Prologue: what is this course about, and why should I care?

In this course, we will be talking about how to express statements precisely using the language of mathematical logic. This gives a way to communicate ideas without any ambiguity, which is an essential skill for any discipline. For example, the English statement "Some people like David" can be interpreted as saying that at least one person likes David, or that few, many, or even all people like David. What about "You can get cake or ice cream"? Does this mean that you may enjoy both cake and ice cream, or that you must choose between the two? Another example is the English expression "If you are a Pittsburgh Pens fan, then you are not a Philadelphia Flyers fan." Its meaning is clear enough if you meet a Pens fan, but what does this mean, if anything, for someone who *isn't* a Pittsburgh Pens fan? Does the same reasoning apply to the statement "If you can solve any problem in this course, then you will get an A"? Mathematical expressions in formal logic, on the other hand, have only one meaning. They remove all ambiguity so that only one interpretation is possible.

The second major theme of the course is developing methods to give rigorous mathematical proofs or disproofs of mathematical statements. We don't just want to be able to express ideas, we want to be able to argue – to both ourselves and others – that these ideas are correct. Mathematical proofs are a way to convince someone of something in an absolute sense, without worrying about biases, rhetoric, feelings, or alternate interpretations. The beauty of mathematics is that unlike other vast areas of human knowledge, it is possible to prove that a mathematical statement is true with one-hundred percent certainty. Without a rigorous mathematical proof, we can be easily fooled by our intuition and preconceptions. We will see throughout the course that some statements that seem perfectly reasonable turn out to be wrong, and others turn out to be true in surprising ways. Sometimes our intuition is valid and a proof seems like a mere formality; but often our intuition is incorrect, and going through the process of a rigorous mathematic proof is the *only* way that we discover the truth!

Why do we need mathematical expression and reasoning in computer science?

Course overview

Mathematical Expression

As a starting point for formalizing our intuition of logic, we will define two mathematical notions that we will use repeatedly throughout the course: sets and functions. Much of the terminology here may be review for you (or at least appear vaguely familiar), but please pay careful attention to the bolded terms, as we will make heavy use of each of them throughout the course. As we will stress again and again, *definitions* are precise statements about the meaning of a term or symbol; whenever we define something, it will be your responsibility to understand that definition so that you can understand (and later, reason about) statements using these terms at any point in the future.

Sets

Functions

Summation and product notation

Inequalities

Propositional logic

Predicate logic

Introduction to Proofs

In the previous chapter, we studied how to express statements precisely using the language of predicate logic. But just as English enables to make both true and false claims, the language of predicate logic allows for the expression of both true and false sentences. In this chapter, we will turn our attention to analyzing and communicating the truth (or falsehood) of these statements. You will develop the skills required to answer the following questions:

- How can you figure out if a given statement is true or false?
- If you know a statement is true, how can you convince others that it is true? How can you do the same if you

know the statement is false instead?

- If someone gives you an explanation of why a statement is true, how do you know whether to believe them or not?

These questions draw a distinction between the internal and external components of mathematical reasoning. When given a new statement, you'll first need to figure out for yourself whether it is true (internal), and then be able to express your thought process to others (external). But even though we make a separation, these two processes are certainly connected: it is only after convincing yourself that a statement is true that you should then try to convince others. And often in the process of formalizing your intuition for others, you notice an error or gap in your reasoning that causes you to revisit your intuition – or make you question whether the statement is actually true!

A **mathematical proof** is how we communicate ideas about the truth or falsehood of a statement to others. There are many different philosophical ideas about what constitutes a proof, but what they all have in common is that a proof is a mode of *communication*, from the person creating the proof to the person digesting it. In this course, we will focus on reading and creating our own written mathematical proofs, which is the standard proof medium in computer science.

As with all forms of communication, the style and content of a proof varies depending on the audience. In this course, the audience for all of our proofs will be an average CSC165 student (and not your TA or instructor).[1]

Some basic examples

What goes into a proof?

A new domain: number theory

Alternating quantifiers revisited

False statements and false proofs

Proof by cases

Generalizing statements

Characterizations

Proof by contrapositive

Greatest common divisor

Modular arithmetic

Proof by contradiction

Induction

In the previous chapters we have studied how to express statements precisely using mathematical expressions, and how to analyze and prove the truth or falsehood of these statements using a variety of proof techniques. In this chapter, we will introduce a new and very important proof technique called **induction**, and use it to prove statements of the form, $\forall n \in \mathbb{N},\ P(n)$.

You may wonder why we need this new technique when we were already proving universal statements in the last chapter just fine without induction. It turns out that many interesting statements in number theory and most other domains cannot be proven or disproven easily with just the techniques from the previous chapter. We will first motivate the principle of induction using an example from modular arithmetic. Then we will apply induction to other statements in number theory, and then to new domains, using induction to prove properties about sequences and to find expressions for various ways of counting combinatorial objects.

The principle of induction

Examples from number theory

Combinatorics

Incorrect proofs by induction

Looking ahead: strong induction

Analyzing Algorithm Running Time

When we first begin writing programs, we are mainly concerned with their correctness: do they work the way they're supposed to? As our programs get larger and more complex, we add in a second consideration: are they designed and documented clearly enough so that another person can read the code and make sense of what's going on? These two properties – correctness and design – are fundamental to writing good software. However, when designing software that is meant to be used on a large scale or that reacts instantaneously to a rapidly-changing environment, there is a third consideration which must be taken into account when evaluating programs: the amount of time the program takes to run.

In this chapter, you will learn how to formally analyze the running time of an algorithm, and explain what factors do and do not matter when performing this analysis. You will learn the notation used by computer scientists to represent running time, and distinguish between best-, worst-, and average-case algorithm running times.

A motivating example

Asymptotic growth

One special case of Big-Oh: O(1)

Omega and Theta

Properties of Big-Oh, Ometa, and Theta

Back to algorithms

Worst-case and best-case running times

Don't assume bounds are tight!

Average-case analysis

Graphs and Trees

Our final mathematical domain of study is a powerful and ubiquitous way of representing entities and the relationships between them. If this sounds generic, that's because it is: this type of representation is abstract enough that we can use it to model concepts as varied as geographic locations and routes, animals and plants in an ecosystem, or people in a social network.

In this chapter, you will begin your study of *graph theory*, learning how to precisely define different types of these models, called *graphs*, and (of course) state and prove properties of these entities. While we are only scratching the surface in this chapter, the material you learn here will serve as a useful foundation in many future courses in computer science.

Initial definitions

Paths and connectedness

A limit for connectedness

Cycles and trees

Rooted trees

Looking Ahead

There are many beautiful ideas in Computer Science that make fundamental use of mathematical expression and reasoning. While we cannot do justice to these topics in these notes (many of them are deep), we would like to give you a glimpse of the power of mathematical reasoning in Computer Science. You will learn these and other topics in depth in other Computer Science courses at University of Toronto, including CSC236, CSC263, CSC373, CSC438, CSC448, CSC463, and CSC473.

Turing's legacy: the limitations of computation

Gödel's legacy: the limitations of proofs

P versus NP

Other cool applications: Cryptography