1.6 Summary of Chapter 1

- *Finite Automata*: Finite automata involve states and transitions among states in response to inputs. They are useful for building several different kinds of software, including the lexical analysis component of a compiler and systems for verifying the correctness of circuits or protocols, for example.

- *Regular Expressions*: These are a structural notation for describing the same patterns that can be represented by finite automata. They are used in many common types of software, including tools to search for patterns in text or in file names, for instance.

- *Context-Free Grammars*: These are an important notation for describing the structure of programming languages and related sets of strings; they are used to build the parser component of a compiler.

- *Turing Machines*: These are automata that model the power of real computers. They allow us to study decidabilty, the question of what can or cannot be done by a computer. They also let us distinguish tractable problems — those that can be solved in polynomial time — from the intractable problems — those that cannot.

- *Deductive Proofs*: This basic method of proof proceeds by listing statements that are either given to be true, or that follow logically from some of the previous statements.

- *Proving If-Then Statements*: Many theorems are of the form "if (something) then (something else)." The statement or statements following the "if" are the hypothesis, and what follows "then" is the conclusion. Deductive proofs of if-then statements begin with the hypothesis, and continue with statements that follow logically from the hypothesis and previous statements, until the conclusion is proved as one of the statements.

- *Proving If-And-Only-If Statements*: There are other theorems of the form "(something) if and only if (something else)." They are proved by showing if-then statements in both directions. A similar kind of theorem claims the equality of the sets described in two different ways; these are proved by showing that each of the two sets is contained in the other.

- *Proving the Contrapositive*: Sometimes, it is easier to prove a statement of the form "if H then C" by proving the equivalent statement: "if not C then not H." The latter is called the contrapositive of the former.

- *Proof by Contradiction*: Other times, it is more convenient to prove the statement "if H then C" by proving "if H and not C then (something known to be false)." A proof of this type is called proof by contradiction.

- *Counterexamples*: Sometimes we are asked to show that a certain statement is not true. If the statement has one or more parameters, then we can show it is false as a generality by providing just one counterexample, that is, one assignment of values to the parameters that makes the statement false.

- *Inductive Proofs*: A statement that has an integer parameter n can often by proved by induction on n. We prove the statement is true for the basis, a finite number of cases for particular values of n, and then prove the inductive step: that if the statement is true for values up to n, then it is true for $n + 1$.

- *Structural Inductions*: In some situations, including many in this book, the theorem to be proved inductively is about some recursively defined construct, such as trees. We may prove a theorem about the constructed objects by induction on the number of steps used in its construction. This type of induction is referred to as structural.

- *Alphabets*: An alphabet is any finite set of symbols.

- *Strings*: A string is a finite-length sequence of symbols.

- *Languages and Problems*: A language is a (possibly infinite) set of strings, all of which choose their symbols from some one alphabet. When the strings of a language are to be interpreted in some way, the question of whether a string is in the language is sometimes called a problem.

1.7 References for Chapter 1

For extended coverage of the material of this chapter, including mathematical concepts underlying Computer Science, we recommend [1].

1. A. V. Aho and J. D. Ullman, *Foundations of Computer Science*, Computer Science Press, New York, 1994.