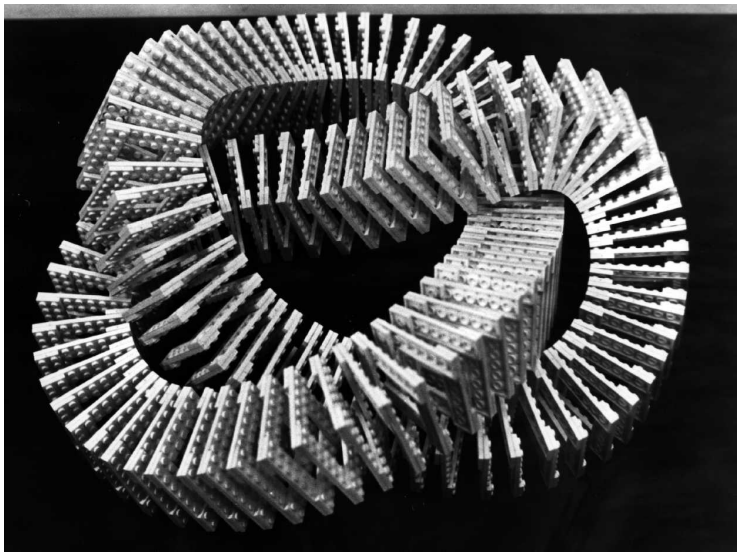
CSC 501 – Semantics of Programming Languages

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



Books

There are no required books in this course; however, occasionally I will assign readings based on material available on the web.

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 procedure calls together with their models.
- Introduce mechanical theorem provers so that we can test and prove properties of non-trivial programs.

Some Definitions

Definition: In **programming language semantics** we are concerned with the *rigorous mathematical study* of the *meaning* of programming languages. The meaning of a language is given by a *formal system* that describes the possible computations expressible within that language.

Some Definitions

Definition: In computer science and software engineering, **formal methods** are techniques for the specification, development and verification of software and hardware systems based on *formal systems*.

Formal Systems

Definition: 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. A formal system may be formulated and studied for its intrinsic properties, or it may be intended as a description (i.e. a model) of external phenomena.[1]

In order to be truly useful in computer science, we require our formal systems to be *machine executable*.

[1] Wikipedia

Uses of Formal Methods

Implementation Issues Formally specified models can be considered machine-independent specifications of program behavior. They can act as "yard sticks" for the correctness of program implementations, transformations, and optimizations.

Verification Basis of methods for reasoning about program properties (e.g. equivalence) and program specifications (program correctness).

Language Design Can bring to light ambiguities and unforeseen subtleties in programming language constructs.

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:

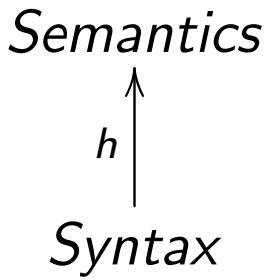
Syntax The formal description of the **structure** of well-formed expressions, phrases, programs, etc.

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.

Formal Systems and Programs

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

- A variant of *string rewriting systems* called a *grammar* to model the syntax of programming languages.
- The *first order predicate calculus* (often also called first order logic) to construct semantics of programming languages.

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.

Mathematical Preliminaries: Logical Notation

We[2] will use first order logic as the basis for our reasoning. Without going into the formal details of first order logic terminology and sentence construction we have the following statements:

- $A \wedge B$ denotes the conjunction A and B,
- $A \vee B$ denotes the disjunction A or B,
- $\neg A$ denotes the negation not A,
- $A \Rightarrow B$ denotes the implication, if A then B,
- $A \Leftrightarrow B$ denotes the logical equivalence, A if and only if B (often written as A iff B),

where A and B are statements or assertions.

[2] The material presented here is based on "Naive Set Theory" by P. Halmos and "The Formal Semantics of Programming Languages" by G. Winskel.

Observe the precedences of the logical operators, ordered from high to low:

- $\neg$
- $\wedge, \vee$
- $\Rightarrow, \Leftrightarrow$

A Word or Two about Implication

The truth table for the implication operator '$\Rightarrow$' can be given as

	A	B	$A \Rightarrow B$
(1)	1	0	0
(2)	1	1	1
(3)	0	0	1
(4)	0	1	1

Entries (1) and (2) are intuitive: When the antecedent A is true but the consequent B is false then the implication itself is false. If both the antecedent and the consequent are true then the implication is true.

However, entries (3) and (4) are somewhat counter intuitive. They state that if the antecedent A is false then the implication is true regardless of the value of the consequent. In other words, we can conclude "anything" from an antecedent that is false. In mathematical jargon we say that (3) and (4) **hold trivially**.

A Word or Two about Implication – An Example

If Bob is a bachelor, then he is single.
Bob is a bachelor.

$\therefore$ Bob is single.

Now consider an antecedent that is not true,

If Bob is a bachelor, then he is single.
Bob is not a bachelor.

$\therefore$ Bob is not single (by rule (3)).

Since the antecedent is not true rule (3) allows us to conclude the opposite of what the implication dictates. However, the following is also valid reasoning,

If Bob is a bachelor, then he is single.
Bob is not a bachelor.

$\therefore$ Bob is single (by rule (4)).

Not being a bachelor does not necessarily imply that Bob is not single. For example, Bob could be a widower or a divorcee.

A Word or Two about Implication

Given the truth table for implication,

	A	B	$A \Rightarrow B$
(1)	1	0	0
(2)	1	1	1
(3)	0	0	1
(4)	0	1	1

this means that in order to show that an implication holds we only have to show that rule (2) holds. Rule (1) states that the implication is false and rules (3) and (4) are trivially true and therefore not interesting.

Closely Related: Equivalence

We write $A \Leftrightarrow B$ if A and B are equivalent.
Given the truth table for the equivalence operator is given as,

	A	B	$A \Leftrightarrow B$
(1)	1	0	0
(2)	1	1	1
(3)	0	0	1
(4)	0	1	0

That is, the operator only produces a true value if A and B have the same truth assignment.
Another, and very useful, way to look at the equivalence operator is as follows:

$$A \Leftrightarrow B \equiv A \Rightarrow B \wedge B \Rightarrow A$$

Exercise: Construct the above truth table using this definition of the equivalence operator.

We also allow predicates (properties) as part of our notation,

$$P(x)$$

where the predicate P is true if it holds for x otherwise it is false. We view our standard relational operators as binary predicates. For example, the predicate $P(x)$ that expresses the fact that x is less or equal to 3 is written as,

$$P(x) \equiv x \leq 3.$$

Note: Predicates can have arities larger than 1, e.g. $P(x, y)$ with $P(x, y) \equiv x \leq y$.

Mathematical Preliminaries: Logical Notation

We also allow for the quantifiers $\exists$ (there exists) and $\forall$ (for all) in our logical statements,

- $\exists x.\, P(x)$ – "there exists an x such that $P(x)$"
- $\forall x.\, P(x)$ – "for all x such that $P(x)$"

Some examples,

- $\forall x, \exists y.\, y = x^2$
- $\forall x, \forall y.\, \text{female}(x) \wedge \text{child}(x, y) \Rightarrow \text{mother}(x, y)$

Mathematical Preliminaries: Sets

Sets[3] are unordered collections of objects and are usually denoted by capital letters. For example, let a, b, c denote some objects then the set A of these objects is written as,

$$A = \{a, b, c\}.$$

There are a number of standard sets which come in handy,

- $\emptyset$ denotes the empty set, i. e. $\emptyset = \{\}$,
- $\mathbb{N}$ denotes the set of all natural numbers including 0, e. g. $\mathbb{N} = \{0, 1, 2, 3, \cdots\}$,
- $\mathbb{I}$ denotes the set of all integers, $\mathbb{I} = \{\cdots, -2, -1, 0, 1, 2, \cdots\}$,
- $\mathbb{R}$ denotes the set of all reals,
- $\mathbb{B}$ denotes the set of boolean values, $\mathbb{B} = \{\mathit{true}, \mathit{false}\}$.

[3] Read Sections 2.1 and 2.2 in the book by David Schmidt.

Mathematical Preliminaries: Sets

The most fundamental property in set theory is the notion of *belonging*,

$$a \in A \text{ iff } a \text{ is an element of the set } A.$$

The notion of belonging allows us to define *subsets*,

$$Z \subseteq A \text{ iff } \forall e \in Z.\, e \in A.$$

We define set *equivalence* as,

$$A = B \text{ iff } A \subseteq B \wedge B \subseteq A$$

We can construct new sets from given sets using *union*,

$$A \cup B = \{e \mid e \in A \vee e \in B\},$$

and *intersection*,

$$A \cap B = \{e \mid e \in A \wedge e \in B\}.$$

There is another important set construction called the *cross product*,

$$A \times B = \{(a, b) \mid a \in A \wedge b \in B\},$$

$A \times B$ is the set of all ordered pairs where the first component of the pair is drawn from the set A and the second component of the pair is drawn from B. (

Exercise: Let $A = \{a, b\}$ and $B = \{c, d\}$, construct the set $A \times B$.

A construction using subsets is the *powerset* of some set X,

$$\mathcal{P}(X),$$

The *powerset* of set X is set of all subsets of X. For example, let $X = \{a, b\}$, then

$$\mathcal{P}(X) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}.$$

Note: $\emptyset \subset X$

Exercise: What would $\mathcal{P}(X \times X)$ look like?

The fact that $\emptyset \subset X$ for any set X is interesting in its own right. Let's see if we can prove it.

Proof: Proof by contradiction. Assume X is any set. Assume that $\emptyset$ is not a subset of X. Then the definition of subsets,

$$A \subseteq B \Leftrightarrow \forall e \in A . e \in B,$$

implies that there exist at least one element in $\emptyset$ that is not also in X. But that is not possible because $\emptyset$ has no elements – a contradiction. Therefore, our assumption the that $\emptyset$ is not a subset of X must be wrong and we can conclude that $\emptyset \subset X$.

Mathematical Preliminaries: Relations

A (binary) relation is a set of ordered pairs. If R is a relation that relates the elements of set A to the elements B, then

$$R \subseteq A \times B.$$

This means if $a \in A$ is related to $b \in B$ via the relation R, then $(a, b) \in R$. We often write

$$a \, R \, b.$$

Consider the relational operator $\leq$ applied to the set $\mathbb{N} \times \mathbb{N}$. This induces a relation, call it $\leq \subseteq \mathbb{N} \times \mathbb{N}$, with $(a, b) \in \leq$ (or $a \leq b$ in our relational notation) if $a \in \mathbb{N}$ is less or equal to $b \in \mathbb{N}$.

The first and second components of each pair in some relation R are drawn from different sets called the *projections* of R onto the first and second *coordinate*, respectively. We introduce the operators *domain* and *range* to accomplish these projections. Let $R \subseteq A \times B$, then,

$$\text{dom}(R) = A,$$

and

$$\text{ran}(R) = B.$$

In this case we talk about a relation *from* A *to* B. The range is often called the co-domain. If $R \subseteq X \times X$, then

$$\text{dom}(R) = \text{ran}(R) = X.$$

Here we talk about a relation *in* X.

Mathematical Preliminaries: Relations

Let $R \subseteq X \times X$ such that $(a, b) \in R$ iff $a = b$. That is, R is the *equality relation* in X. (What do the elements of the equality relation look like for $\mathbb{N} \times \mathbb{N}$?)

A relation $R \subseteq X \times X$ is an *equivalence relation* if the following conditions hold,

- R is *reflexive*[4] – $x\,R\,x$,
- R is *symmetric* – $x\,R\,y \Rightarrow y\,R\,x$,
- R is *transitive* – $x\,R\,y \wedge y\,R\,z \Rightarrow x\,R\,z$,

where $x, y, z \in X$.

The *smallest* equivalence relation in some set X is the equality relation defined above. The *largest* equivalence relation is some set is the cross product $X \times X$. (Consider the smallest/largest equiv. relation in $\mathbb{I}$)

[4] Recall that $x\,R\,x \equiv (x, x) \in R$

A *function* f from X to Y is a relation $f \subseteq X \times Y$ such that

$$\forall x \in X, \exists y, z \in Y. (x, y) \in f \wedge (x, z) \in f \Rightarrow y = z.$$

In other words, each $x \in X$ has a unique value $y \in Y$ with $(x, y) \in f$ or functions are constrained relations.

We let $X \to Y$ denote the *set of all functions* from X to Y (i. e. $X \to Y \subset \mathcal{P}(X \times Y)$, why is the subset strict? Hint: it is not a relation), then the customary notation for specifying functions can be defined as follows,

$$f : X \to Y \text{ iff } f \in X \to Y.$$

For *function application* it is customary to write

$$f(x) = y$$

for $(x, y) \in f$. In this case we say that the function is *defined* at point x. Otherwise we say that the function is *undefined* at point x and we write $f(x) = \perp$.

Note that $f(\perp) = \perp$ and we say the f is *strict*.

We say that $f : X \to Y$ is a *total* function if f is defined for all $x \in X$. Otherwise we say that f is a *partial* function.

Mathematical Preliminaries: Functions

We can now make the notion of a predicate formal – a predicate is a function whose range (co-domain) is restricted to the boolean values:

$$P : X \to \mathbb{B}$$

where P is a predicate that returns true or false for the objects in set X.

Example: Let U be the set of all possible objects – a universe if you like, and let,

$$human : U \to \mathbb{B}$$

be the predicate that returns true if the object is a human and will return false otherwise, then

$$human(\text{socrates}) = true$$
$$human(\text{car}) = false$$

Mathematical Preliminaries: Exercises

1. In your own words explain what the function $m : X \times Y \to Z$ does.
2. How would you describe the function $c : X \to (Y \to Z)$?
3. In your own words explain what the relation $R \subseteq (X \times Y) \times (Z \times W)$ does.