Computer Science 513

Introduction to CPSC 513

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #1

What is CPSC 513?

CPSC 513 is a senior optional course on computability theory.
It concerns the following questions.

Which problems can be solved using computers — and which cannot?

*How can we **prove** that a problem cannot be solved using a computer?*

What is CPSC 513?

A few more questions might also be addressed.

Which "limited" models of computation are useful? How can studying them improve our ability to solve some problems well?

Are there relationships between "unsolvable" problems that can help us to understand these problems — which might be useful if our understanding of "computation" should change?

Relationship To Other Courses

- CPSC 351 (or CPSC 313) is the prerequisite for this course and includes an introduction to computability.
- ***Computational Complexity Theory*** is another part of theoretical computer science that is closely related to computability theory. This is introduced in CPSC 413 and studied in more detail in CPSC 511. It turns out that many of the techniques and results in these two areas are closely related!

  Indeed, many techniques and concepts that are used to solve problems in complexity theory were developed and used to solve problems in computability theory first!

Something a Bit Strange

- Most textbooks for a course like CPSC 351, or 313 (that introduce computability theory, along with automata) use a ***Turing machine*** as a model of computation. These also generally focus on the ***decidability of languages***.
- Courses for senior courses in computability generally use *different* computational models, instead, and often focus on the *computability of functions* $f : \mathbb{N}^k \to \mathbb{N}$ for an integer $k \geq 1$).
- As a result, material that reviews material from CPSC 351 or 313 will be found near the *middle* or *end* of CPSC 513 — we will need to introduce new material, concerning computability of functions of (sequences of) natural numbers, before that.

Necessary Background and Skills: Mathematics

- The ability to *read and write mathematical proofs* will be extremely important.
- As a result, an ability to use material that was introduced in CPSC 251, or MATH 271 or 273 will be assumed.

Topics

1. Models of Computation
2. First Hard Problems: Universality, Diagonalization, Reductions, and Identifiication of Functions that are Not Computable, Sets that are Not Computable, and Sets that are Not Computably Enumerable
3. Computations with Oracles, and the Arithmetic Hierarchy
4. Post's Problem
5. Introduction to the Chomsky Hierarchy

Topic: Models of Computation

- We will begin with a definition and study of ***primitive recursive functions*** — an early attempt to formalize the notion of computation.
- These were initially called “recursive functions” — because it was not initially realized that there are any computable total functions that are ***not*** primitive recursive.
- This course will (eventually) include a proof that there *are*, indeed, computable total functions that are not primitive recursive.
- Nevertheless, the primitive recursive functions still play an important role in computability theory.

Topic: Models of Computation

- We will continue with an introduction of a programming language, the set of "$\mathcal{S}$ programs" that can be written using this language, and the computable functions that are defined by these programs.
- Additional machine models — including ***Post-Turing Programs*** and generalizations of the ***Turing machines*** you have already seen — will be introduced.
- Results establishing the ***equivalence*** of these models, up to differences involving different choices of how to ***encode*** computational problems, will be established.

Topic: First Hard Problems

- One of the things you learned about in CPSC 351, or 313, is a ***universal Turing machine*** — which receives a string encoding a Turing machine M (with some input alphabet Σ) and an encoding of a string $\omega \in \Sigma^\star$, which accepts its input if and only if M accepts ω.
- This is, effectively, a Turing machine that behaves as a "Turing machine emulator."
- This course will continue with a presentation of a "universal $\mathcal{S}$-program" — effectively, an $\mathcal{S}$-program that acts as an "$\mathcal{S}$-program emulator," in much the same way.
- This will help to identify a partial function from $\mathbb{N}$ to $\mathbb{N}$ that is provably ***not*** computable by an $\mathcal{S}$-program.

Topic: First Hard Problems

- ***Diagonalization*** — a proof technique that you (almost certainly) learned about in CPSC 351 or 313 — will be reviewed. This will be used to identify a function that is provably not recursive.
- Following this, ***reducibilities*** and ***reductions*** be introduced. This will resemble (and, probably, generalize) material that you saw in CPSC 351 or 313 and, again, at the end of CPSC 413.
- We will see that these are an important part of a process that you can follow in order to prove that other functions are not computable.

Topic: First Hard Problems

Lectures continue with results that can be established once this background is available.

- ***Rice's Theorem:*** A general result establishing that all non-trivial "semantic" properties of programs are undecidable
- ***Recursion Theorem:*** A fundamental result (or pair of results) about the application of computable functions to their own descriptions
- Demonstration of a ***computable total function that is provably not primitive recursive***

Topic: Computations with Oracles, and the Arithmetic Hierarchy

When most modern textbooks on computability theory or complexity theory *informally* introduce reductions, they often seem to be describing "subroutine reductions" — essentially, programs to solve one problem (or decide one language) that are allowed to use a hypothetical "black box," or subroutine that (somehow) solves *another* problem — or decides another language.

- This can be formalized as an ***oracle reduction***. These reductions, and their properties, will be studied near the end of this course.
- They will also be used to define an infinite collection of (increasingly more "impossible to decide") sets of subsets of $\mathbb{N}$ and decision problems, called the ***arithmetic hierarchy***.

Topic: Post's Problem

- Lectures will continue with the consideration of a problem — ***Post's problem*** — that concerns the finer structure of one of classes, Σ_1, in the arithmetic hierarchy and whose study considerably advanced computational computability theory — somewhat like the way that the study of the question "Is $\mathcal{P} = \mathcal{NP}$?" advanced computational complexity theory.

  It will be easier to introduce this problem *after* the arithmetic hierarchy has been more formally introduced.

Topic: Introduction to the Chomsky Hierarchy

The ***Chomsky Hierarchy*** is a containment hierarchy of classes of formal ***grammars*** and the languages that these can represent.

- We will see that ***regular languages*** — introduced in CPSC 351 or 313 — are the languages that have "regular grammars".
- ***Context-free languages*** and ***context-sensitive languages*** will be introduced, along with corresponding classes of formal grammars.
- Arbitrary ***formal grammars*** will be introduced — and we will see that these correspond to the class of ***recursively enumerable languages*** that were introduced along with Turing machines.

Resources: Additional References

There is no textbook for this course, and no references will be available from the bookstore. The instructor is using the following references when preparing lecture material.

- Martin D. Davis, Ron Sigal, and Elaine J. Weyuker
  Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science (Second Edition)
  Academic Press, 1994
- Borut Robič
  The Foundations of Computability Theory (Second Edition)
  Springer, 2020

Resources: Lectures

Preparatory Reading

- This will be a “flipped course,” so that students are expected to have worked through material ***before*** each lecture.
- In particular, lecture slides will be available as a PDF file — or as a lecture video, if time permits. This material is based on slides used by instructor in face-to-face lectures, in this course, in the past.

Resources: Lectures

Lecture Activities

- During the lecture time the instructor might work through an activity, or help small students to solve problems when working in small groups.
- During presentations, the instructor will try to focus on ***how problems are being solved***, rather than the format of completed solutions, during the lectures.
- ***Participation in the lectures is highly recommended*** because these will prepare students to solve problems on assignments and tests.

Resources: Lectures

Material for After the Lectures

- Review questions, that can generally be answered by examining the preparatory reading and considering what happened during lectures, will provided for test preparation.
- Additional practice problems — resembling problems solved during lecture presentations and, sometimes, in assignments and tests — will be made available too.

Course Administration

- A supplemental document, including addition about course administration, will be included with the preparatory material for the first lecture.