The Cook-Levin Theorem via Boolean Circuits

1 Circuits

We refer to Sipser (Section 9.3) for the definition of *Boolean circuit*. In general, a Boolean circuit is a directed acyclic graph with n sources (for inputs) and 1 sink (for the output), where each node is a "logic gate" of indegree (at most) two, computing some Boolean function on (at most) two inputs. The entire circuit computes a function $f : \{0,1\}^n \to \{0,1\}$.

Lemma 1 *Let $f : \{0,1\}^n \to \{0,1\}$ be an arbitrary Boolean function. Then there is a circuit of size $O(2^n)$ that computes f.*

Proof: We give a recursive construction of the circuit. If $n = 1$, then either $f(x) = x$, in which case it is computed by a circuit of zero gates, or $f(x) = \neg x$, which can be computed by a circuit of size one, or $f(x) = 0 = (x \wedge \neg x)$, which is computed by a circuit of size two, or $f(x) = 1 = (x \vee \neg x)$, which is also computed by a circuit of size 2.

Let now f be an arbitrary function of n variables. We can write it as

$$f(x_1, \ldots, x_n) = (x_n \wedge f(x_1, \ldots, x_{n-1}, 1)) \vee (\neg x_n \wedge f(x_1, \ldots, x_{n-1}, 0))$$

Where both $f(x_1, \ldots, x_{n-1}, 1)$ and $f(x_1, \ldots, x_{n-1}, 0)$ are functions of $n-1$ variables, that can be recursively realized by a circuit.

The size $S(n)$ of the circuit constructed this way satisfies the recursion $S(1) \leq 2$, $S(n) \leq 4 + 2S(n-1)$, which solves to $S(n) \leq 3 \cdot 2^n - 4$. $\Box$

Boolean circuits are a great computational model for computing *finite* functions (i.e. finite languages) over $\{0,1\}^n$ for some n. A key fact about Boolean circuits is that they can simulate time-bounded Turing machines in an efficient way, as shown in the following theorem.

Theorem 2 *Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$ be a Turing machine that, on inputs of length n, runs in time at most t. Then there is a circuit of size $O((|\Gamma| \cdot |Q|)^3 \cdot t^2)$ that, given an input x of length n, outputs 1 if and only if M accepts x.*

Proof: We first construct a circuit C_{next} that, given a configuration of M that uses $\leq t$ cells of tape, computes the configuration at the following step. A final configuration is left unchanged.

Let M have tape alphabet Γ and set of states Q. We represent a configuration by using t blocks of bits. The i-th block of bits contains the alphabet element of the i-th cell of the tape (represented as a sequence of $\lceil \log |\Gamma| \rceil$ bits), a bit that says whether the head of the machine is over the i-th cell of the tape, and, if so, the current state of the machine, represented as a sequence of $\lceil \log |Q| \rceil$ bits.

Each block, therefore, is $1 + \lceil \log |\Gamma| \rceil + \lceil \log |Q| \rceil$ bits long. Let us call this number B. (Note that, for a fixed machine M, B is a constant.)

We want to build a circuit C_{next} that, given ct bits in input representing a configuration, produces ct bits in output representing the next-step configuration.[1] It suffices to observe that every bit of the output depends on only $\leq 3B$ bits of the input, and so each output bit of the circuit can be computed using $O(2^{3B})$ gates, so that the entire circuit has size $O(t \cdot 2^{3B})$. To justify the previous observation, let c be an input configuration for the circuit and c' be the desired output. The portion of c' corresponding to the i-th cell of the tape depends only on the portion of c corresponding to the $(i-1)$-th, i-th, and $(i+1)$-th cells of the tape; in one step, the content of no other cell can have any effect on the i-th cell. In total, these three cells are described by $3B$ bits, including a description of where the head of the machine is and what is the state.

Let us now construct a circuit C_t by layering t copies of circuit C_{next} one on top of the other, that is, with the outputs of the i-th copy fed as inputs of the $(i+1)$-th copy. Clearly, C_t has size $O(t^2 \cdot 2^{3c})$ and, given a configuration c, $C_t(c)$ computes the configuration reached by M starting from c in t steps. We can modify it into a circuit C'_t of size $O(t^2 \cdot 2^{3c})$ that has only one output and such that $C'_t(c) = 1$ if and only if M reaches an accepting configuration starting from c and running for at most t steps.

Finally, let us hard-wire into C'_t that the head is in the first cell, that the state is q_0, and that all the cells except the first n contain a blank symbol, and let us call C the resulting circuit. Now, on input x, $C(x) = 1$ if and only if M accepts x in at most t steps.[2] $\square$

2 Satisfiability Problems

Definition 3 (Circuit-SAT) *Define the Circuit Satisfiability (Circuit-SAT) problem as follows: given a circuit C the question is whether there is an input x such that $C(x) = 1$.*

Using Theorem 2, it is easy to prove that Circuit Satisfiability is NP-complete.

Theorem 4 *Circuit-SAT is NP-complete.*

Proof: First, we argue that Circuit-SAT is in NP: given a circuit C, a short proof that C is in the language is an input x such that $C(x) = 1$. Note that such an x is no longer than the length of the description if the circuit C, and its validity can be checked in polynomial time by evaluating C on x.

We now wish to show that Circuit-SAT is NP-hard. Let L be a problem in NP. By our characterization of NP in terms of polynomial-time verifiers, there is a polynomial time algorithm $V(\cdot, \cdot)$

[1]There is one more detail to take care of: what happens if the input is a configuration c that uses t cells of tape and the next-step configuration c' uses $t+1$ cells of tape? In this case, we will let the circuit output only the content of the first t cells of the tape of c'.

[2]There is one final detail: C_t and C'_t expect in input a configuration, which is a sequence of triples (b, q, a) where b is a bit that tells whether the head is on that cell of the tape, q tells, if $b = 1$, what is the state of the machine, and a is the *tape alphabet* element on that cell of the tape. After we hard-wire the values of b and q, we still cannot let a be the input of the circuit, because each a is a sequence of $\lceil \log_2 |\Gamma| \rceil$ bits designed to represent an element of Γ, while we want our final circuit to have only one input bit per cell of the tape. We can solve this problem by assuming that Γ is represented in binary so that 0 is mapped into $0 \cdots 00$ and 1 is mapped into $0 \cdots 01$, then we just have to hardwire zeroes into all the input bits corresponding to an alphabet element except for the last bit in each cell.

and a polynomial $p(\cdot)$ such that

$$x \in L \text{ if and only if there exists a } w \text{ such that } |w| \leq p(|x|) \text{ and } V(x, w) = 1.$$

Our polynomial-time reduction from L to Circuit-SAT works as follows. Given an input x of length n, first construct a circuit C such that for every z of length n and every w of length $\leq p(n)$ we have $V(z, y) = C(z, y)$. Since V runs in polynomial time, the circuit C has size polynomial in n and can be constructed in time polynomial in n, by applying Theorem 2.

Next, we "hard wire" the given input x into the z-input of C, obtaining a new circuit C_x such that for every w of length $\leq p(n)$ we have $C_x(w) = C(x, w) = V(x, w)$. Our reduction then outputs the circuit C_x.

In summary, the polynomial-time reduction takes an input x and outputs a circuit C_x. By construction, C_x is in Circuit-SAT if and only if there is a w such that $C_x(w) = V(x, w) = 1$, which happens if and only if x is in L. $\square$

Next we define the problem 3SAT. In 3SAT, an input is a Boolean formula in 3-Conjunctive-Normal-Form (3CNF). A 3CNF formula is a AND-of-ORs, with each OR being over precisely three distinct variables. Variables are allowed to be completed.

Definition 5 (3SAT) *The 3SAT problem is: given a 3CNF formula ϕ, is there an assignment of values to the variables that satisfies ϕ?*

It is easy to see that 3SAT is in NP.

Theorem 6 *Circuit-SAT $\leq_m^p$ 3SAT. Therefore 3SAT is NP-hard, and so NP-complete.*

This is Theorem 9.27 in Sipser's book.