# CS 154 - Introduction to Automata and Complexity Theory

Spring Quarter, 2000

Handout 2 (3/29/00) – Course Description

**Textbook:** The textbook for this course is the course reader *Introduction to Automata and Languages*, by Hopcroft, Motwani, and Ullman. This is to be supplemented by an old textbook called *Introduction to Automata Theory, Languages and Computation*, by Hopcroft and Ullman.

**Prerequisites:** The prerequisite for this course is CS 109, which covers some basic material on finite automata, regular expressions, and context-free grammars. We will review portions of this material in the first few classes.

**Course Outline:** We will cover the following topics in this course. People enrolled in CS154N will only be expected to attend the lectures on the last three topics.

1. *Finite state automata:* Deterministic and non-deterministic finite state machines; regular expressions and languages. Techniques for identifying and describing regular languages; techniques for showing that a language is not regular. Properties of such languages. (*Chapters 1, 2, 3, & 4 in the Course Reader, and Chapters 1, 2 & 3 in the Hopcroft-Ullman book.*)
2. *Context-free languages:* Context-free grammars, parse trees, derivations and ambiguity. Relation to pushdown automata. Properties of such languages and techniques for showing that a language is not context-free. (*Chapters 5, 6, & 7 in the Course Reader, and Chapters 4, 5 & 6 in the Hopcroft-Ullman book.*)
3. *Turing Machines:* Basic definitions and relation to the notion of an algorithm or program. Power of Turing Machines and Church's hypothesis. (*Chapter 8 in the Course Reader, and Chapter 7 in the Hopcroft-Ullman book.*)
4. *Undecidability:* Recursive and recursively enumerable languages. Universal Turing Machines. Limitations on our ability to compute; undecidable problems. (*Chapter 9 in the Course Reader, and Chapter 8 in the Hopcroft-Ullman book.*)
5. *Computational Complexity:* Decidable problems for which no "efficient" algorithms are known. Polynomial time computability. The notion of NP-completeness and problem reductions. Examples of "hard" problems. (*Chapter 10 in the Course Reader, and Chapters 12 & 13 in the Hopcroft-Ullman book.*)

## Motivation and Overview

I have been told by past students that this is one of the more difficult courses in the Computer Science curriculum, but at the same time they enjoyed it the most. Keep this in mind when the going gets tough! The reason for the difficulty is that it covers abstract and mathematical topics which are not very easy to grasp without putting in a good deal of hard work. There are a number of excellent reasons for becoming proficient with the theoretical tools that we will develop in this course.

1. Most of what you learn in the first part of the course will be required in the design or analysis of almost any reasonably complex software or hardware system. For example, the theory of finite state machines and regular expressions is needed for the design of switching circuits, components of compilers such as lexical analysis, pattern-matching, text-editors, unification as needed in Prolog or for automated deduction, and almost any program which processes user commands. The description of programming languages and the design of parsers for them will require an intimate knowledge of context-free grammars. More interestingly, with the presence of a large amount of unstructured text on the World-wide Web, it has become increasingly important to employ techniques taught in the course to extract structured information from this chaos.

2. The second half of the course is concerned with a more philosophical approach to computer science. Here we will be concerned with the basic questions of computability and tractability. Using the concept of Turing Machines we will try to make precise the notion of an algorithm and explore its limitations. We will encounter *undecidable* problems, viz. those which cannot be solved by any algorithm or computer. Even if a problem is decidable it may turn out to be *intractable*, i.e., there does not exist any efficient algorithm to solve that problem. These notions have had (and will continue to have) a profound influence on our approach to using computers to solve problems.

3. Finally, I think the most important role of this course is to turn you into “mathematically mature” computer scientists. This course is quite mathematical and should develop your skills of precise and formal reasoning. These skills will prove to be extremely important in the design, analysis, and verification of complex software and hardware systems.