

Lecture 6

The Circuit Value Problem

In the early 1970s, Stephen Cook [31] and independently Leonid Levin [78] showed that the Boolean satisfiability problem (SAT)—whether a given a Boolean formula has a satisfying truth assignment—is *NP*-complete. Thus Boolean satisfiability is in *P* iff $P = NP$. This theorem has become known as the *Cook–Levin theorem*. Around the same time, Richard Karp [69] showed that a large number of optimization problems in the field of operations research such as the traveling salesperson problem, graph coloring, bin packing, and many others were all interreducible and therefore *NP*-complete. These two milestones established the study of *NP*-completeness as an important aspect of theoretical computer science. In fact, the question of whether $P = NP$ is today widely considered one of the most important open questions in all of mathematics.

There is a theorem of Ladner [77] that plays the same role for the $P = NLOGSPACE$ or $P = LOGSPACE$ question that the Cook–Levin theorem plays for the $P = NP$ question. The decision problem involved is the *circuit value problem* (CVP): given an acyclic Boolean circuit with several inputs and one output and a truth assignment to the inputs, what is the value of the output? The circuit can be evaluated in deterministic polynomial time; the theorem says that this problem is \leq_m^{\log} -complete for *P*. It follows from the transitivity of \leq_m^{\log} that $P = NLOGSPACE$ iff $CVP \in NLOGSPACE$ and $P = LOGSPACE$ iff $CVP \in LOGSPACE$.

Formally, a *Boolean circuit* is a program consisting of finitely many assignments of the form

$$\begin{aligned}
 P_i &:= 0, \\
 P_i &:= 1, \\
 P_i &:= P_j \wedge P_k, \quad j, k < i, \\
 P_i &:= P_j \vee P_k, \quad j, k < i, \quad \text{or} \\
 P_i &:= \neg P_j, \quad j < i,
 \end{aligned}$$

where each P_i in the program appears on the left-hand side of exactly one assignment. The conditions $j, k < i$ and $j < i$ ensure acyclicity. We want to compute the value of P_n , where n is the maximum index.

Theorem 6.1 *The circuit value problem is \leq_m^{\log} -complete for P .*

Proof. We have already argued that $\text{CVP} \in P$. To show hardness, we will reduce an arbitrary $A \in P$ to CVP. Let M be a deterministic single-tape polynomial-time-bounded TM accepting A , say with time bound n^c . Let Γ be the worktape alphabet of M and let Q be the set of states of M 's finite control. The transition function δ of M is of type $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$. Intuitively, $\delta(p, a) = (q, b, d)$ says, "When in state p scanning symbol a on the tape, print b on that cell, move in direction d , and enter state q ." We can encode configurations of M over a finite alphabet Δ as usual.

Now given x of length n , think of the successive configurations of M on input x as arranged in an $(n^c + 1) \times (n^c + 1)$ time/space matrix R with entries in Δ . The i^{th} row of R is a string in Δ^{n^c+1} describing the configuration of the machine at time i . The j^{th} column of R describes what is going on at tape cell j throughout the history of the computation.

For example, the i^{th} row of the matrix might look like

⊢	a	b	a	a	b	a	b	a	a	b	a	b	□	□	□
						p									

and the $i + 1^{\text{st}}$ might look like

⊢	a	b	a	a	b	b	b	a	a	b	a	b	□	□	□
								q							

This would happen if $\delta(p, a) = (q, b, R)$. The elements of Δ are thus of the form

$$\begin{array}{|c|} \hline a \\ \hline \end{array}
 \quad \text{or} \quad
 \begin{array}{|c|} \hline a \\ \hline q \\ \hline \end{array}$$

for $a \in \Gamma$ and $q \in Q$.

The conditions we will write down amount to a set of *local consistency conditions* on $(n^c + 1) \times (n^c + 1)$ matrices over Δ . Each local consistency condition is a relation on the entries of the matrix in a small neighborhood of some location i, j . The conjunction of all these local consistency conditions is enough to determine the matrix R uniquely.

Our circuit will involve Boolean variables

$$\begin{aligned} P_{ij}^a, & 0 \leq i, j \leq n^c, a \in \Gamma, \\ Q_{ij}^q, & 0 \leq i, j \leq n^c, q \in Q. \end{aligned}$$

The variable P_{ij}^a says, “The symbol occupying tape cell j at time i is a ,” and the variable Q_{ij}^q says, “The machine is in state q scanning tape cell j at time i .”

Now we write down a set of conditions in terms of the P_{ij}^a and Q_{ij}^q describing all ways that the machine could be in state q scanning tape cell j at time i or that the symbol occupying tape cell j at time i is a .

For $1 \leq i \leq n^c$, $0 \leq j \leq n^c$, and $b \in \Gamma$, we include the assignment

$$P_{ij}^b := \bigvee_{\delta(p,a)=(q,b,d)} (Q_{i-1,j}^p \wedge P_{i-1,j}^a) \quad (6.1)$$

$$\vee (P_{i-1,j}^b \wedge \bigwedge_{p \in Q} \neg Q_{i-1,j}^p). \quad (6.2)$$

in our circuit. Intuitively, this says, “The symbol occupying tape cell j at time i is b if and only if either the machine was scanning tape cell j at time $i - 1$ and printed b (clause (6.1)), or the machine was not scanning tape cell j at time $i - 1$ and the symbol occupying that cell at time $i - 1$ was b (clause (6.2)).” The join in (6.1) is over all states $p, q \in Q$, symbols $a \in \Gamma$, and directions $d \in \{L, R\}$ such that $\delta(p, a) = (q, b, d)$; that is, all situations that would cause b to be printed.

For $1 \leq i \leq n^c$, $1 \leq j \leq n^c - 1$ (that is, ignoring the left and right boundaries), and $q \in Q$, we include the assignment

$$Q_{ij}^q := \bigvee_{\delta(p,a)=(q,b,R)} (Q_{i-1,j-1}^p \wedge P_{i-1,j-1}^a) \quad (6.3)$$

$$\vee \bigvee_{\delta(p,a)=(q,b,L)} (Q_{i-1,j+1}^p \wedge P_{i-1,j+1}^a). \quad (6.4)$$

Intuitively, this says, “The machine is scanning tape cell j at time i if and only if either it was scanning tape cell $j - 1$ at time $i - 1$ and moved right (clause (6.3)), or it was scanning tape cell $j + 1$ at time $i - 1$ and moved left (clause (6.4)).” The join in (6.3) is over all states $p \in Q$ and symbols

$a, b \in \Gamma$ such that $\delta(p, a) = (q, b, R)$; that is, all situations that would cause the machine to move right and enter state q .

For $j = 0$, that is, for the leftmost tape cell, we define Q_{ij}^q in terms of (6.3) only. Similarly, for $j = n^c$, we define Q_{ij}^q in terms of (6.4) only.

This takes care of everything except the first row of the matrix. The values P_{ij}^b and Q_{ij}^q are determined by the start configuration; these are the inputs to the circuit. If $x = a_1 \cdots a_n$, the start state is s , and the endmarker and blank symbol are \vdash and \sqcup , respectively, we include

$$\begin{aligned}
P_{0,0}^{\vdash} &:= 1, \\
P_{0,0}^b &:= 0, \quad b \in \Gamma - \{\vdash\}, \\
P_{0,j}^{a_j} &:= 1, \quad 1 \leq j \leq n, \\
P_{0,j}^b &:= 0, \quad b \in \Gamma - \{a_j\}, \quad 1 \leq j \leq n, \\
P_{0,j}^{\sqcup} &:= 1, \quad n+1 \leq j \leq n^c, \\
P_{0,j}^b &:= 0, \quad b \in \Gamma - \{\sqcup\}, \quad n+1 \leq j \leq n^c, \\
Q_{0,0}^s &:= 1 \\
Q_{0,j}^s &:= 0, \quad 1 \leq j \leq n^c, \\
Q_{0,j}^q &:= 0, \quad 0 \leq j \leq n^c, q \in Q - \{s\}.
\end{aligned}$$

This gives a circuit. Assuming that the machine moves its head all the way to the left before entering its accept state t , the Boolean value of

$$Q_{n^c,0}^t \vee Q_{n^c,1}^t$$

determines whether M accepts x .

The construction we have just given can be done in logspace. Even though the circuit is polynomial size, it is highly uniform in the sense that it is built of many identical pieces. The only differences are the indices i, j , which can be written down in logspace. \square

The Cook–Levin Theorem

Now we show how to derive the Cook–Levin theorem as a corollary of the previous construction. We would like to show that the Boolean satisfiability problem SAT—given a Boolean formula, does it have a satisfying truth assignment?—is *NP*-complete. The two main differences between SAT and CVP are:

- (i) With SAT, the input values are not provided. The problem asks whether there *exist* values making the formula evaluate to true.

- (ii) CVP is defined in terms of circuits and SAT is defined in terms of formulas. The difference is that in circuits, Boolean values may be used more than once. A circuit is represented as a labeled directed acyclic graph (dag), whereas a formula is a labeled tree. Another way to look at it is that a circuit allows sharing of common subexpressions. The satisfiability problem is *NP*-complete, regardless of whether we use circuits or formulas; but the problem of evaluating a formula on a given truth assignment is apparently easier than CVP, since it can be done in logspace (Homework 2, Exercise 3).

We define a *circuit with unspecified inputs* exactly as above, except that we also include assignments

$$P_i := ?$$

denoting inputs whose value is unspecified.

Theorem 6.2 *Boolean satisfiability is \leq_m^{\log} -complete for NP.*

Proof. Boolean satisfiability is in *NP*, since we can guess a truth assignment and verify that it satisfies the given formula or circuit in polynomial time.

To show that the problem is \leq_m^{\log} -hard for *NP*, let A be an arbitrary set in *NP*, and let M be a nondeterministic machine accepting A and running in time n^c . Assume without loss of generality that the nondeterminism is binary branching. Then a computation path of M is specified by a string in $\{0, 1\}^{n^c}$.

Let M' be a deterministic machine that takes as input $x\#y$, where $|y| = |x|^c$, and runs M on input x , using y to resolve the nondeterministic choices and accepting if the computation path of M specified by y leads to acceptance. By the construction of Theorem 6.1, there is a circuit that has value 1 iff M' accepts $x\#y$. Note from the construction that if $|z| = |y| = n^c$, the circuit constructed for $x\#z$ is identical to that for $x\#y$ except for the inputs corresponding to y ; making these inputs unspecified, we obtain a circuit $C(P_1, \dots, P_{n^c})$ with unspecified inputs P_1, \dots, P_{n^c} such that M accepts x if and only if there exist $y_1, \dots, y_{n^c} \in \{0, 1\}$ such that $C(y_1, \dots, y_{n^c}) = 1$.

We can transform the circuit into a formula by replacing each assignment $P_i := E$ with the clause $P_i \leftrightarrow E$ and taking the conjunction of all clauses obtained in this way. The resulting formula is satisfiable iff M accepts x . \square

The Boolean satisfiability problem remains *NP*-hard even when restricted to formulas in conjunctive normal form (CNF with at most three literals per clause (3CNF) (Miscellaneous Exercise 10), whereas it is solvable in polynomial time for formulas in 2CNF (Homework 2, Exercise 2). The satisfiability problem for 3CNF formulas is known as 3SAT.