Schedule of Topics
CSC 341, “Automata, Formal Languages, and Computational Complexity”
Department of Computer Science
Grinnell College
August 30, 2019

August 30: How can one measure the computational abilities of a programmable machine (or, equivalently, of a family of special-purpose machines that share a general design)? How can one compare the computational abilities of two such machines? How can one model computation so as to make such measurements and comparisons precise and rigorously demonstrable?

Reading: Sipser, Introduction to the Theory of Computation, chapter 0 (pages 1–28)

September 2: What techniques are commonly used in reasoning about computational models? How is mathematical induction related to structural induction? How is simple induction related to course-of-values induction?

Reading: Sipser, from the beginning of chapter 1 through section 1.1 (pages 31–47)

September 4: What kinds of problems can deterministic finite automata solve?

Reading: Sipser, section 1.2 (pages 47–63)

September 6: What kinds of problems can nondeterministic finite automata solve? Can nondeterministic finite automata emulate deterministic ones? Can deterministic finite automata emulate nondeterministic ones?

Reading: Sipser, section 1.3 (pages 63–76)

September 9: What kinds of formal languages can regular expressions describe? Considered as a model of computation, are regular expressions more powerful or less powerful than deterministic finite automata?

Reading: Sipser, from section 1.4 to the end of chapter 1 (pages 77–99)

September 11: What kinds of problems can deterministic finite automata not solve? How can one prove, for a given problem, that no deterministic finite automaton can solve it?

Reading: Sipser, from the beginning of chapter 2 through section 2.1 (pages 101–111)

September 13: What kinds of formal languages can context-free grammars generate?

Reading: Sipser, from the beginning of section 2.2 through the subsection “Examples of pushdown automata” (pages 111–116)

September 16: What kinds of problems can pushdown automata solve?

Reading: Sipser, subsection “Equivalence with context-free grammars” of section 2.2 (pages 117–125)

September 18: Considered as a model of computation, are context-free grammars more powerful or less powerful than deterministic finite automata? Are they more powerful or less powerful than pushdown automata?

September 20: (pause for breath)

Reading: Sipser, section 2.3 (pages 125–129)

September 23: What kinds of problems can pushdown automata not solve? Can pushdown automata emulate deterministic finite automata? Can deterministic finite automata emulate pushdown automata?
Reading: Sipser, Exercises and Problems for chapter 2 (pages 154–162)

**September 25:** What operations on languages preserve regularity? What operations on languages preserve context-freedom?

Reading: Sipser, from the beginning of chapter 3 through section 3.1 (pages 165–175)

**September 27:** What kinds of problems can deterministic Turing machines solve? What kinds of formal languages can they decide, and what kinds can they only recognize? Can deterministic Turing machines emulate pushdown automata? Can pushdown automata emulate deterministic Turing machines?

Reading: Sipser, from the beginning of section 3.2 through the subsection “Multitape Turing Machines” (pages 176–178)

**September 30:** What kinds of problems can multitape Turing machines solve? Can multitape Turing machines emulate deterministic Turing machines? Can deterministic Turing machines emulate multitape Turing machines?

Reading: Sipser, from the subsection “Nondeterministic Turing Machines” to the end of section 3.2 (pages 178–182)

**October 2:** Can nondeterministic Turing machines emulate deterministic ones? Can deterministic Turing machines emulate nondeterministic ones? What other variant kinds of Turing machines have been considered? Which of them are more powerful than deterministic Turing machines? Which of them are less powerful?

Reading: Sipser, sections 7.1 and 7.2 (pages 275–291)

**October 4:** Does the complexity of a problem depend on the nature of the model of computation in which it is solved? If so, how? How might one classify both problems and models of computation so as to explain their relationships concisely? Can one measure the complexity of a problem by the number of steps it takes a Turing machine to solve it? What are some of the pitfalls in defining and using such a measure?

Reading: Sipser, section 7.3 and from the beginning of section 7.4 through the subsection “Definition of NP-Completeness” (pages 292–304)

**October 7:** What kinds of problems can deterministic Turing machines solve in a number of steps that is bounded by some polynomial function of the size of the input? For what kinds of problems can such machines verify proposed solutions, under the same limitation on the number of steps? Are these two classes of problems the same, or is the first a proper subclass of the second?

Reading: Sipser, from the subsection “The Cook-Levin Theorem” to the end of section 7.4 (pages 304–311)

**October 9:** How can a Boolean formula model the sequence of configurations of a Turing machine?

Reading: Sipser, section 7.5 (pages 311–322)

**October 11:** What are some of the least tractable problems for which a Turing machine can verify a solution in polynomial time?

**October 14:** (pause for breath)

Reading: Sipser, Exercises and Problems for chapter 7 (pages 322–330)

**October 16:** What other problems can be proven to be as intractable as SAT?

Reading: Sipser, from section 3.3 to the end of chapter 3 (pages 182–191)

**October 18:** What is the relation between deterministic Turing machines and algorithms? If there is a problem that no deterministic Turing machine can solve, does it follow that there is no algorithm whatever for solving that problem?
Reading: Sipser, from the beginning of chapter 4 through section 4.1 (pages 193–201)

**October 28:** How can one formulate problems about automata, grammars, functions, and the like, and submit them for computational solution? And deterministic Turing machines solve such problems?

Reading: Sipser, from the beginning of section 4.2 through the subsection “The Diagonalization Method” (pages 201–207)

**October 30:** What does it mean to say that some infinite sets are larger than others? How does one measure the size of an infinite set? How does the “diagonalization” proof method work? Is it sound?

Reading: Sipser, from the subsection “An Undecidable Language” in section 4.2 to the end of chapter 4 (pages 207–214)

**November 1:** What kinds of problems, if any, can deterministic Turing machines not solve? How can one prove, for a given problem, that no deterministic Turing machine can solve it?

Reading: Sipser, from the beginning of chapter 5 through section 5.1 (pages 215–226)

**November 4:** How do “reduction” proofs of decidability and undecidability work?

Reading: Sipser, section 5.2 (pages 227–233)

**November 6:** Can a Turing machine solve the Post Correspondence Problem? Is the Post Correspondence Problem a model of computation? If so, is it more powerful or less powerful than the other models of computation that we have studied?

Reading: Handout: “Semi-Thue Processes”

**November 8:** What kinds of formal languages can general grammars (“semi-Thue processes”) generate? Considered as models of computation, are general grammars more powerful or less powerful than deterministic Turing machines?

Reading: Sipser, section 5.3 (pages 234–238)

**November 11:** What is “mapping reducibility”? How does one establish that one language is mapping reducible to another? What does this relation imply about the decidability of the languages?

Reading: Sipser, Exercises and Problems from chapter 5 (pages 239–244)

**November 13:** How can we generalize these strikingly similar mapping-reducibility proofs to show that entire classes of problems that have the same general structure are undecidable?

Reading: Sipser, from the beginning of chapter 6 through section 6.1 (pages 245–252)

**November 15:** Is it possible for a deterministic Turing machine to recover and use information about its own internal structure when performing a computation? If so, what are the implications of this idea, and how far can one take them?

**November 18:** (pause for breath)

Reading: Sipser, sections 8.1 and 8.2 (pages 331–337)

**November 20:** What kinds of problems can deterministic Turing machines solve, using a number of tape cells that is bounded by some polynomial function of the size of the input?

Reading: Sipser, section 8.3 (pages 337–348)

**November 22:** What are some of the least tractable problems that can be solved by deterministic Turing machines using a number of type cells that is bounded by some polynomial function of the size of the input?
Sipser, from the beginning of section 6.4 through the subsection “Optimality of the Definition” (pages 261–265)

**November 25:** How can one measure the complexity of a problem, or of the automaton that solves it? Can such measures be computed by deterministic Turing machines? How can one measure the complexity of a single computation?

Reading: Sipser, from the subsection “Incompressible Strings and Randomness” to the end of chapter 6 (pages 267–272)

**November 27:** Given a Turing-decidable language, is it possible to find the simplest Turing machine that decides it? Could there be an algorithm for finding the simplest Turing machine to solve a given problem? Could there be an algorithm that takes a description of a Turing machine as input and outputs either a shorter description of a Turing machine that recognizes the same language, or (if no such Turing machine exists) the input description unchanged?

Reading: Handout: “Recursive Function Theory,” sections 1 through 9

**December 2:** Starting with a very simple class of “primitive” functions taking natural numbers as arguments and yielding natural numbers as values, what other functions can be derived, using the operations of composition and recursive definition? How can data of other types be encoded as natural numbers? How can one formulate computational solutions to problems involving such data as primitive recursive functions?

Reading: Handout: “Recursive Function Theory,” sections 10 and 11

**December 4:** If we extend the notion of a function to include partial functions, which fail to have values for certain arguments or combinations of arguments, how might one adapt the constructive model for defining functions? What operators might one use to extend the class of primitive recursive functions in this wider universe of discourse? Can one encode partial recursive functions as natural numbers?

Reading: Handout: “Recursive Function Theory,” sections 12 through 14

**December 6:** Can one define partial recursive functions that solve problems about partial recursive functions? Can one encode computations as natural numbers? Are all partial functions partial recursive? How might one find a partial function that is not partial recursive, and how would one prove that it is not partial recursive?

Reading: Handout: “Recursive Function Theory,” section 15

**December 9:** In recursive function theory, what corresponds to the distinction between Turing-recognizable and Turing-decidable languages? How can one apply recursive function theory to study and learn about such languages?

Reading: Handout: “Recursive Function Theory,” section 16 through 19

**December 11:** Can one devise partial-recursive functions that, in some sense, emulate Turing machines? Can one implement Turing machines that compute partial-recursive functions? Which, if either, of these two models of computation is more powerful?

**December 13:** What questions relating to automata theory, formal languages, and computational complexity remain to be answered?