Course notes for CSC165H:
Mathematical expression and reasoning for computer science

Gary Baumgartner and Danny Heap and Richard Krueger
krueger@cs.toronto.edu
http://www.cs.toronto.edu/~krueger/165/

Fall 2007

Chapter 1

Introduction

1.1 What's csc165 about?

In addition to hacking, computer scientists have to be able to understand program specifications, APIs, and their workmate's code. They also have to be able to write clear, concise documentation for others.

In our course you'll work on:

Expressing yourself clearly (using English and mathematical expression).

Understanding technical documents and logical expressions.

Deriving conclusions from logical arguments, several proof techniques.

Analyzing program efficiency.

Binary notation, why it's used and how to manipulate it. Toward the end of the course you'll learn about how real numbers are represented in computers with floating-point data types.

In this course we care about communicating precisely:

Knowing and saying what you mean

Understanding what others say and mean

We want this course to help you during your university career whenever you need to read and understand technical material (course textbooks, assignment specifications).

Who needs csc165?

You need this course if you do:

- memorize math
- have trouble explaining what you are doing in a mathematical or technical question
- have trouble understanding word problems

You need this course if you don't:

- like reading math textbooks to learn new math
- enjoy talking about abstract x and y just as much as when concrete examples are given for x and y.
- have a credit for CSC238 in your academic history, or intend to take CSC240.

Why does CS need mathematical expressions and reasoning?

We all enjoy hacking, that is designing and implementing interesting algorithms on computers. Perhaps not all of us associate this with the sort of abstract thinking and manipulation of symbols associated with mathematics. However there is a useful two-way contamination between mathematics and computer science. Here are some examples of branches of mathematics that taint particular branches of Computer Science:

Computer graphics use multi-variable calculus, projective geometry, linear algebra, physics-based modelling

Numerical analysis uses multivariable calculus and linear algebra

Cryptography uses number theory, field theory

Networking uses graph theory, statistics

Algorithms use combinatorics, probability, set theory

Databases use set theory, logic

AI uses set theory, probability, logic

Programming languages use set theory, logic

How to do well in csc165

Check the course web page frequently.

Understand the course information sheet. This is the document that we are committed to live by in this course.

Get in the habit of asking questions and contributing to the answers.

Spend time on this course. The model that we instructors assume is that you work an average of 8 hours per week (3 in lecture, 2 in tutorial, 3 reviewing and working on assignments) on this course. Any material that's new to you will require time for you to really acquire it and use it on other courses.

Don't plagiarize. Passing off someone else's work as your own is an academic offense. Always give generous and complete credit when you consult other sources (books, web pages, other students).

About these notes

These notes were originally created by Gary Baumgartner (and contributed to by many others), expanded and typeset by Danny Heap and Richard Krueger. These notes are written to stand alone and cover the material included in the present CSC165H syllabus without the need of a supplementary textbook (though it's often advantageous to read another perspective). Please let us know of any typos or errors, or anything that seems (unintentionally) confusing on the first read so we can correct it.

In these notes you'll often find superscripts.[1] These often indicate answers to questions worked out in lecture, and through the wonders of word processing, those answers are formatted as endnotes (at the end of the chapter). Our motivation isn't so much to give you whiplash moving your gaze between the question and the answer, as to allow you to form your own answer before looking at our version.

1.2 Human versus technical communication

Natural languages (English, Chinese, Arabic, for example) are rich and full of potential ambiguity. In many cases humans speaking these languages share a lot of history, context, and assumptions that remove or reduce the ambiguity. If we don't share (or choose to momentarily forget) the history and context, there is a rich source of humour in double-meanings created by natural languages, for example consider these headlines (http://www.departments.bucknell.edu/linguistics/semhead.htm)

> Prostitutes appeal to Pope
>
> Iraqi head seeks arms
>
> Police begin campaign to run down jay-walkers
>
> Death may cause loneliness, feelings of isolation
>
> Two sisters reunite after 18 years at checkout counter.[2]

Computers are notorious for lacking a sense of humour, and we communicate with them using extremely constrained languages, programming languages. In programming languages, expressions aren't expected to be ambiguous.

Human technical communication about computing must be similarly constrained. We have to assume less common history and context is shared with the other humans participating in technical communication, and misplaced assumptions can result in catastrophe. We aim for increased PRECISION, that is a smaller tolerance for ambiguity. We will use MATHEMATICAL LOGIC, a precise language, as a form of communication in this course.

Mathematicians share a common dialect to talk precisely about precise concepts in their work (e.g. "differentiable functions are continuous"). Often ordinary words ("continuous") are used with restricted, or special, meanings. The same word may have different technical meaning in different mathematical contexts, for example GROUP may mean one thing in group theory, another in combinatorial design theory.

Since technical language is used between human beings, some degree of ambiguity is tolerated, and probably necessary. For example, an audience of Java programmers would not object to the subtle shift in the meaning used for "a" in the following fragments:[3]

```
/** Sorts a in ascending order */
public void sort(int[] a) ...
```

versus

```
// sets a to 1
a = 1;
```

Since another human is reading our comments, this potential for double meaning is benign. A computer reading our comments would, of course, be unforgiving. However, Java programmers have to assume familiarity with programming from their audience, to avoiding driving others crazy by writing long comments (and being driven crazy by long comments written by others).

In this course we can't assume the necessary context to always conclude that you know what you're saying, so you'll have to demonstrate it explicitly. On the other hand, you will learn to understand somewhat imprecise statements that can be made precise from the context.

1.3 Problem-solving

Of course, as a computer scientist you are expected to do more than express yourself clearly about the algorithms, methods, and classes that you either develop or use. You must also work on solving new and challenging problems, sometimes without even knowing in advance whether a solution is possible. You will learn to balance insights that may not be fully articulate, with rigour that convinces yourself and others that your insights are correct. You need both pieces to succeed.

We will try to teach some techniques that increases your chances of gaining insight into mathematical problems that you encounter for the first time. Although these techniques aren't guaranteed to succeed for every mathematical problem, they work often enough to be useful.

Much of our approach is based on George Polya's approach in "How to Solve it," and other books following that approach. Although you can find lots of references to this on the web, here's a précis of Polya's approach:

Understand the problem: Make sure you know what is being asked, and what information you've been given. It helps to re-state the problem (sometimes several times) in your own words, perhaps representing it in different ways or drawing some diagrams.

Plan a solution: Perhaps you've seen a similar problem. You might be able to use either its result, or the method of solving it. Try working backwards: assume you've solved the problem and try to deduce the next-to-last step in solving it. Try solving a simpler version of the problem, perhaps solving small or particular cases.

Carry out your plan: See whether your plan for a solution leads somewhere. It may be necessary to repeat parts of the earlier steps. When you're stuck, try to articulate exactly what you're missing.

Review any solution you achieve: Look back on any pieces of the puzzle you solve, try to remember what lead to breakthroughs and what blocked progress. Carefully test your solution until you're convinced (and can convince a skeptical peer) that you've got a solution. Extend the solved problem to new problems.

Notice how these steps differ from our usual pattern of either avoiding work on a problem (staring at a blank page), or diving in without a plan. The very idea of separating the plan for finding a solution from the act of finding a solution seems weird and unnatural. We'll add a further unnatural suggestion: you should keep a record (notes or a journal) of your problem-solving attempts. This turns out to be useful both in solving the problem at hand and later, related problems.

Here's an example of a real-life problem (eavesdropping on a streetcar) that you might apply Polya's approach to. You're swinging from the grip on a streetcar during rush hour, and you hear the following conversation fragments behind you, between persons A and B:

Person A: I haven't seen you in ages! How old are your three kids now?

Person B: The product of their ages (in years) is 36. [You begin to suspect that B is a difficult conversation partner].

Person A: That doesn't really answer my question...

Person B: Well, the sum of their ages (in years) is — [at this point a fire engine goes by and obscures the rest of the answer].

Person A: That still doesn't really tell me how old they are.

Person B: Well, the eldest plays piano.

Person A: Okay, I see, so their ages are — [at this point you have to get off, and you miss the answer].

1.4 Inspirational puzzles

As inspiration to the usefulness of mathematical logic and reasoning, we submit to you the following puzzles. Each is related to problems common in computer science, and are interesting in their own right. The difficulty of these puzzles varies widely, and we intentionally give no indication of their presumed difficulty nor their solutions. By the end of this course, you will likely be able to solve most of these puzzles (indeed many you may be able to solve now).

- 3 boxes
  Suppose you are a contestant on a game show and you are presented with 3 boxes. Inside one is a prize, which you will win if you chose the correct box. The game goes thusly: you choose a box, and the host opens one of the remaining boxes which is empty. You may then switch your choice to the remaining box or stay with your original choice. Which box would you choose for the best chance of winning the prize? Why?

- 3 labelled boxes
  In the next round, you are again presented with 3 boxes, one containing a small prize. This time, you must choose one box, and if you choose correctly, you win the prize. On closer examination, you notice a label on each box.

The prize is not here.	The prize is here.	The prize is not here.

  Which box do you choose?

- 2 labelled boxes
  In the next round, you are presented with just two boxes, with one containing a prize. The host explains that two stagehands Adam and Brian pack the boxes. Adam always puts a true statement on the box, and Brian always puts a false statement on the box. You don't know who packed the boxes, or even if they were both packed by the same person or different people. The boxes say:

The prize is not here.	Exactly one box was packed by Brian.

  Which box do you choose?

- 2 labelled boxes surprise
  In the final round, you are presented with two more boxes. The host tells you one box contains the grand prize, but if you choose the wrong one, you lose everything. The labels say:

The prize is not here.	Exactly one statement on the boxes is false.

  You reason that if the statement on the right box is true, the left box statement must be false, so the prize is in the left box. If the statement on the right box is false, either both statements are true or both are false. They cannot both be true, since the one on the right is false. So both are false, and the prize must be in the left box. So you choose the left box.

  You open it and it's empty! The host claims he didn't lie to you. What's wrong? (The difference between this and the previous puzzle is subtle but basic and essential for rigorous treatment of logic.)

- Knights and Knaves
  On the island of knights and knaves, every inhabitant is either a knight or a knave. Knights always tell the truth, and knaves always lie. You come across two inhabitants, let's call them A and B.

  1. Person A says: "I am a knave or B is a knight." Can you determine what A and B are?
  2. Person A says: "We are both knaves." What are A and B?

3. Person A says: "If B is a knave, then I'm a knight." Person B says: "We are different." What are A and B?
4. You ask A: "Are you both knights?" A answers either Yes or No, but you don't know enough to solve the problem. You then ask B: "Are you both knaves?" B answers either Yes or No, and now you know the answer. What are A and B?
5. A and B are guarding two doors, one leading to treasure and one leading to a ferocious lion which will surely eat you. You must choose one door. You may ask one guard one yes/no question before choosing a door. What question do you ask? Is it easier if you know one is a knight and one is a knave (but you don't know which is which)?

- Mother Eve

  Theorem: There is a woman on Earth such that is she becomes sterile, the whole human race will die out because all women will become sterile.

  Proof: Either all women will become sterile or not. If yes, then any woman satisfies the theorem. If no, then there is some woman that does not become sterile. She is then the one such that if she becomes sterile (but she does not), the whole human race will die out.

  Is this argument convincing?

- Santa Claus

  Wife: Santa Claus exists, if I am not mistaken.
  Husband: Well of course Santa Claus exists, if you are not mistaken.
  Wife: Hence my statement is true.
  Husband: Of course!
  Wife: So I was not mistaken, and you admitted that if I was not mistaken, then Santa Claus exists. Therefore Santa Claus exists.

  Are you convinced? Why or why not?

- Daemon in a Pentagon

  There is a pentagon and at each vertex there is an integer number. The numbers can be negative, but their sum is positive. A daemon living inside the pentagon manipulates the numbers with the following atomic action. If it spots a negative number at one of the vertices, it adds that number to its two neighbours and negates the number at the original vertex. Prove that no matter what numbers we start with, eventually the daemon cannot change any of the numbers.

Chapter 1 Notes

[1]Like this.

[2]The word "appeal" in the first headline has two meanings, so one interpretation is that the Pope is fond of prostitutes, and another is that prostitutes have asked the Pope for something. The words "head" and "arms" each have two meanings, so one interpretation is that the body part above some Iraqi person's shoulders is looking for the appendages below their shoulders, and another is that the most senior Iraqi is looking for weapons. The phrase "run down" can mean either hitting with a car or looking for. In the fourth headline, it's not clear to whom death causes loneliness and isolation: the dead person or their survivors. In the fifth headline it's not clear whether the checkout line was moving REALLY slowly, or that the checkout counter was just the location of their reunion.

[3]In the first fragment "a" means "the object referred to by the value in a." In the second "a" means "the variable a."