

Welcome to CS 245

Instructor: Brad Lushman

Web page:

`http://www.student.cs.uwaterloo.ca/~cs245`

Read everything on the Web page carefully as soon as possible,
especially the academic offenses page.

Newsgroup: `uw.cs.cs245`

Read the newsgroup at least daily for fast-breaking items and
discussions of common interest.

Course logistics

Assignments (six to eight in total) will be worth roughly 25%, the midterm 25%, and the final exam 50%.

One or more graduate student TAs will be delivering weekly tutorials, with some reinforcement of lecture material and some practice in solving problems.

The required textbook is “Logic in computer science: modelling and reasoning about systems” (second edition), by Michael Huth and Mark Ryan.

The bookstore currently has copies of a book by Nissanke on its shelves. **We will not be using this text.** The Huth and Ryan book has been ordered.

We will use Chapters 1, 2, and 4 (out of 6). If time permits, we will look at some of the other relevant topics in the book.

Overview of course

This course is about formal mathematical logic and its application to computer science.

In your previous CS and math courses, you have been exposed to some aspects of logic, in varying degrees of formality.

Math 135 introduced you to varying styles of mathematical proof using the topic of elementary number theory.

But what is logic?

A timeless question...

UW Motto: *Concordia cum veritate*

WLU Motto: *Veritas omnia vincit*

Quid est veritas?

Quid est veritas — What is truth?

What is “truth”?

More concretely, let S be a statement. What do we mean when we say that S is “true”?

One possible answer

S is true if S can be obtained from some established “basic truths” by some established “truth derivation procedure”.

This is “syntactic truth” or derivability.

Example

- for any x , $x = x$ is a basic truth
- for any x, y, z , if $x = y$ is true, then $z + x = y + z$ is true

Then “john = john” is true and “mary + john = john + mary” is true,
and “john + mary = john + mary” is true,
but “john + mary + ted = mary + john + ted” is not true.

Clearly, this setup does not capture all that we believe to be “true”,
but maybe with the right set of basic truths, and the right derivation
procedure....

Another answer

Every statement S is either an atomic statement (which may or may not be a basic truth) or a pair of statements S_1 and S_2 , joined by some connective \square .

Let \mathbb{B} be the set $\{T, F\}$. To each connective \square associate a function $\text{meaning}(\square) : \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$.

An *interpretation* is a function ϕ that maps each atomic statement to either T or F , and such that

$$\Phi(S_1 \square S_2) = (\text{meaning}(\square))(\phi(S_1), \phi(S_2)) .$$

Then S is true if every interpretation that maps all basic truths to T also maps S to T . This is “semantic truth” or entailment.

Example

Let \Box be a connective, with $\text{meaning}(\Box)$ defined as follows:

$$(\text{meaning}(\Box))(x, y) = \begin{cases} T & \text{if } x = y \\ F & \text{otherwise} \end{cases} .$$

Then $z\Box z$ is true, because for every interpretation ϕ , either $\Phi(z) = T$ or $\Phi(z) = F$. In the first case, we get

$$\Phi(z\Box z) = (\text{meaning}(\Box))(\Phi(z), \Phi(z)) = (\text{meaning}(\Box))(T, T) = T .$$

In the second case, we get

$$\Phi(z\Box z) = (\text{meaning}(\Box))(\Phi(z), \Phi(z)) = (\text{meaning}(\Box))(F, F) = T .$$

Since every interpretation Φ maps $(z\Box z)$ to T , $z\Box z$ must be true.

Two notions of truth

- syntactic: S is true if we can prove it from basic truths (“axioms”) A_1, \dots, A_n .
- semantic: S is true if all interpretations of S that yield truth (i.e., T) for all basic truths A_1, \dots, A_n , must also yield truth for S .

In the first case, we write $A_1, \dots, A_n \vdash S$.

In the second case, we write $A_1, \dots, A_n \models S$.

Two notions of truth cont'd

The turnstile (\vdash) encodes the “rules of the game”, i.e., what manipulations of the elements of statements constitute valid deductions.

The double turnstile (\models) encodes “truth by lack of counterexample”—since there is no way to interpret S as false without also falsifying an axiom, S must be true. Put another way, S is true in all universes, or in all experiences.

What is (mathematical) logic, then?

Do \vdash and \models mean the same thing?

Are all things we experience as true provable as such?

Is everything we can prove actually true?

These are the central questions of logic.

Logic is all about various ways of formulating \vdash and \models , and the relationships between them.

Two sample deductions

All courses taught by Lushman are hard.

Lushman is teaching CS245.

Therefore, CS245 is hard.

No one has ever passed an exam set by Lushman.

Lushman will be setting your CS245 exam.

Therefore, you will not pass your CS245 exam.

(Note: where you see “X, Y, therefore Z,” think “ $X, Y \vdash Z$ ”.)

Are these valid deductions? Why or why not? How do you decide?

A “Math 135-style” example

Thm: Every even natural number is the sum of two odd natural numbers whose difference is at most 2.

Proof: An even natural number is of the form $2k$, for $k \geq 1$. If k is odd, then the number can be expressed as $k + k$. If k is even, then the number can be expressed as $(k - 1) + (k + 1)$. \square

This is an example of what we will call a mathematical proof.

But in this course, we will eye such proofs with suspicion. How do we *know* that this argument is valid?

What are the valid rules of deduction? Does this proof follow them?

Our goal is to formalize the notions of proof and truth, and to explore the implications of such formalizations.

Note that in this example, we have used the Math 135 definition of the natural numbers which begins at 1, excluding 0. The textbook for this course also implicitly assumes this, in its discussion of induction.

However, when creating a precise and formal definition of the natural numbers (as we will do later in this course), it is more convenient and useful to include 0 as a natural number.

As a compromise, we will use the Math 135 definition in mathematical discussion, and the CS-style definition in formalization.

Provability vs. non-refutation

(\vdash vs. \models again)

We showed a short mathematical proof of “Every even natural number is the sum of two odd natural numbers whose difference is at most 2”.

This statement is both provable and “true” in the sense that there are no counterexamples (i.e., irrefutable, though ironically this claim also demands proof).

“Every odd natural number is the sum of two even natural numbers.”

This statement is false. Any odd number suffices as a counterexample, as the sum of two even numbers is always even. Luckily, it is also not provable (another fact which demands proof!).

\vdash vs. \models continued

What about this statement:

“Every even natural number greater than 2 is the sum of two prime numbers”.

We believe that it is either false or true—either there is a counterexample or there isn’t one.

But we currently don’t know which it is. No counterexample has been found, but no proof has been found either!

If the statement is false, then there will be a counterexample somewhere, and eventually we will find it. Conversely, if the statement is true, we will not find a counterexample.

Assuming the rules of mathematics are “well-behaved”, if the statement is false, we will not find a proof. But if it is true, are we guaranteed to find a proof?

In principle, we can enumerate all possible proofs until we find one for this statement. But does such a proof even exist??

An answer to a question like this would be a “proof about proofs” – a “meta-proof”.

How, then, do we know that our meta-reasoning always yield truth, or that the meta-proof we seek even exists?

Proofs of such things are “proofs about meta-proofs” or “meta-meta-proofs”.

Anyone’s head hurting yet?

Reasoning on multiple levels

Can a formal proof system reason about itself?

Can a system prove its own consistency?

If not, then for any logic, we can only prove results about the theorems it contains by “stepping outside the system” (or “up a level”) and working in a larger proof system.

But then proving anything (e.g. consistency) about the larger system requires stepping outside again and working on an even higher level.

And so on....

What is the highest level? Mathematics? English?

Can English prove its own consistency? Is it even consistent?

Reasoning on multiple levels cont'd.

We may be able to construct as many levels of proof rules as we like over the core system, but ultimately, at the highest level, we still may not be able to prove basic consistency (or other) results that we “know” to be true.

At some point, then, we must stop trying to prove these modes of reasoning that we “know” are right, and simply believe them.

But for the lowest couple of levels, such reasoning may be profitable, and we will certainly study it.

What about the “Computation” part?

What is the role of computation in our study of logic?

Computers are really good at syntactic manipulation. We can use computers, therefore, to implement proof-checkers and other tools to assist us in proving theorems (catching mistakes, proving some simple results automatically).

Conversely, we will see how to use logic to verify that a computer program computes correct results.

Again we have a nasty cycle – can you use logic to verify a proof-checker?

Finally, there is a fundamental and deep connection between types in certain programming languages and theorems in a particular

logic. The same connection links programs with proofs, so that a theorem has a proof if and only if there is a program with the corresponding type. Unfortunately, we will not be able to develop the necessary machinery to explore this idea in depth.

Two levels of reasoning—syntax and semantics

The ideas of syntax and semantics are central to computer science.

A computer program is just a sequence of symbols. There are **syntactic** rules that determine whether a program is well-formed or not (e.g. parentheses must balance).

The **semantics** of a program (its meaning or interpretation) describe what happens when that program is run. Usually some additional context is needed (e.g. the data on which the program is to be run). This is also true for some things which are not programs (e.g. Web pages in HTML, database queries in SQL).

The syntactic level—logic as a formal language

We will define a logical statement as a sequence of symbols that obey certain syntactic rules (specified, for example, by a context-free grammar).

A proof of a logical statement will be defined as a sequence of syntactic productions of logical statements, resulting in the statement we wish to prove.

Important—all we do at the syntactic level is manipulate symbols. Any intended meaning behind those symbols is irrelevant to us.

We will thus define a formal notion of “proof” without any attached semantics or meaning. It will just involve manipulation of symbols.

The semantic level—but what do these statements actually mean?

Independently, we will define semantics for logical statements, which is a way of interpreting them to determine if they are true or false. Thus we will have a notion of “truth” that is separate from the notion of “proof”.

Our goal will be to show that these two notions coincide: that every statement we can prove is true, and vice-versa. This justifies the utility of proofs and the benefits of looking for them.

We will start with a small system (incapable of expressing the theorems we looked at) but one which is familiar to you from Math 135. Once we show that provable equals true for this system, we will expand it.

The style of proof we use at the semantic level is less formal than what we insist upon at the syntactic level. At the semantic level, we will be working in the style of mathematical proof you're used to, even as we attempt to formalize it in stages.

Ideally, we would like to reach the point where proofs about aspects of our formal systems can be expressed in those systems themselves. This is unfortunately not always possible, and we will briefly examine the reasons.

Our goal, however, will be to formalize enough of mathematics to be able to apply the formalisms of logic to proofs of program correctness.

This course extends ideas you may have seen briefly in Math 135, 136, 137, 138, CS 134, 135, 136, 240, and 251.

It will be relevant to ideas you will see in CS 240 and 251 (if you have not taken them), CS 341, 360, 365, 442, 445, 446, 447 (and the SE equivalents), 462, 466, 480, 486, and just about any CS course where you have to either write programs or reason about algorithms.

Some of these courses may not directly depend on CS 245, but the exposure to concepts and practice in techniques here will strengthen your ability to deal with these courses. It will also help in just about any other Math Faculty course you take. If you really like the notions presented here, you might consider taking PMath 432, which goes further than we can (in ways to be alluded to later on in the course).