Computational Complexity: A Modern Approach

Draft of a book: Dated January 2007
Comments welcome!

Sanjeev Arora and Boaz Barak
Princeton University
complexitybook@gmail.com

This is an Internet draft. Some chapters are more finished than others. References and attributions are very preliminary and we apologize in advance for any omissions (but hope you will nevertheless point them out to us).

Please send us bugs, typos, missing references or general comments to complexitybook@gmail.com — **Thank You!!**

Chapter 21

Logic in complexity theory

Very sketchy

As mentioned in the book's introduction, complexity theory (indeed, all of computer science) arose from developments in mathematical logic in the first half of the century. Mathematical logic continues to exert an influence today, suggesting terminology and choice of problems (e.g., "boolean satisfiability") as well as approaches for attacking complexity's central open questions. This chapter is an introduction to the basic concepts.

Mathematical logic has also influenced many other areas of computer science, such as programming languages, program verification, and model checking. We will not touch upon them, except to note that they supply interesting examples of hard computational problems —ranging from NP-complete to EXPSPACE-complete to undecidable.

The rest of the chapter assumes only a nodding familiarity with logic terminology, which we now recount informally; for details see a logic text.

A *logic* usually refers to a set of rules about constructing *valid sentences*. Here are a few logics we will encounter. *Propositional logic* concerns sentences such as $(p \vee \neg q) \wedge (\neg p \vee r)$ where p, q, r are boolean variables. Recall that the SAT problem consists of determining the satisfiability of such sentences. In *first order logic*, we allow *relation* and *function* symbols as well as *quantification* symbols $\exists$ and $\forall$. For instance, the statement $\forall x S(x) \neq x$ is a first order sentence in which x is quantified universally, $S()$ is a unary relation symbol and $\neq$ is a binary relation. Such logics are used in well-known axiomatizations of mathematics, such as Euclidean geometry, Peano Arithmetic or Zermelo Frankel set theory. Finally, *second order logic* allows sentences in which one is allowed quantification over *structures*, i.e., functions and relations. An example of a second order sentence is $\exists S \forall x S(x) \neq x$, where S is a unary relation symbol.

A sentence (or collection of sentences) in a logic has no intrinsic "meaning." The meaning —including truth or falsehood—can be discussed only with reference to a *structure*, which gives a way of interpreting all symbols in the sentence. To give an example, Peano arithmetic consists of five sentences ("axioms") in a logic that consists of symbols like $S(x)$, $=$, $+$ etc. The standard structure of these sentences is the set of positive integers, with $S()$ given the intepretation of "successor function," $+$ given the interpretation of addition, and so on. A structure is said to be a *model* for a sentence or a group of sentences if those sentences are true in that structure.

Finally, a *proof system* consists of a set of sentences Σ called *axioms* and one or more *derivation* *rules* for deriving new sentences from the axioms. We say that sentence σ can be *proved from* Σ, denoted $\Sigma \vdash \sigma$, if it can be derived from Σ using a finite number of applications of the derivation rules. A proveable sentence is called a *theorem*.

Note that a theorem is a result of a mechanical (essentially, algorithmic) process of applying derivation rules to the axioms. There is a related notion of whether or not σ is *logically implied by* Σ, denoted $\Sigma \models \sigma$, which means that every model of Σ is also a model of σ. In other words, there is no "counterexample model" in which the axioms Σ are true but σ is not. The two notions are in general different but Gödel in his *completeness* theorem for first order theories exhibited a natural set of derivation rules such that logically implied sentences are exactly the set of theorems. (This result was a stepping stone to his even more famous *incompleteness* theorem.)

Later in this chapter we give a complexity-theoretic definition of a proof system, and introduce the area of *proof complexity* that studies the *size* of the smallest proof of a mathematical statement in a given proof system.

21.1 Logical definitions of complexity classes

Just as Church and others defined computation using logic without referring to any kind of computing machine, it is possible to give "machineless" characterizations of many complexity classes using logic. We describe a few examples below.

21.1.1 Fagin's definition of NP

In 1974, just as the theory of **NP**-completeness was coming into its own, Fagin showed how to define **NP** using second-order logic. We describe his idea using an example.

Example 21.1
(Representing 3-COLOR) We show how to represent the set of 3-colorable graphs using second order logic.

Let E be a symbol for a binary relation, and C_0, C_1, C_2 be symbols for unary relations, and $\phi(E, C_0, C_1, C_2)$ be a first order formula that is a conjunction of the following formulae where $i+1, i+2$ are meant to be understood modulo 3:

$$\forall u, v \quad E(u,v) = E(v,u) \tag{1}$$

$$\forall u \quad \wedge_{i=1,2,3} \left(C_i(u) \Rightarrow \neg(C_{i+1}(u) \vee C_{i+2}(u)\right) \tag{2}$$

$$\forall u C_i(u) \vee C_{i+1}(u) \vee C_{i+2}(u) \tag{3}$$

$$\forall u, v \quad E(u,v) \Rightarrow \wedge_{i=1,2,3}(C_i(u) \Rightarrow \neg C_i(v)) \tag{4}$$

What set of E's defined on a finite set satisfy $\exists C_0 \exists C_1 \exists C_2 \phi(E, C_0, C_1, C_2)$? If E is defined on a universe of size n (i.e., u, v take values in this universe) then (1) says that E is symmetric, i.e., it may be viewed as the edge set of an undirected graph on n vertices. Conditions (2) and (3) say that C_0, C_1, C_2 partition the vertices into three classes. Finally, condition (4) says that the partition is a valid coloring.

Now we can sketch the general result. To represent a general **NP** problem, there is a unary relation symbol that represents the input (in the above case, E). The witness is a tableau (see Chapter 2) of an accepting computation. If the tableau has size n^k, the witness can be represented by a k-ary relation (in the above case the witness is a 3-coloring, which has representation size $3n$ and hence was represented using 3 unary relations). The first order formula uses the Cook-Levin observation that the tableau is correct iff it is correct in all 2×3 "windows".

The formal statement of Fagin's theorem is as follows; the proof is left as an exercise.

Theorem 21.2 (Fagin)
To be written.

21.1.2 MAX-SNP

21.2 Proof complexity as an approach to NP versus coNP

Proof complexity tries to study the size of the smallest proof of a statement in a given proof system. First, we need a formal definition of what a proof system is. The following definition due to Cook and Reckow focuses attention on the intuitive property that a mathematical proof is "easy to check."

Definition 21.3
A *proof system* consists of a polynomial-time Turing machine M. A statement T is said to be a *theorem* of this proof system iff there is a string $\pi \in \{0,1\}^*$ such that M accepts (T, π).

If T is a theorem of proof system M, then the *proof complexity of T with respect to M* is the minimun k such that there is some $\pi \in \{0,1\}^k$ for which M accepts (T, π).

Note that the definition of theoremhood ignores the issue of the length of the proof, and insists only that the M's running time is polynomial in the input length $|T| + |\pi|$. The following is an easy consequence of the definition and the motivation for much of the field of proof complexity.

Theorem 21.4
A proof system M in which $\overline{\mathsf{SAT}}$ has polynomial proof complexity exists iff $\mathbf{NP} = \mathbf{coNP}$.

Many branches of mathematics, including logic, algebra, geometry, etc. give rise to proof systems. Algorithms for SAT and *automated theorem provers* (popular in some areas of computer science) also may be viewed as proof systems.

21.2.1 Resolution

This concerns

21.2.2 Frege Systems

21.2.3 Polynomial calculus

21.3 Is P $\neq$ NP unproveable?