Chapter 6

Circuits

> *"One might imagine that* $\mathbf{P} \neq \mathbf{NP}$*, but* SAT *is tractable in the following sense: for every* ℓ *there is a very short program that runs in time* ℓ^2 *and correctly treats all instances of size* ℓ*. "*
> Karp and Lipton, 1982

This chapter investigates a model of computation called a *Boolean circuit*, which is a generalization of Boolean formulae and a rough formalization of the familiar "silicon chip." Here are some motivations for studying it.

First, it is a natural model for *nonuniform* computation, by which we mean that a different "algorithm" is allowed for each input size. By contrast, our standard model thus far was *uniform computation*: the same Turing Machine (or algorithm) solves the problem for inputs of all (infinitely many) sizes. Nonuniform computation crops up often in complexity theory, and also in the rest of this book.

Second, in principle one can separate complexity classes such as **P** and **NP** by proving *lowerbounds* on circuit size. This chapter outlines why such lowerbounds ought to exist. In the 1980s, researchers felt boolean circuits are mathematically simpler than the Turing Machine, and thus proving circuit lowerbounds may be the right approach to separating complexity classes. Chapter 13 describes the partial successes of this effort and Chapter 23 describes where it is stuck.

This chapter defines the class **P**/poly of languages computable by polynomial-sized boolean circuits and explores its relation to **NP**. We also encounter some interesting subclasses of **P**/poly, including **NC**, which tries to capture computations that can be efficiently performed on *highly parallel* computers. Finally, we show a (yet another) characterization of the polynomial hierarchy, this time using exponential-sized circuits of constant depth.

Complexity Theory: A Modern Approach. © 2006 Sanjeev Arora and Boaz Barak.
References and attributions are still incomplete.
DRAFT

6.1 Boolean circuits

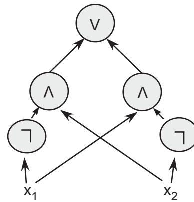

Figure 6.1: A circuit C computing the XOR function (i.e., $C(x_1, x_2) = 1$ iff $x_1 \neq x_2$).

A Boolean circuit is a just a diagram showing how to derive an output from an input by a combination of the basic Boolean operations of OR ($\vee$), AND ($\wedge$) and NOT ($\neg$). For example, Figure 6.1 shows a circuit computing the XOR function. Here is the formal definition.

Definition 6.1 (Boolean circuits)
For every $n, m \in \mathbb{N}$ a *Boolean circuit C with n inputs and m outputs*[1] is a directed acyclic graph. It contains n nodes with no incoming edges; called the *input nodes* and m nodes with no outgoing edges, called the *output nodes*. All other nodes are called *gates* and are labeled with one of $\vee$, $\wedge$ or $\neg$ (in other words, the logical operations OR, AND, and NOT). The $\vee$ and $\wedge$ nodes have fanin (i.e., number of incoming edges) of 2 and the $\neg$ nodes have fanin 1. The *size* of C, denoted by $|C|$, is the number of nodes in it.
The circuit is called a *Boolean formula* if each node has at most one outgoing edge.

The boolean circuit in the above definition implements a function from $\{0,1\}^n$ to $\{0,1\}^m$. This may be clear intuitively to most readers (especially those who have seen circuits in any setting) but here is the proof. Assume that the n input nodes and m output nodes are numbered in some canonical way. Thus each n-bit input can be used to assigned a value in $\{0,1\}$ to each input node. Next, since the graph is acyclic, we can associate an integral *depth* to each node (using breadth-first search, or the so-called *topological sorting* of the graph) such that each node has incoming edges only from nodes of higher depth. Now each node can be assigned a value from $\{0,1\}$ in a unique way as follows. Process the nodes in decreasing order of depth. For each node, examine its incoming edges and the values assigned to the nodes at the other end, and then apply the boolean operation ($\vee, \wedge$, or $\neg$)

Web draft 2006-09-28 18:09

DRAFT

that this node is labeled with on those values. This gives a value to each node; the values assigned to the m output nodes by this process constitute an m-bit output of the circuit.

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

We recall that the Boolean operations OR, AND, and NOT form a *universal basis*, by which we mean that every function from $\{0,1\}^n$ to $\{0,1\}^m$ can be implemented by a boolean circuit (in fact, a boolean formula). See Claim 2.15. Furthermore, the "silicon chip" that we all know about is nothing but[2] an implementation of a boolean circuit using a technology called VLSI. Thus if we have a small circuit for a computational task, we can implement it very efficiently as a silicon chip. Of course, the circuit can only solve problems on inputs of a certain size. Nevertheless, this may not be a big restriction in our finite world. For instance, what if a small circuit *exists* that solves 3SAT instances of up to say $100,000$ variables? If so, one could imagine a government-financed project akin to the Manhattan project that would try to discover such a small circuit, and then implement it as a silicon chip. This could be used in all kinds of commercial products (recall our earlier depiction of a world in which $\mathbf{P} = \mathbf{NP}$) and in particular would jeopardize every encryption scheme that does not use a huge key. This scenario is hinted at in the quote from Karp and Lipton at the start of the chapter.

As usual, we resort to asymptotic analysis to study the complexity of deciding a language by circuits.

Definition 6.2 (Circuit families and language recognition)
Let $T : \mathbb{N} \to \mathbb{N}$ 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 $|C_n| \le T(n)$ for every n.

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

As noted in Claim 2.15, every language is decidable by a circuit family of size $O(n2^n)$, since the circuit for input length n could contain 2^n "hardwired" bits indicating which inputs are in the language. Given an input, the circuit looks up the answer from this table. (The reader may wish to work out an implementation of this circuit.) The following definition formalizes what we can think of as "small" circuits.

[2] Actually, the circuits in silicon chips are not acyclic; in fact the cycles in the circuit are crucial for implementing "memory." However any computation that runs on a silicon chip of size C and finishes in time T can be performed by a boolean circuit of size $O(C\cdot T)$.

Web draft 2006-09-28 18:09

DRAFT

DEFINITION 6.3
P/poly is the class of languages that are decidable by polynomial-sized circuit families, in other words, $\cup_c \mathbf{SIZE}(n^c)$.

Of course, one can make the same kind of objections to the practicality of **P**/poly as for **P**: viz., in what sense is a circuit family of size n^{100} practical, even though it has polynomial size. This was answered to some extent in Section 1.4.1. Another answer is that as complexity theorists we hope (eventually) to show that languages such as SAT are not in **P**/poly. Thus the result will only be stronger if we allow even such large circuits in the definition of **P**/poly.

The class **P**/poly contains **P**. This is a corollary of Theorem 6.7 that we show below. Can we give a reasonable upperbound on the computational power of **P**/poly? Unfortunately not, since it contains even undecidable languages.

EXAMPLE 6.4
Recall that we say that 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:

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

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

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 logspace computations.

Recall that a function $f : \{0,1\}^* \to \{0,1\}^*$ is implicitly logspace computable if the mapping $x, i \mapsto f(x)_i$ can be computed in logarithmic space (see Definition 3.14).

DEFINITION 6.5 (LOGSPACE-UNIFORM CIRCUIT FAMILIES)
A circuit family $\{C_n\}$ is *logspace uniform* if there is an implicitly logspace computable function mapping 1^n to the description of the circuit C_n.

DRAFT

Actually, to make this concrete 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 input/output) of each node. This means that $\{C_n\}$ is logspace uniform if and only if the following functions are computable in $O(\log n)$ space:

- `SIZE`(n) returns the size m (in binary representation) of the circuit C_n.
- `TYPE`(n, i), where $i \in [m]$, returns the label and type of the ith node of C_n. That is it returns one of $\{\vee, \wedge, \neg, NONE\}$ and in addition $\langle OUTPUT, j\rangle$ or $\langle INPUT, j\rangle$ if i is the jth input or output node of C_n.
- `EDGE`(n, i, j) returns 1 if there is a directed edge in C_n between the ith node and the jth node.

Note that both the inputs and the outputs of these functions can be encoded using a logarithmic (in $|C_n|$) number of 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.

Remark 6.6
Exercise 7 asks you to prove that the class of languages decided by such circuits does not change if we use the adjacency list (as opposed to matrix) representation. We will use the matrix representation from now on.

Polynomial circuits that are logspace-uniform correspond to a familiar complexity class:

Theorem 6.7
A language has logspace-uniform circuits of polynomial size iff it is in $\mathbf{P}$.

Remark 6.8
Note that this implies that $\mathbf{P} \subseteq \mathbf{P}/\text{poly}$.

Proof sketch: The only if part is trivial. The if part follows the proof of the Cook-Levin Theorem (Theorem 2.11). Recall that we can simulate every time $O(T(n))$ TM M by an *oblivious* TM $\tilde{M}$ (whose head movement is independent of its input) 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

Web draft 2006-09-28 18:09
DRAFT

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 ith step.[3]

Given this insight, it is fairly straightforward to translate the proof of Theorem 2.11 to prove that every language in **P** has a logspace-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 logspace 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, as we saw in that proof, the *transcript* of $\tilde{M}$'s execution on input x is the sequence $z_1, \ldots, z_T$ of *snapshots* (machine's 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 $z_{i_1}, \ldots, z_{i_k}$ 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 ith 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 polylogarithmic time.) ■

6.1.1 Turing machines that take advice

There is a way to define **P**/poly using Turing machines that "take advice."

Definition 6.9
Let $T, a : \mathbb{N} \to \mathbb{N}$ be functions. The class of *languages decidable by time-$T(n)$*

[3] In fact, typically the movement pattern is simple enough (for example a sequence of $T(n)$ left to right and back sweeps of the tape) that for every i we can compute this information using only $O(\log T(n))$ space and polylog$T(n)$ time.

DRAFT

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 $\alpha_n \in \{0,1\}^{a(n)}$ 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, α_n) the machine M runs for at most $O(T(n))$ steps.

Example 6.10
Every unary language can be 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. In particular this is true of the language of Example 6.4.

This is an example of a more general phenomenon described in the next theorem.

Theorem 6.11
$\mathbf{P}/\mathrm{poly} = \cup_{c,d}\mathbf{DTIME}(n^c)/n^d$

Proof: If $L \in \mathbf{P}/\mathrm{poly}$, we can provide the polynomial-sized description of its circuit family as advice to a Turing machine. When faced with an input of size n, the machine just simulates the circuit for this circuit provided to it.

Conversely, if L is decidable by a polynomial-time Turing machine M with access to an advice family $\{\alpha_n\}_{n\in\mathbb{N}}$ of size $a(n)$ for some polynomial a, then we can use the construction of Theorem 6.7 to construct for every n, a polynomial-sized circuit D_n such that on every $x \in \{0,1\}^n$, $\alpha \in \{0,1\}^{a(n)}$, $D_n(x, \alpha) = M(x, \alpha)$. 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 α_n "hardwired" as its second input. ■

Remark 6.12
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. This simple idea is often used in complexity theory.

Web draft 2006-09-28 18:09
DRAFT

6.2 Karp-Lipton Theorem

Karp and Lipton formalized the question of whether or not SAT has small circuits as: Is SAT in $\mathbf{P}$/poly? They showed that the answer is "NO" if the polynomial hierarchy does not collapse.

THEOREM 6.13 (KARP-LIPTON, WITH IMPROVEMENTS BY SIPSER)
If $\mathbf{NP} \subseteq \mathbf{P}/\mathrm{poly}$ *then* $\mathbf{PH} = \mathbf{\Sigma}_2^p$.

PROOF: To show that $\mathbf{PH} = \mathbf{\Sigma}_2^p$ it is enough to show that $\mathbf{\Pi}_2^p \subseteq \mathbf{\Sigma}_2^p$ and in particular it suffices to show that $\mathbf{\Sigma}_2^p$ contains the $\mathbf{\Pi}_2^p$-complete language Π_2SAT consisting of all true formulae of the form

$$\forall u \in \{0,1\}^n \, \exists v \in \{0,1\}^n \, \varphi(u,v) = 1 \,. \tag{2}$$

where φ is an unquantified Boolean formula.

If $\mathbf{NP} \subseteq \mathbf{P}/\mathrm{poly}$ 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 φ 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 search to decision reduction of Theorem 2.20, we actually know that there is a $q(n)$-sized circuit family $\{C'_n\}_{n\in\mathbb{N}}$ such that for every such formula φ 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 (2) is true then the following quantified formula is also true:

$$\exists w \in \{0,1\}^{10q(n)^2} \, \forall u \in \{0,1\}^n \, w \text{ describes a circuit } C' \text{ s.t. } \varphi(u, C'(\varphi,u)) = 1 \,. \tag{3}$$

Yet if (2) is false then certainly (regardless of whether $\mathbf{P} = \mathbf{NP}$) the formula (3) is false as well, and hence (3) is actually *equivalent* to (2)! However, since evaluating a circuit on an input can be done in polynomial time, evaluating the truth of (3) can be done in $\mathbf{\Sigma}_2^p$. ■

Similarly the following theorem can be proven, though we leave the proof as Exercise 3.

THEOREM 6.14 (KARP-LIPTON, ATTRIBUTED TO A. MEYER)
If $\mathbf{EXP} \subseteq \mathbf{P}/\mathrm{poly}$ *then* $\mathbf{EXP} = \mathbf{\Sigma}_2^p$.

Combining the time hierarchy theorem (Theorem 4.1) with the previous theorem implies that if $\mathbf{P} = \mathbf{NP}$ then $\mathbf{EXP} \not\subseteq \mathbf{P}/poly$. Thus upperbounds (in this case, $\mathbf{NP} \subseteq \mathbf{P}$) can potentially be used to prove circuit lowerbounds.

Web draft 2006-09-28 18:09

DRAFT

6.3 Circuit lowerbounds

Since $\mathbf{P} \subseteq \mathbf{P}/\text{poly}$, if $\mathbf{NP} \nsubseteq \mathbf{P}/\text{poly}$ then $\mathbf{P} \neq \mathbf{NP}$. The Karp-Lipton theorem gives hope that $\mathbf{NP} \nsubseteq \mathbf{P}/\text{poly}$. Can we resolve $\mathbf{P}$ versus $\mathbf{NP}$ by proving $\mathbf{NP} \nsubseteq \mathbf{P}/\text{poly}$? There is reason to invest hope in this approach as opposed to proving direct lowerbounds on Turing machines. By representing computation using circuits we seem to actually peer into the guts of it rather than treating it as a blackbox. Thus we may be able to get around the limitations of relativizing methods shown in Chapter 4.

Sadly, such hopes have not yet come to pass. After two decades, the best circuit size lowerbound for an $\mathbf{NP}$ language is only $5n$. (However, see Exercise 1 for a better lowerbound for a language in $\mathbf{PH}$.) On the positive side, we have had notable success in proving lowerbounds for more restricted circuit models, as we will see in Chapter 13.

By the way, it is easy to show that for large enough n, almost every boolean function on n variables requires 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 indegree at most 2). Hence this is an upperbound on the number of functions on n variables with circuits of size s. For $s = 2^n/(10n)$, this number is at most $2^{2^n/10}$, which is miniscule compared 2^{2^n}, the number of boolean functions on n variables. Hence most Boolean functions do not have such small circuits. ■

Remark 6.16
Another way to present this result is as showing that with high probability, a *random* function from $\{0,1\}^n$ to $\{0,1\}$ does not have a circuit of size $2^n/10n$. This kind of proof method, showing the existence of an object with certain properties by arguing that a random object has these properties with high probability, is called the *probabilistic method*, and will be repeatedly used in this book.

The problem with the above counting argument is of course, that it does not yield an explicit Boolean function (say an $\mathbf{NP}$ language) that requires large circuits.

DRAFT

6.4 Non-uniform hierarchy theorem

As in the case of deterministic time, non-deterministic time and space bounded machines, 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 $T, T' : \mathbb{N} \to \mathbb{N}$ *with* $2^n/(100n) > T'(n) > T(n) > n$ *and* $T(n) \log T(n) = o(T'(n))$,

$$\mathbf{SIZE}(T(n)) \subsetneq \mathbf{SIZE}(T'(n))$$

Proof: The diagonalization methods of Chapter 4 do not seem to work for such a function, but nevertheless, we can prove it using the counting argument from above. To show the idea, we prove that $\mathbf{SIZE}(n) \subsetneq \mathbf{SIZE}(n^2)$.

For every ℓ, there is a function $f : \{0,1\}^\ell \to \{0,1\}$ that is not computable by $2^\ell/(10\ell)$-sized circuits. On the other hand, every function from $\{0,1\}^\ell$ to $\{0,1\}$ is computable by a $2^\ell 10\ell$-sized circuit.

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

$$\begin{aligned} g \in \quad & \mathbf{SIZE}(2^\ell 10\ell) = & \mathbf{SIZE}(11n^{1.1} \log n) \subseteq & \quad \mathbf{SIZE}(n^2) \\ g \notin \quad & \mathbf{SIZE}(2^\ell/(10\ell)) = & \mathbf{SIZE}(n^{1.1}/(11 \log n)) \supseteq & \quad \mathbf{SIZE}(n) \end{aligned}$$

■

6.5 Finer gradations among circuit classes

There are two reasons why subclasses of $\mathbf{P}$/poly are interesting. First, proving lowerbounds for these subclasses may give insight into how to separate $\mathbf{NP}$ from $\mathbf{P}$/poly. Second, these subclasses correspond to interesting computational models in their own right.

Perhaps the most interesting connection is to *massively parallel computers*. In such a computer one uses simple off-the-shelf microprocessors and links them using an *interconnection network* that allows them to send messages to each other. Usual interconnection networks such as the *hypercube* allows linking n processors such that interprocessor communication is possible —assuming some upperbounds on the total load on the network—in $O(\log n)$ steps. The processors compute in lock-step (for instance, to the ticks of a global clock) and are assumed to do a small amount of computation in each step, say an operation on $O(\log n)$ bits. Thus each processor

DRAFT

computers has enough memory to remember its own address in the interconnection network and to write down the address of any other processor, and thus send messages to it. We are purposely omitting many details of the model (Leighton [**?**] is the standard reference for this topic) since the validity of Theorem 6.24 below does not depend upon them. (Of course, we are only aiming for a loose characterization of parallel computation, not a very precise one.)

6.5.1 Parallel computation and NC

DEFINITION 6.18
A computational task is said to have *efficient parallel algorithms* if inputs of size n can be solved using a parallel computer with $n^{O(1)}$ processors and in time $\log^{O(1)} n$.

EXAMPLE 6.19
Given two n bit numbers x, y we wish to compute $x + y$ fast in parallel. The gradeschool algorithm proceeds from the least significant bit and maintains a *carry bit*. The most significant bit is computed only after n steps. This algorithm does not take advantage of parallelism. A better algorithm called *carry lookahead* assigns each bit position to a separate processor and then uses interprocessor communication to propagate carry bits. It takes $O(n)$ processors and $O(\log n)$ time.

There are also efficient parallel algorithms for integer multiplication and division (the latter is quite nonintuitive and unlike the gradeschool algorithm!).

EXAMPLE 6.20
Many matrix computations can be done efficiently in parallel: these include computing the product, rank, determinant, inverse, etc. (See exercises.)

Some graph theoretic algorithms such as shortest paths and minimum spanning tree also have fast parallel implementations.

But many well-known polynomial-time problems such as minimum matching, maximum flows, and linear programming are not known to have any good parallel implementations and are conjectured not to have any; see our discussion of **P**-completeness below.

Web draft 2006-09-28 18:09
DRAFT

Now we relate parallel computation to circuits. The *depth* of a circuit is the length of the longest directed path from an input node to the output node.

DEFINITION 6.21 (NICK'S CLASS OR NC)
A language is in $\mathbf{NC}^i$ if there are constants $c, d > 0$ such that it can be decided by a logspace-uniform family of circuits $\{C_n\}$ where C_n has size $O(n^c)$ and depth $O(\log^d n)$. The class $\mathbf{NC}$ is $\cup_{i\geq 1}\mathbf{NC}^i$.

A related class is the following.

DEFINITION 6.22 (**AC**)
The class $\mathbf{AC}^i$ is defined similarly to $\mathbf{NC}^i$ except gates are allowed to have unbounded fanin. The class $\mathbf{AC}$ is $\cup_{i\geq 0}\mathbf{AC}^i$.

Since unbounded (but poly(n)) fanin can be simulated using a tree of ORs/ANDs of depth $O(\log n)$, we have $\mathbf{NC}^i \subseteq \mathbf{AC}^i \subseteq \mathbf{NC}^{i+1}$, and the inclusion is known to be strict for $i = 0$ as we will see in Chapter 13. (Notice, $\mathbf{NC}^0$ is extremely limited since the circuit's output depends upon a constant number of input bits, but $\mathbf{AC}^0$ does not suffer from this limitation.)

EXAMPLE 6.23
The language PARITY $=\{x : x$ has an odd number of 1s$\}$ is in $\mathbf{NC}^1$. The circuit computing it has the form of a binary tree. The answer appears at the root; the left subtree computes the parity of the first $|x|/2$ bits and the right subtree computes the parity of the remaining bits. The gate at the top computes the parity of these two bits. Clearly, unwrapping the recursion implicit in our description gives a circuit of dept $O(\log n)$.

The classes **AC**, **NC** are important because of the following.

THEOREM 6.24
A language has efficient parallel algorithms iff it is in **NC**.

PROOF: Suppose a language $L \in \mathbf{NC}$ and is decidable by a circuit family $\{C_n\}$ where C_n has size $N = O(n^c)$ and depth $D = O(\log^d n)$. Take a general purpose parallel computer with N nodes and configure it to decide L as follows. Compute a description of C_n and allocate the role of each circuit

DRAFT

node to a distinct processor. (This is done once, and then the computer is ready to compute on any input of length n.) Each processor, after computing the output at its assigned node, sends the resulting bit to every other circuit node that needs it. Assuming the interconnection network delivers all messages in $O(\log N)$ time, the total running time is $O(\log^{d+1} N)$.

The reverse direction is similar, with the circuit having $N \cdot D$ nodes arranged in D layers, and the ith node in the tth layer performs the computation of processor i at time t. The role of the interconnection network is played by the circuit wires. ■

6.5.2 P-completeness

A major open question in this area is whether $\mathbf{P} = \mathbf{NC}$. We believe that the answer is NO (though we are currently even unable to separate **PH** from $\mathbf{NC}^1$). This motivates the theory of **P**-*completeness*, a study of which problems are likely to be in **NC** and which are not.

DEFINITION 6.25
A language is **P**-*complete* if it is in **P** and every language in **P** is logspace-reducible to it (as per Definition 3.14).

The following easy theorem is left for the reader as Exercise 12.

THEOREM 6.26
If language L is **P***-complete then*

1. $L \in \mathbf{NC}$ *iff* $\mathbf{P} = \mathbf{NC}$.
2. $L \in \mathbf{L}$ *iff* $\mathbf{P} = \mathbf{L}$. *(Where* **L** *is the set languages decidable in logarithmic space, see Definition 3.5.)*

The following is a fairly natural **P**-complete language:

THEOREM 6.27
Let CIRCUIT-EVAL *denote the language consisting of all pairs $\langle C, x \rangle$ such that C is an n-inputs single-output circuit and $x \in \{0,1\}^n$ satisfies $C(x) = 1$. Then* CIRCUIT-EVAL *is* **P***-complete.*

PROOF: The language is clearly in **P**. A logspace-reduction from any other language in **P** to this language is implicit in the proof of Theorem 6.7. ■

Web draft 2006-09-28 18:09

DRAFT

6.6 Circuits of exponential size

As noted, every language has circuits of size $O(n2^n)$. However, actually finding these circuits may be difficult— sometimes even undecidable. If we place a uniformity condition on the circuits, that is, require them to be efficiently computable then the circuit complexity of some languages could exceed $n2^n$. In fact it is possible to give alternative definitions of some familiar complexity classes, analogous to the definition of **P** in Theorem 6.7.

Definition 6.28 (DC-Uniform)
Let $\{\mathcal{C}_n\}_{n\geq 1}$ be a circuit family. We say that it is a *Direct Connect uniform* (DC uniform) family if, given $\langle n, i\rangle$, we can compute in polynomial time the ith but of (the representation of) the circuit C_n. More concretely, we use the adjacency matrix representation and hence a family $\{C_n\}_{n\in\mathbb{N}}$ is DC uniform iff the functions `SIZE`, `TYPE` and `EDGE` defined in Remark **??** are computable in polynomial time.

Note that the circuits may have exponential size, but they have a succinct representation in terms of a TM which can systematically generate any required node of the circuit in polynomial time.

Now we give a (yet another) characterization of the class **PH**, this time as languages computable by uniform circuit families of bounded depth. We leave it as Exercise 13.

Theorem 6.29
$L \in PH$ iff L can be computed by a DC uniform circuit family $\{\mathcal{C}_n\}$ that

- *uses AND, OR, NOT gates.*
- *has size $2^{n^{O(1)}}$ and constant depth (i.e., depth $O(1)$).*
- *gates can have unbounded (exponential) fanin.*
- *the NOT gates appear only at the input level.*

If we drop the restriction that the circuits have constant depth, then we obtain exactly **EXP** (see Exercise 14).

6.7 Circuit Satisfiability and an alternative proof of the Cook-Levin Theorem

Boolean circuits can be used to define the following **NP**-complete language:

Web draft 2006-09-28 18:09

DRAFT

DEFINITION 6.30
The *circuit satisfiability* language CKT-SAT consists of all (strings representing) circuits with a single output that have a satisfying assignment. That is, a string representing an n-input circuit C is in CKT-SAT iff there exists $u \in \{0,1\}^n$ such that $C(u) = 1$.

CKT-SAT is clearly in **NP** because the satisfying assignment can serve as the certificate. It is also clearly **NP**-hard as every CNF formula is in particular a Boolean circuit. However, CKT-SAT can also be used to give an alternative proof (or, more accurately, a different presentation of the same proof) for the Cook-Levin Theorem by combining the following two lemmas:

LEMMA 6.31
CKT-SAT *is* **NP**-*hard.*

PROOF: Let L be an **NP**-language and let p be a polynomial and M a polynomial-time TM such that $x \in L$ iff $M(x,u) = 1$ for some $u \in \{0,1\}^{p(|x|)}$. We reduce L to CKT-SAT by mapping (in polynomial-time) x to a circuit C_x with $p(|x|)$ inputs and a single output such that $C_x(u) = M(x,u)$ for every $u \in \{0,1\}^{p(|x|)}$. Clearly, $x \in L \Leftrightarrow C_x \in$ CKT-SAT and so this suffices to show that $L \leq_P$ CKT-SAT.

Yet, it is not hard to come up with such a circuit. Indeed, the proof of Theorem 6.7 yields a way to map M, x into the circuit C_x in logarithmic space (which in particular implies polynomial time). ■

LEMMA 6.32
CKT-SAT $\leq_p$ 3SAT

PROOF: As mentioned above this follows from the Cook-Levin theorem but we give here a direct reduction. If C is a circuit, we map it into a 3CNF formula φ as follows:

For every node v_i of C we will have a corresponding variable z_i in φ. If the node v_i is an AND of the nodes v_j and v_k then we add to φ clauses that are equivalent to the condition "$z_i = (z_j \wedge z_k)$". That is, we add the clauses

$$(\overline{z}_i \vee \overline{z}_j \vee z_k) \wedge (\overline{z}_i \vee z_j \vee \overline{z}_k) \wedge (\overline{z}_i \vee z_j \vee z_k) \wedge (z_i \vee \overline{z}_j \vee \overline{z}_k) .$$

Similarly, if v_i is an OR of v_j and v_k then we add clauses equivalent to "$z_i = (z_j \vee z_k)$", and if v_i is the NOT of v_j then we add the clauses $(z_i \vee z_j) \wedge (\overline{z}_i \vee \overline{z}_j)$.

Finally, if v_i is the output node of C then we add the clause z_i to φ. It is not hard to see that the formula φ will be satisfiable if and only if the circuit C is. ■

Web draft 2006-09-28 18:09

DRAFT

WHAT HAVE WE LEARNED?

- Boolean circuits can be used as an alternative computational model to TMs. The class **P**/poly of languages decidable by polynomial-sized circuits is a strict superset of **P** but does not contain **NP** unless the hierarchy collapses.
- Almost every function from $\{0,1\}^n$ to $\{0,1\}$ requires exponential-sized circuits. Finding even one function in **NP** with this property would show that $\mathbf{P} \neq \mathbf{NP}$.
- The class **NC** of languages decidable by (uniformly constructible) circuits with polylogarithmic depth and polynomial size corresponds to computational tasks that can be efficiently parallelized.

Chapter notes and history

Karp-Lipton theorem is from [**?**]. Karp and Lipton also gave a more general definition of advice that can be used to define the class $\mathcal{C}/a(n)$ for every complexity class $\mathcal{C}$ and function a. However, we do not use this definition here since it does not seem to capture the intuitive notion of advice for classes such as $\mathbf{NP} \cap \mathbf{coNP}$, **BPP** and others.

The class of **NC** algorithms as well as many related issues in parallel computation are discussed in Leighton [**?**].

Exercises

§1 [Kannan [**?**]] Show for every $k > 0$ that **PH** contains languages whose circuit complexity is $\Omega(n^k)$.

Hint: Keep in mind the proof of the *existence* of functions with high circuit complexity.

§2 Solve the previous question with **PH** replaced by $\mathbf{\Sigma}_2^p$.

§3 ([**?**], attributed to A. Meyer) Show that if $\mathbf{EXP} \subseteq \mathbf{P}/poly$ then $\mathbf{EXP} = \mathbf{\Sigma}_2^p$.

§4 Show that if $\mathbf{P} = \mathbf{NP}$ then there is a language in **EXP** that requires circuits of size $2^n/n$.

Web draft 2006-09-28 18:09

DRAFT

§5 A language $L \subseteq \{0,1\}^*$ is sparse if there is a polynomial p such that $|L \cap \{0,1\}^n| \leq p(n)$ for every $n \in \mathbb{N}$. Show that every sparse language is in $\mathbf{P}$/poly.

§6 (X's Theorem 19??) Show that if a sparse language is $\mathbf{NP}$-complete then $\mathbf{P} = \mathbf{NP}$. (This is a strengthening of Exercise 13 of Chapter 2.)

Hint: Show a recursive exponential-time algorithm S that on input a n-variable formula φ and a string $v \in \{0,1\}^n$ outputs 1 iff φ has a satisfying assignment v such that $v > u$ when both are interpreted as the binary representation of a number in $[2^n]$. Use the reduction from SAT to L to prune possibilities in the recursion tree of S.

§7 Show a logspace implicitly computable function f that maps any n-vertex graph in adjacency matrix representation into the same graph in adjacency list representation. You can think of the adjacency list representation of an n-vertex graph as a sequence of n strings of size $O(n \log n)$ each, where the ith string contains the list of neighbors of the ith vertex in the graph (and is padded with zeros if necessary).

§8 (*Open*) Suppose we make a stronger assumption than $\mathbf{NP} \subseteq \mathbf{P}$/poly: every language in $\mathbf{NP}$ has linear size circuits. Can we show something stronger than $\mathbf{PH} = \mathbf{\Sigma}_2^p$?

§9 (a) Describe an $\mathbf{NC}$ circuit for the problem of computing the product of two given $n \times n$ matrices A, B.

(b) Describe an $\mathbf{NC}$ circuit for computing, given an $n \times n$ matrix, the matrix A^n.

Hint: Use repeated squaring: $A^{2^k} = (A^{2^{k-1}})^2$.

(c) Conclude that the PATH problem (and hence every $\mathbf{NL}$ language) is in $\mathbf{NC}$.

Hint: What is the meaning of the (i,j)th entry of A^n?

§10 A *formula* is a circuit in which every node (except the input nodes) has outdegree 1. Show that a language is computable by polynomial-size formulae iff it is in (nonuniform) $\mathbf{NC}^1$.

Hint: a formula may be viewed —once we exclude the input nodes—as a directed binary tree, and in a binary tree of size m there is always a node whose removal leaves subtrees of size at most $2m/3$ each.

Web draft 2006-09-28 18:09
DRAFT

§11 Show that $\mathbf{NC}^1 = \mathbf{L}$. Conclude that $\mathbf{PSPACE} \neq \mathbf{NC}^1$.

§12 Prove Theorem 6.26. That is, prove that if L is **P**-complete then $L \in \mathbf{NC}$ (resp. **L**) iff $\mathbf{P} = \mathbf{NC}$ (resp. **L**).

§13 Prove Theorem 6.29 (that **PH** is the set of languages with constant-depth DC uniform circuits).

§14 Show that **EXP** is exactly the set of languages with DC uniform circuits of size 2^{n^c} where c is some constant (c may depend upon the language).

§15 Show that if linear programming has a fast parallel algorithm then $\mathbf{P} = \mathbf{NC}$.

Hint: in your reduction, express the CIRCUIT-EVAL problem as a linear program and use the fact that $x \vee y = 1$ iff $x + y \geq 1$. Be careful; the variables in a linear program are real-valued and not boolean!

DRAFT