Notes for Lecture 3

In this lecture we introduce the computational model of boolean circuits and prove that polynomial size circuits can simulate all polynomial time computations, and we show how to use this result to prove that 3SAT is NP-complete.

1 Circuits

A circuit C has n inputs, m outputs, and is constructed with AND gates, OR gates and NOT gates. Each gate has in-degree 2 except the NOT gate which has in-degree 1. The out-degree can be any number. A circuit must have no cycle. See Figure 1.

A circuit C with n inputs and m outputs computes a function $f_C : \{0,1\}^n \to \{0,1\}^m$. See Figure 2 for an example.

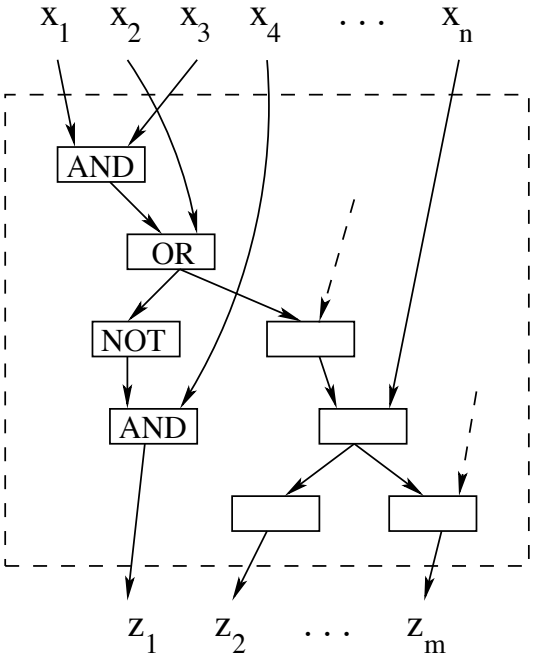

Figure 1: A Boolean circuit.

Define **SIZE**(C) = # of AND and OR gates of C. By convention, we do *not* count the NOT gates.

To be compatible with other complexity classes, we need to extend the model to arbitrary input sizes:

Definition 1 *A language L is solved by a family of circuits $\{C_1, C_2, \ldots, C_n, \ldots\}$ if for every $n \geq 1$ and for every x s.t. $|x| = n$,*

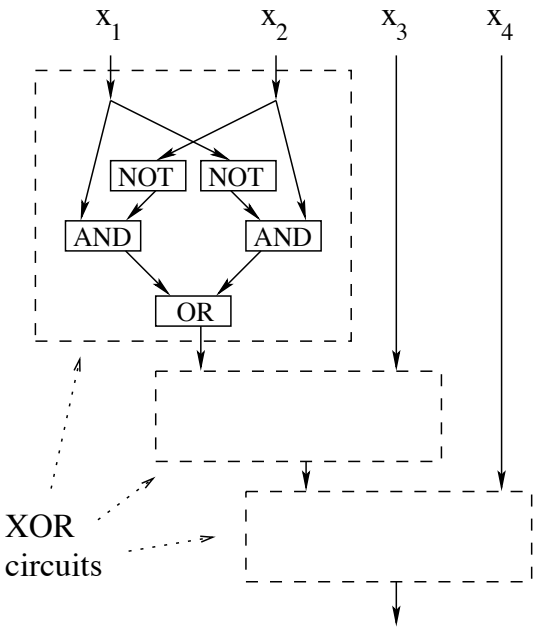

Figure 2: A circuit computing the boolean function $f_C(x_1x_2x_3x_4) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$.

$$x \in L \iff f_{C_n}(x) = 1.$$

Definition 2 *Say $L \in$ **SIZE**$(s(n))$ if L is decided by a family $\{C_1, C_2, \ldots, C_n, \ldots\}$ of circuits, where C_i has at most $s(i)$ gates.*

2 Relation to other complexity classes

Unlike other complexity measures, like time and space, for which there are languages of arbitrarily high complexity, the size complexity of a problem is always at most exponential.

Theorem 3 *For every language L, $L \in$ **SIZE**$(O(2^n))$.*

PROOF: We need to show that for every 1-output function $f : \{0,1\}^n \to \{0,1\}$, f has circuit size $O(2^n)$.

Use the identity $f(x_1x_2 \ldots x_n) = (x_1 \wedge f(1x_2 \ldots x_n)) \vee (\overline{x}_1 \wedge f(0x_2 \ldots x_n))$ to recursively construct a circuit for f, as shown in Figure 3.

The recurrence relation for the size of the circuit is: $s(n) = 3 + 2s(n-1)$ with base case $s(1) = 1$, which solves to $s(n) = 2 \cdot 2^n - 3 = O(2^n)$. □

The exponential bound is nearly tight.

Theorem 4 *There are languages L such that $L \notin$ **SIZE**$(o(2^n/n))$. In particular, for every $n \geq 11$, there exists $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit of size $2^n/4n$.*

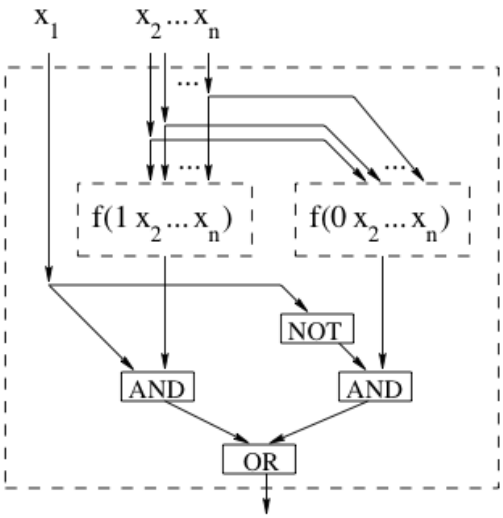

Figure 3: A circuit computing any function $f(x_1 x_2 \ldots x_n)$ of n variables assuming circuits for two functions of $n-1$ variables.

PROOF: This is a counting argument. There are 2^{2^n} functions $f : \{0,1\}^n \to \{0,1\}$, and we claim that the number of circuits of size s is at most $2^{O(s \log s)}$, assuming $s \geq n$. To bound the number of circuits of size s we create a compact binary encoding of such circuits. Identify gates with numbers $1, \ldots, s$. For each gate, specify where the two inputs are coming from, whether they are complemented, and the type of gate. The total number of bits required to represent the circuit is

$$s \times (2\log(n+s)+3) \leq s \cdot (2\log 2s + 3) = s \cdot (2\log 2s + 5).$$

So the number of circuits of size s is at most $2^{2s \log s + 5s}$, and this is not sufficient to compute all possible functions if

$$2^{2s \log s + 5s} < 2^{2^n}.$$

This is satisfied if $s \leq \frac{2^n}{4n}$ and $n \geq 11$. □

The following result shows that efficient computations can be simulated by small circuits.

Theorem 5 *If $L \in$ **DTIME**$(t(n))$, then $L \in$ **SIZE**$(O(t^2(n)))$.*

PROOF: Let L be a decision problem solved by a machine M in time $t(n)$. Fix n and x s.t. $|x| = n$, and consider the $t(n) \times t(n)$ tableau of the computation of $M(x)$. See Figure 4.

Assume that each entry (a, q) of the tableau is encoded using k bits. By Proposition 3, the transition function $\{0,1\}^{3k} \to \{0,1\}^k$ used by the machine can be implemented by a "next state circuit" of size $k \cdot O(2^{3k})$, which is exponential in k but constant in n. This building block can be used to create a circuit of size $O(t^2(n))$ that computes the complete tableau, thus also computes the answer to the decision problem.This is shown in Figure 5.

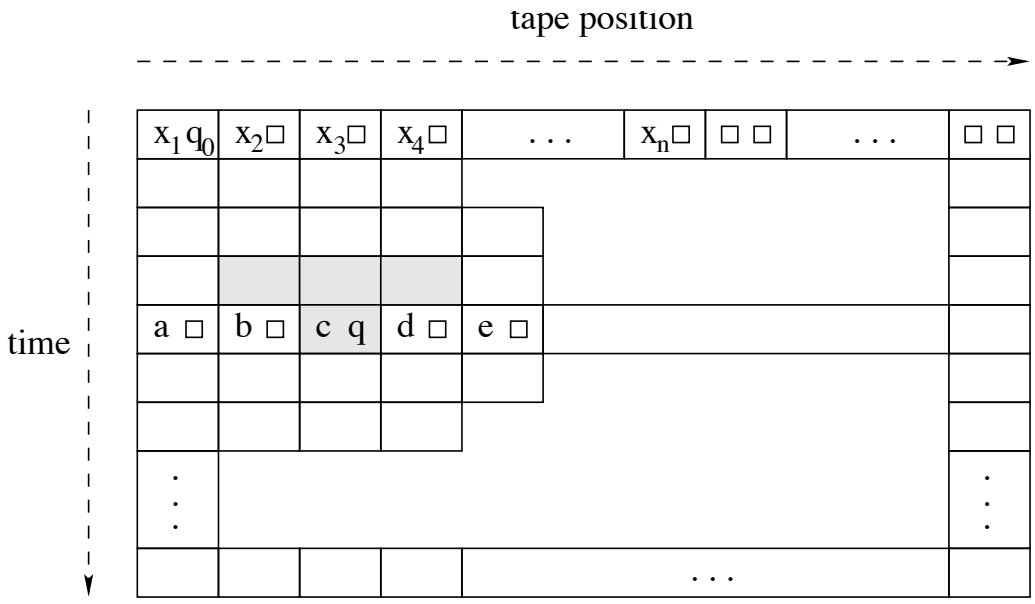

Figure 4: $t(n) \times t(n)$ tableau of computation. The left entry of each cell is the tape symbol at that position and time. The right entry is the machine state or a blank symbol, depending on the position of the machine head.

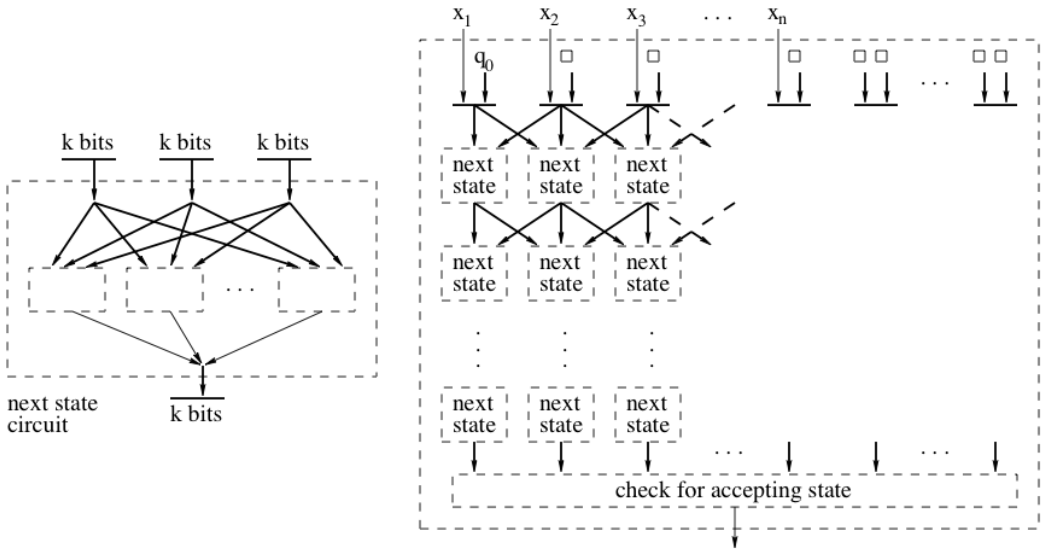

Figure 5: Circuit to simulate a Turing machine computation by constructing the tableau.

□

Corollary 6 $\mathbf{P} \subseteq \mathbf{SIZE}(n^{O(1)})$.

On the other hand, it's easy to show that $\mathbf{P} \neq \mathbf{SIZE}(n^{O(1)})$, and, in fact, one can define languages in $\mathbf{SIZE}(O(1))$ that are undecidable.

An equivalent characterization of languages decidable by polynomial size circuits can be given using the notion of *advice*.

Definition 7 *A language L can be decided in time $t(n)$ and advice $a(n)$ if there is an algorithm $A(\cdot,\cdot)$ runnint in time $\leq t(n)$ on inputs of length n, such that for every input length n there exists an "advice" string s_n of length $\leq a(n)$ such that for every x of length n*

$$x \in L \Leftrightarrow A(x, s_n) \text{ accepts} \quad .$$

We denote by $\mathbf{P}/poly$ the class of languages that can be decided in polynomial time using advice of polynomial length.

Theorem 8 $\mathbf{P}/poly = \mathbf{SIZE}(n^{O(1)})$.

PROOF: Suppose that $L \in \mathbf{SIZE}(n^{O(1)})$ and consider the circuit evaluation algorithm A that on input a string x and a circuit C outputs $C(x)$. Clearly A is a polynomial time algorithm, and it witnesses $L \in \mathbf{P}/poly$, by using a minimal-size circuit for $L \cap \{0,1\}^n$ as the advice string for inputs of length n.

Suppose that $L \in \mathbf{P}/poly$, and let A be the advice algorithm. Then, for every input length n, we can construct a circuit C of size $n^{O(1)}$ such that, for the appropriate advice string s_n, we have $C(x, s_n) = 1$ iff $x \in L$. Hard-wire the string s_n into the circuit. □

3 Circuit SAT and 3SAT

In the Circuit-SAT problem, we are given a boolean circuit C and we want to determine if there exists an input x such that $C(x) = 1$. This is clearly a problem in NP, and now we are going to argue that it is NP-complete.

Let L be a problem in **NP**, R be an **NP** relation showing that $L \in \mathbf{NP}$. That is, there is a deterministic machine M such that M determines whether a given pair (x, y) belong to R in time at most $p(|x| + |y|)$, where $p()$ is a polynomial, and $(x, y) \in R$ implies that $|y| \leq q(|x|)$, where $q()$ is also a polynomial. Furthermore, $x \in L$ if and only if there is a y such that $(x, y) \in R$.

Here is the reduction from L to Circuit-SAT: given an input x of length L, we construct the circuit C of size $O((p(n + q(n)))^2)$ that given a string z of length n and a string y of length at most $q(n)$ determines whether M accepts the pair (z, y). Then we construct the circuit C_x such that $C_x(y) = C(x, y)$ by hard-wiring x into the first n bits of C. The circuit C_x is the output of the reduction, and clearly x is in L if and only C_x is satisfiable.

3SAT is the problem in which the input is a boolean formula in 3CNF (3-conjunctive-normal-form). A 3CNF formula is an AND-of-ORs boolean formula in which each OR

involves at most three variables; each variable may or may not be complemented. For example, the following is a 3CNF:

$$(x_1 \vee x_2 \vee \neg x_4) \wedge (x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4)$$

A boolean formula is *satisfiable* if there is an assignment of boolean values to the variables that makes the formula true. The above boolean formula is true, as witnessed, for example, by the assignment that sets x_1 to true and all other variables to false. In the 3SAT problem, given a 3CNF formula we want to know if it is satisfiable.

To see that circuit-SAT reduces to 3SAT, given a circuit C with s gates and inputs $x_1, \ldots, x_n$, define new variables $y_1, \ldots, y_s$, one new variable for each gate of C, and number them so that y_s corresponds to the output gate. Now, for every gate, write down the 3CNF formula that states that variable corresponding to the gate has values consistent to the values of the variables corresponding to the input. For example, if gate 9 is an AND gate that takes inputs from x_5 and from the seventh gate, we write the 3CNF corresponding to the $y_9 = x_5 \wedge y_7$, which is

$$(y_9 \vee \neg x_5 \vee \neg y_7) \wedge (\neg y_9 \vee x_5 \vee y_7) \wedge (\neg y_9 \vee \neg x_5 \vee y_7) \wedge (\neg y_9 \vee x_5 \vee \neg y_7)$$

We take the AND of all these expression, and of the term (y_s). It is easy to see that the resulting 3CNF can be satisfied if and only if there is an x such that $C(x) = 1$.