

# Circuits

Slides based on *S.Aurora, B.Barak. Complexity Theory: A Modern Approach.*

Ahto Buldas

Ahto.Buldas@ut.ee



*“One might imagine that  $P \neq NP$ , but SAT is tractable in the following sense: for every  $\ell$  there is a very short program that runs in time  $\ell^2$  and correctly treats all instances of size  $\ell$ . ”*

Karp and Lipton, 1982

We study a model of computation called a *Boolean circuit*, which is a generalization of Boolean formulae and a rough formalization of a silicon chip.

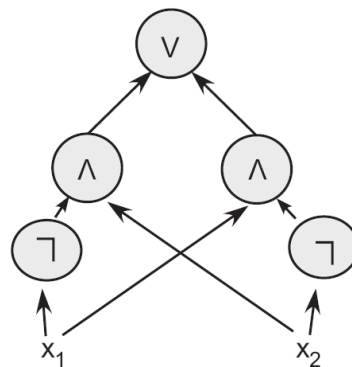
Circuit is a natural model for *non-uniform computation*, by which we mean that a different algorithm is allowed for each input size.

One can separate complexity classes such as  $P$  and  $NP$  by proving lower bounds on circuit size. We outline why such lower bounds ought to exist.

We define the class  $P/poly$  of languages computable by polynomial-sized boolean circuits and explore its relation to  $NP$ .

## Boolean circuits

**Def.:** For every  $n, m \in \mathbb{N}$  a **Boolean circuit**  $C$  with  $n$  inputs and  $m$  outputs is a DAG, that contains  $n$  **input nodes** with no incoming edges and  $m$  **output nodes** with no outgoing edges. All other nodes are called **gates** and are labeled with OR, AND, and NOT. Gates have 2 or 1 incoming nodes. The **size**  $|C|$  of  $C$  is the number of nodes.  $C$  is called a **Boolean formula** if each node has at most one outgoing edge. For example, a circuit computing the XOR function:



## Circuits and Boolean Functions

Every circuit implements a function  $\{0, 1\}^n \rightarrow \{0, 1\}^m$ . If each  $n$ -bit input is assigned a value in  $\{0, 1\}$ , we can compute the  $m$  output values.

For every string  $u \in \{0, 1\}^n$ , we denote by  $C(u)$  the output of the circuit  $C$  on input  $u$ .

The Boolean operations OR, AND, and NOT form a *universal basis*, by which we mean that every function  $\{0, 1\}^n \rightarrow \{0, 1\}^m$  can be implemented by a boolean circuit (and boolean formula).

## The Class P/poly

**Def:** Let  $T: \mathbb{N} \rightarrow \mathbb{N}$  be a function. A  $T(n)$ -sized circuit family is a sequence  $\{C_n\}_{n \in \mathbb{N}}$  of Boolean circuits, where  $C_n$  has  $n$  inputs and a single output, such that  $|C_n| \leq T(n)$  for every  $n$ .

We say that  $L \in \mathbf{SIZE}(T(n))$  if there exists a  $T(n)$ -size circuit family  $\{C_n\}_{n \in \mathbb{N}}$  such that for every  $x \in \{0, 1\}^n$ :  $x \in L \Leftrightarrow C(x) = 1$ .

We already know that every language is decidable by a circuit family of size  $O(n \cdot 2^n)$ , since the circuit for input length  $n$  could contain  $2^n$  hardwired bits indicating which inputs are in the language.

**P/poly** is the class of languages that are decidable by polynomial-sized circuit families, in other words:

$$\mathbf{P/poly} = \cup_c \mathbf{SIZE}(n^c) .$$

## Example: Unary Languages

A language  $L$  is *unary* if it is a subset of  $\{1^n : n \in \mathbb{N}\}$ . Every unary language has linear size circuits since the circuit for an input size  $n$  only needs to have a single hardwired bit indicating whether or not  $1^n$  is in the language. Hence the following unary language has linear size circuits, even though it is undecidable:

$$\{1^n : M_n \text{ outputs } 1 \text{ on input } 1^n\} .$$

where  $M_n$  is the machine represented by the (modified) binary expansion of the number  $n$ .

## Log-space Uniform Circuit Families

This example suggests that it may be fruitful to consider the restriction to circuits that can actually be built, say using a fairly efficient Turing machine. It will be most useful to formalize this using log-space computations.

Recall that a function  $f: \{0, 1\}^* \rightarrow \{0, 1\}^*$  is implicitly log-space computable if the mapping  $\langle x, i \rangle \mapsto f(x)_i$  can be computed in logarithmic space.

***Def. (log-space-uniform circuit families):*** A circuit family  $\{C_n\}$  is log-space uniform if there is an implicitly log-space computable function mapping  $1^n$  to the description of the circuit  $C_n$ .

## Circuits as Binary Strings

We need to fix some representation of the circuits as strings. We will assume that the circuit of size  $N$  is represented by its  $N \times N$  adjacency matrix and in addition an array of size  $N$  that gives the labels (gate type and i/o) of each node. This means that  $C_n$  is log-space uniform if and only if the following functions are computable in  $O(\log n)$  space:

- $\text{SIZE}(n)$  returns the size  $m$  (in binary) of the circuit  $C_n$ .
- $\text{TYPE}(n, i)$ , where  $i \in \{1 \dots m\}$ , returns the label and type of the  $i$ -th node.
- $\text{EDGE}(n, i, j)$  returns 1 if there is a directed edge in  $C_n$  between the  $i$ -th node and the  $j$ -th node.

## $\mathbf{P} \subset \mathbf{P}/\text{poly}$

Note that inputs and outputs of these functions can be encoded using  $O(\log |C_n|)$  bits. The requirement that they run in  $O(\log n)$  space means that we require that  $\log |C_n| = O(\log n)$  or in other words that  $C_n$  is of size at most polynomial in  $n$ .

**Theorem:** A language has log-space-uniform circuits of polynomial size iff it is in  $\mathbf{P}$ .

**Proof sketch:** The only if part is trivial—log-space computations must be finished in polynomial time. The if part follows the proof of the Cook-Levin Theorem. Recall that we can simulate every time  $O(T(n))$  TM  $M$  by an oblivious TM  $\tilde{M}$  running in time  $O(T(n)^2)$  (or even  $O(T(n) \log T(n))$  if we are more careful). In fact, we can ensure that the movement of the

oblivious TM  $\tilde{M}$  do not even depend on the contents of its work tape, and so, by simulating  $\tilde{M}$  while ignoring its read/write instructions, we can compute in  $O(\log T(n))$  space for every  $i$  the position its heads will be at the  $i$ -th step.

Given this insight, it is fairly straightforward to translate the proof of Cook-Levin theorem to prove that every language in  $\mathbf{P}$  has a log-space-uniform circuit family. The idea is that if  $L \in \mathbf{P}$  then it is decided by an oblivious TM  $\tilde{M}$  of the form above. We will use that to construct a log-space uniform circuit family  $\{C_n\}_{n \in \mathbb{N}}$  such that for every  $x \in \{0, 1\}^n$ :  $C_n(x) = \tilde{M}(x)$ .

Recall that the transcript of  $\tilde{M}$ 's execution on input  $x$  is the sequence  $z_1, \dots, z_T$  of snapshots (machines state and symbols read by all heads) of the execution at each step in time. Assume that each such  $z_i$  is encoded by a string (that needs only to be of constant size). We can compute the

string  $z_i$  based the previous snapshots  $z_{i-1}$  and  $z_{i_1}, \dots, z_{i_k}$  where  $z_{i_j}$  denote the last step that  $\tilde{M}'_s$   $j$ -th head was in the same position as it is in the  $i$ -th step. Because these are only a constant number of strings of constant length, we can compute  $z_i$  from these previous snapshot using a constant-sized circuit. Also note that, under our assumption above, given the indices  $i$  and  $i' < i$  we can easily check whether  $z_i$  depends on  $z_{i'}$ .

The composition of all these constant-sized circuits gives rise to a circuit that computes from the input  $x$ , the snapshot  $z_T$  of the last step of  $\tilde{M}$ 's execution on  $x$ . There is a simple constant-sized circuit that, given  $z_T$  outputs 1 if and only if  $z_T$  is an accepting snapshot (in which  $\tilde{M}$  outputs 1 and halts). Thus, we get a circuit  $C$  such that  $C(x) = \tilde{M}(x)$  for every  $x \in \{0, 1\}^n$ . Because our circuit  $C$  is composed of many small (constant-sized) circuits, and determining which small circuit is applied to which nodes can be done in logarithmic space, it is not hard to see that we

can find out every individual bit of  $C$ 's representation in logarithmic space. (In fact, one can show that the functions SIZE, TYPE and EDGE above can be computed using only logarithmic space and poly-logarithmic time.)

## Turing Machines that Take Advice

There is a way to define  $\mathbf{P}/\text{poly}$  using Turing machines that take *advice*:

**Def.:** Let  $T, a: \mathbb{N} \rightarrow \mathbb{N}$  be functions. The class of languages decidable by time- $T(n)$  TM's with  $a(n)$  advice, denoted  $\mathbf{DTIME}(T(n))/a(n)$ , contains every  $L$  such that there exists a sequence  $\{\alpha_n\}_{n \in \mathbb{N}}$  of strings with  $\alpha \in \{0, 1\}^{a(n)}$  and a TM  $M$  satisfying  $M(x, \alpha_n) = 1 \Leftrightarrow x \in L$  for every  $x \in \{0, 1\}^n$ , where on input  $(x, \alpha_n)$  the machine  $M$  runs for at most  $O(T(n))$  steps.

**Example:** Every unary language can be decided by a polynomial time Turing machine with 1 bit of advice. The advice string for inputs of length  $n$  is the single bit indicating whether or not  $1^n$  is in the language.

## New Definition for P/poly

**Theorem**  $\mathbf{P/poly} = \cup_{c,d \geq 1} \mathbf{DTIME}(n^c)/n^d$ .

**Proof:** If  $L \in \mathbf{P/poly}$ , we use the poly-sized description of its circuit family  $\{C_n\}$  as advice to a TM. On input of size  $n$ , the TM just simulates  $C_n$ .

Conversely, if  $L$  is decidable by a poly-time  $M$  with advices  $\{\alpha_n\}_{n \in \mathbb{N}}$  of size  $a(n)$  for some polynomial  $a$ , then we can construct for every  $n$ , a polynomial-sized circuit  $D_n$  such that on every  $x \in \{0, 1\}^n$  and  $\alpha \in \{0, 1\}^{a(n)}$ , we have  $D_n(x, \alpha) = M(x, \alpha)$ . We let the circuit  $C_n$  be the polynomial circuit that maps  $x$  to  $D_n(x, \alpha_n)$ . That is,  $C_n$  is equal to the circuit  $D_n$  with the string  $\alpha_n$  hardwired as its second input.

**Remark:** By hardwiring an input into a circuit we mean taking a circuit  $C$  with two inputs  $x \in \{0, 1\}^n$ ,  $y \in \{0, 1\}^m$  and transforming it into the circuit  $C_y$  that for every  $x$  returns  $C(x, y)$ . It is easy to do so while ensuring that the size of  $C_y$  is not greater than the size of  $C$ .

## Karp-Lipton Theorem

Karp and Lipton formalized the question of whether or not SAT has small circuits as: Is SAT in  $P/poly$ ? They showed that the answer is NO if  $PH$  does not collapse:

*Theorem (Karp-Lipton):* If  $NP \subseteq P/poly$  then  $PH = \Sigma_2^p$ .

*Proof:* To show that  $PH = \Sigma_2^p$  it is enough to show that  $\Pi_2^p \subseteq \Sigma_2^p$  and in particular it suffices to show that  $\Sigma_2^p$  contains the  $\Pi_2^p$ -complete language  $\Pi_2SAT$  consisting of all true formulae of the form:

$$\forall u \in \{0, 1\}^n \exists v \in \{0, 1\}^n \varphi(u, v) = 1 . \quad (1)$$

where  $\varphi$  is an un-quantified Boolean formula. If  $NP \subseteq P/poly$  then there is a polynomial  $p$  and a  $p(n)$ -sized circuit family  $\{C_n\}_{n \in \mathbb{N}}$  such that for every Boolean formula  $\varphi$  and  $u \in \{0, 1\}^n$ :  $C_n(\varphi, u) = 1$  if and only

if there exists  $v \in \{0, 1\}^n$  such that  $\varphi(u, v) = 1$ . Yet, using the self-reducibility of SAT, we actually know that there is a  $q(n)$ -sized circuit family  $\{C'_n\}_{n \in \mathbb{N}}$  such that for every such formula  $\varphi$  and  $u \in \{0, 1\}^n$ , if there is a string  $v \in \{0, 1\}^n$  such that  $\varphi(u, v) = 1$  then  $C'_n(\varphi, u)$  outputs such a string  $v$ . Since  $C'_n$  can be described using  $10q(n)^2$  bits, this implies that if (1) is true then the following quantified formula is also true:

$$\exists w \in \{0, 1\}^{10q^2(n)}, \forall u \in \{0, 1\}^n :$$

$$w \text{ describes a circuit } C' \text{ s.t. } \varphi(u, C'(\varphi, u)) = 1 .(2)$$

Yet if (1) is false then certainly (regardless of whether  $\mathbf{P} = \mathbf{NP}$ ) the formula (2) is false as well, and hence (2) is actually equivalent to (1)! However, since evaluating a circuit on an input can be done in poly-time, evaluating the truth of (2) can be done in  $\Sigma_2^p$ .

## Exponential Version of the Karp-Lipton Theorem

Similarly the following theorem can be proven, though we leave the proof as an exercise.

*Theorem (Exponential Karp-Lipton):* If  $\mathbf{EXP} \subseteq \mathbf{P}/\text{poly}$  then  $\mathbf{EXP} = \Sigma_2^p$ .

Combining the time hierarchy theorem with the previous theorem implies that if  $\mathbf{P} = \mathbf{NP}$  then  $\mathbf{EXP} \not\subseteq \mathbf{P}/\text{poly}$ . Indeed,  $\mathbf{P} = \mathbf{NP}$  would imply  $\Sigma_2^2 = \mathbf{P}$  which under the assumption  $\mathbf{EXP} \subseteq \mathbf{P}/\text{poly}$  leads to  $\mathbf{EXP} = \mathbf{P}$ , which is impossible due to the time hierarchy theorem.

Thus, upper bounds (in this case,  $\mathbf{NP} \subseteq \mathbf{P}$ ) can potentially be used to prove circuit lower bounds, i.e. if  $\mathbf{NP} \subseteq \mathbf{P}$  then there are languages that cannot be decided by polynomial-size circuits.

## Circuit Lower Bounds

Since  $P \subset P/poly$ , if  $NP \not\subseteq P/poly$  then  $P \neq NP$ . The Karp-Lipton theorem gives hope that  $NP \not\subseteq P/poly$ . Can we resolve  $P$  versus  $NP$  by proving  $NP \not\subseteq P/poly$ ?

There is reason to invest hope in this approach as opposed to proving direct lower bounds on Turing machines. By representing computation using circuits we seem to actually peer into the guts of it rather than treating it as a black box. Thus we may be able to *get around the limitations of relativizing methods*.

Sadly, such hopes have not yet come to pass. After two decades, the best circuit size lower bound for an  $NP$  language is only  $5n$ .

On the positive side, we have had notable success in proving lower bounds for more restricted circuit models

## Circuit Complexity of Boolean Functions

It is easy to show that for large enough  $n$ , a majority of boolean functions on  $n$  variables require large circuits.

**Theorem 6.15:** For  $n \geq 100$ , almost all boolean functions on  $n$  variables require circuits of size at least  $2^n / (10n)$ .

**Proof:** We use a simple counting argument. There are at most  $s^{3s}$  circuits of size  $s$ —just count the number of labeled directed graphs, where each node has in-degree at most 2. Hence this is an upper bound on the number of functions on  $n$  variables with circuits of size  $s$ . For  $s = 2^n / (10n)$ , this number is at most  $2^{2n} / 10$ , which is small compared to the number  $2^{2n}$  of boolean functions on  $n$  variables. Hence most Boolean functions do not have such small circuits.

## Non-Uniform Hierarchy Theorem

Boolean circuits also have a hierarchy theorem. That is, larger circuits can compute strictly more functions than smaller ones:

**Theorem 6.17:** For every functions  $T, T' : \mathbb{N} \rightarrow \mathbb{N}$  with  $2^n / (100n) > T'(n) > T(n) > n$  and  $T(n) \log T(n) = o(T'(n))$ :

$$\mathbf{SIZE}(T'(n)) \not\subseteq \mathbf{SIZE}(T(n)) .$$

**Proof:** The diagonalization methods do not seem to work here, but we can use the counting argument. To show the idea, we prove that  $\mathbf{SIZE}(n^2) \not\subseteq \mathbf{SIZE}(n)$ . For every  $\ell$ , there is a function  $f : \{0, 1\}^\ell \rightarrow \{0, 1\}$  that is not computable by  $2^\ell / (10\ell)$ -sized circuits. On the other hand, every function from  $\{0, 1\}^\ell \rightarrow \{0, 1\}$  is computable by a  $2^\ell 10\ell$ -sized circuit.

Therefore, if we set  $\ell = 1.1 \log n$  and let  $g: \{0, 1\}^n \rightarrow \{0, 1\}$  be the function that applies  $f$  on the first  $\ell$  bits of its input, then

$$g \in \mathbf{SIZE}(2^\ell 10\ell) = \mathbf{SIZE}(11n^{1.1} \log n) \subseteq \mathbf{SIZE}(n^2)$$

$$g \notin \mathbf{SIZE}(2^\ell / (10\ell)) = \mathbf{SIZE}(n^{1.1} / (11 \log n)) \supseteq \mathbf{SIZE}(n) .$$