Computer Science 313

Welcome To This Course!

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #1

What is CPSC 313?

CPSC 313 is...

- a second year course in theoretical computer science offered by the Department of Computer Science at the University of Calgary.
- until recently, the first course in theoretical computer science required for students in the B.Sc. and Honours B.Sc. Computer Science programs.[1]
- a course that applies knowledge and skills from prerequisite courses, in order to introduce various models and ideas that will be used in later computer science courses.

[1] Much of the content of CPSC 313 has been moved to the new course, CPSC 351. It is possible that CPSC 313 will not be available after this term.

The Big Question

This course is about the following question:

What are the fundamental capabilities and limitations of computers?

A consideration of this question is considered in a later course — CPSC 413 — which is also required for students in the B.Sc. and Honours B.Sc. programs in Computer Science.

Several optional courses, including CPSC 513 and 511, also extend this study.

Topic #1: Introduction to Automata Theory

By the end of this course *three* significant mathematical models of computation will be introduced and related.

1. ***Finite State Machines***
2. ***Context-Free Grammars***
3. ***Turing Machines***

We will see that (moving down the list) this is a list of progressively more ***general*** or ***powerful*** models — with the last of these being (at least as) powerful as "real computers."

The first two are *not* as powerful as real computers — but ***applications*** (or "advantages") of these models will also be presented. Among other things, these have applications involving ***compiler development***.

Topic #1: Introduction to Automata Theory

Indeed ***none*** of the formal models, introduced in this course, just "fell from the sky" or got invented to complicate the lives of computer science students.

They are all ***abstractions*** — and useful ***simplifications*** — of various things that were seen to be repeatedly arising in applications like compiler development...

... or in attempts to more clearly say what a "computer" can really do.

Topic #1: Introduction to Automata Theory

This part of the course also makes it possible to consider ***problem solving*** in a potentially useful way: Both

- the job of ***designing a solution for a posed problem*** and
- the job of ***proving correctness of the solution that has been developed***

can be introduced, first in a very simple setting... then in a progressively more complicated settings.

Indeed, the initial settings are so simple that reasonably understandable ***processes*** can be presented, which can be used to carry out these jobs.

"Big ideas" can get introduced without complications — with the potentially confusing and distracting complications considered later.

Topic #2: Introduction to Computability Theory

We will also see that there are problems that ***provably, cannot*** be solved using computers at all!

Furthermore, a general technique for ***proving*** that a problem cannot be solved, by a computer, will be described and used.

This technique — the use of a ***reduction*** — will also be extremely important in CPSC 413.

One occasionally does get asked to "do the impossible" in the workplace. Again... at the end of this course you will learning about a process that allows some ***problems that, provably, have no solutions*** to be identified.

Alternatives can be negotiated and explored once "impossibility" has been demonstrated.

Goals for Learning — and Justifications for Them

- ***Dealing with Abstraction, Part One:*** You will be introduced to problems in computer science that that can be solved using techniques that you learned in one or more mathematics prerequisites — notably including MATH 271. ***You will learn how to apply these techniques from mathematics to solve these problems in computer science.***

  The problems in computer science are simple enough to make the "jump" from the introduction of ideas in mathematics to their application to solve problems in computer science reasonably easy. This will leave you better prepared for later courses, where the ideas in mathematics must be used in more complicated ways.

Goals for Learning — and Justifications for Them

- ***Design:*** You will ***follow a design process*** to develop various kinds of simple machines — or expressions — that can be used to ***solve various computational problems***.

  The intention is that this will leave you better prepared for later situations, where you will be expected to make use of other design processes in more complicated settings.

- ***Analysis:*** You will apply mathematical tools to ***prove that your solutions for problems are correct***.

  Once again, it can be helpful to start to do this in a simple, less distracting, setting. You will be required to prove correctness of solutions for problems, in more complicated settings, in later courses.

Goals for Learning — and Justifications for Them

- ***Proving Impossibility:*** On the other hand, you will also learn about and apply methods to ***prove that various problems cannot be solved*** using various kinds of simple machines.

  This is occasionally necessary, in part because you may encounter these problems in "disguised" forms that require you to do a bit of work to recognize them. This exercise will also improve your understanding of the limits of computation.

Goals for Learning — and Justifications for Them

- ***Dealing with Abstraction, Part Two:*** The problems in computer science discussed in this course will probably seem to be artificial. It may be hard to see why anyone would care about them.

  These are also ***abstractions*** — involving ***simplifications*** of problems that have repeatedly had to be solved.

  In order to motivate the study of these problems you will see at least a *little* bit about how the simple machines, introduced in this course, have been used to design (and use) software tools.

  Time for this will be limited: This course is already pretty full, and most of the rest of your program focusses on this more applied material.

Something That is Not a Learning Goal

- As software developers, we benefit from the work of people who looked at the *real* problems that they had to solve and — somehow — discovered and described the ***abstractions*** (in a sense ***simplifications***) that included the essential elements of multiple problems, which should be focussed on.

  The process of doing this — discovering the important abstractions — will not be studied in this course.

Expected Background and Skills

- ***Discrete Mathematics and Logic:*** You will be asked to write (reasonably simple) mathematical ***proofs*** of various claims. Thus you will apply knowledge and skills from MATH 271 or 273 to kinds of problems that were not considered in that course.
- ***Computer Science:*** It is possible that you will see connections between this course and courses in computer architecture (CPSC 355 and 359): While this course focuses on "mathematical" models, and the architecture courses focus on real low-level computer components, both demonstrate that real computers are related to (or constructed from) *very* simple components.

Lecture Notes and Presentations

Lectures and tutorials will be ***synchronous***, but ***flipped***: Material that students are expected to work through before attending each lecture will be available on D2L.

- This may be provided as one or more videos in which the instructor works through lecture slides resembling those that have been used in past versions of the course. The slides will also be available as a PDF file.
- A set of ***questions for review*** will (sometimes) be provided as well.[2] These were originally intended as an aid for students preparing summaries or "cheat sheets" in preparation for closed-book tests, and should still be useful for review of course material.

[2] These may be included as part of the material for the lecture presentation.

Lecture Notes and Presentations

Lecture presentations (either remote, or face-to-face) will focus on how to solve problems that are related to the above material, which might resemble problems that students are asked to solve in tutorial exercises, assignments and tests. These might also present additional material that supplement the material students have worked through on their own,

- An outline for the presentation will generally be provided ahead of time. If there is time for it, students can try to complete this outline on their own, before the lecture presentation, to test their understanding of material in the lecture notes.
- A completed version of the outline will be made available shortly after the lecture presentation.

Tutorials and Tutorial Exercises

Problems to be solved in tutorials will be described ***before*** the tutorials.

- Students will be asked to try to solve these problems before the tutorial too.
- In either face-to-face or remote meetings during scheduled tutorial times, teaching assistants will help students to see *how* these problems can be solved and *how* students' solutions can be evaluated.
- Written ***solutions*** for the problems will be made available, on D2L, ***after*** the tutorials.

Figuring out how to solve tutorial exercise problems will help students to complete problems on later assignments and tests.

Resources: Recommended References

- Michael Sipser
  Introduction to the Theory of Computation
  Third Edition, CENGAGE Learning, 2012

  A very readable reference for Topic #1 in this course.

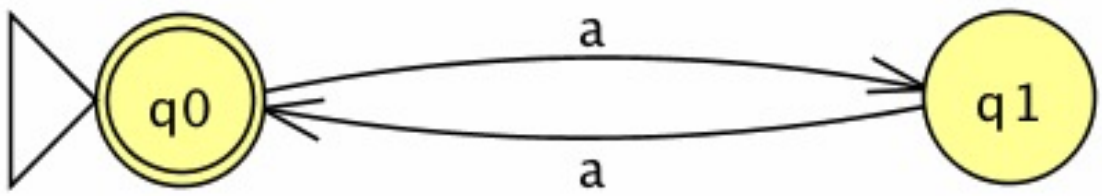
- John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman
  Introduction to Automata Theory, Languages, and Computation
  Third Edition, Pearson Education, 2016

  A readable (and better) reference for Topic #2 in this course.

While these are available at the University of Calgary library they are ***not*** available at the bookstore: Both are now too extensive to be worth their cost.

Resources: Software

JFLAP is a (Java-based) software tool you will be able to use in CPSC 313 to draw pictures of simple computational devices, run them on various inputs, and investigate their properties.



The above picture of a "deterministic finite automaton" was produced using JFLAP. You will be given access to this tool — and to documentation that will help you to use it — in this course.

Advice

Participation in lectures and tutorials, use of the instructor's office hours, and use of the Java development exercises described below, is ***highly recommended***.

- There are ***lots*** of "little things to do" in this course. Participation in the scheduled activities helps you to get all these things done without being overwhelmed and also helps you to make sure that you understand the material that is being presented and applied.
- The instructor is available for extra help during the office hours scheduled for this course. Asking questions can both you and other students, because it can help the instructor to see which material needs to be explained again (or differently).

Additional Material and Activities

1. A ***lecture supplement*** includes additional information about ***course administration***. Some of this is subject to approval by the Faculty of Science — and may be subject to change before the class begins. Quite a bit of it can also be found on the course outline.
2. Another ***lecture supplement*** provides information about mathematical proofs and two versions of the proof technique, ***mathematical induction***. This should review material that was also covered in the discrete mathematics prerequisite, MATH 271 or 273, and ***students should read this material before attending the first lecture presentation.***

Additional Material and Activities

3. A ***lecture outline*** for the scheduled lecture presentation is now available. This concerns the material about mathematical proofs, and the proof technique "mathematical induction", that are reviewed in the above lecture supplement.

   If time permits, please try to solve the problem that will be solved in the lecture, before it — so you can compare your solution to the one that is developed during the presentation.

   If time permits, students will form small groups at the end of the lecture period, so that they can get to know a few more students in the course.

Additional Material and Activities

4. Additional information, that might be of interested, is also included in the area for this lecture on the course web site.
5. Tutorials begin during the second week of class, and the ***tutorial exercise*** for that week's first tutorial is now available. It continues the mathematics review that is the main part of the presentation for this lecture.

   Please work through the material in this exercise ***before*** attending the tutorial if you can.

Additional Material and Activities

6. The next lecture begins the first major topic of this course — concerning ***finite automaton and regular languages*** — by introducing "deterministic finite automata" and describing how that they can be used to process strings of symbols.