Chapter 5

Circuits

5.1 Basic results and definitions

A *Boolean circuit* can be seen as a directed, acyclic, connected graph in which the input nodes are labeled with variables x_i and constants $1, 0$ (or $\mathtt{T}, \mathtt{F}$), and the internal nodes are labeled with $\mathtt{Not}, \mathtt{And}, \mathtt{Or}$. We often denote $\mathtt{Not}, \mathtt{And}, \mathtt{Or}$ with the standard Boolean connectives $\neg, \wedge, \vee$, respectively. Furthermore, we often use $\bar{x}$ to denote $\neg x$.

Nodes are also called *gates* in the context of circuits. The *fan-in* (i.e., number of incoming edges) of a $\mathtt{Not}$-gate is always 1, and the fan-in of $\mathtt{And}, \mathtt{Or}$ can be anything. The *fan-out* (i.e., number of outgoing edges) of any node can be arbitrary. Each node in the graph can be associated with a Boolean function in the obvious way. The function associated with the output gate(s) is the function computed by the circuit. Note that a Boolean formula can be seen as a circuit in which every node has fan-out 1.

The *size* of a circuit is its number of gates, and the *depth* of a circuit is the largest number of gates on any path from an input gate to an output gate.

Alternatively, circuits (of fan-in 2 and one output) can be defined more formally as follows. A Boolean circuit is a program consisting of finitely many instructions of the form: $P_i := 0$, $P_i := 1$, $P_i := x_l$, $P_i := P_j \wedge P_k$, $P_i := P_j \vee P_k$, $P_i := \neg P_j$, where $j, k < i$ (to ensure acyclicity), and where $x_1, \ldots, x_n$ are the input variables. We want to compute P_m where m is the maximum index.

A *family of circuits* is an infinite sequence $C = \{C_n\} = \{C_0, C_1, C_2, \ldots\}$ of Boolean circuits where C_n has n input variables.

We say that a language $L \subseteq \{0,1\}^*$ has polysize circuits if there exists a polynomial p and a family C such that $|C_n| \leq p(n)$, and $\forall x \in \{0,1\}^*$, $x \in L$ iff $C_{|x|}(x) = 1$.

Lemma 5.1.1 *All languages in* P *have polysize circuits.*

See proof of theorem 2.2.2.

The converse of the above lemma (i.e., "L has polysize circuits implies $L \in \mathsf{P}$") does not hold, unless we put a severe restriction on how the n-th circuit is generated; as it stands, there are undecidable languages that have polysize circuits.

Lemma 5.1.2 *There are undecidable languages that have polysize circuits.*

PROOF: Recall that we showed in theorem 4.1.1 that the language A_{TM} is undecidable. We may assume that the instances of A_{TM}, i.e., $\langle M, x\rangle$ are encoded as binary strings where the first bit is 1. This is a minor technical point that makes the rest of the proof simpler. Let $(n)_b$ be the binary representation of $n \in \mathbf{N}$, so for example $(5)_b = 101$.

Let $U = \{1^n | (n)_b \in \mathsf{A}_{\mathrm{TM}}\}$, and since A_{TM} is reducible to U (via an exponential time reduction, which is nevertheless a recursive reduction), it follows that U is also undecidable.

On the other hand, U has a polysize family of circuits $C = \{C_n\}$ deciding it, where C_n is an `And`-gate connected to all the inputs if $1^n \in U$, and a single `F`-gate otherwise. □

The trick in the above proof is to hide the computation in the "non-uniformity" of the circuits; the circuit C_n is indeed small, but given n, we do not know how to construct C_n (i.e., is it the `And`-gate or is it the `F`-gate?). Thus, we make the following definition: a family of circuits $C = \{C_n\}$ is *uniform* if there is a $O(\log(n))$-space TM M which on input 1^n, outputs C_n.

Theorem 5.1.3 *A language L has uniform polysize circuits iff $L \in \mathsf{P}$.*

Those languages (or Boolean functions) that can be decided with polysize, constant fan-in, and depth $O(\log^k(n))$ circuits, form the class NC^k. The class AC^k is defined in the same way, except we allow unbounded fan-in. We set $\mathsf{NC} = \cup_k \mathsf{NC}^k$, and $\mathsf{AC} = \cup_k \mathsf{AC}^k$, and while it is easy to see that the

uniform version of NC is in P, it is an interesting open question whether they are equal.

Lemma 5.1.4 *For all k, $\mathsf{AC}^k \subseteq \mathsf{NC}^{k+1} \subseteq \mathsf{AC}^{k+1}$. Thus, $\mathsf{NC} = \mathsf{AC}$.*

Exercise 5.1.5 *Prove lemma 5.1.4.*

We say that a circuit family is in *layered form* if each circuit in the family has the following three properties: (i) it consists in alternating layers of `And`'s and `Or`'s, (ii) there are edges only between consecutive layers, and (iii) negations occur only at the input level. Note that in a layered circuit, the depth is the same as the number of layers.

Lemma 5.1.6 *Any* AC *circuit family C can be put in layered form incurring a polynomial increase in size, and a constant increase in depth.*

PROOF: Let C_i be the i-th circuit in the given family. First, traverse the circuit from its output towards inputs, pushing any `Not` gates through `And` and `Or` gates using de Morgan laws, and replicating gates if necessary (see figure 5.1). At the end, all variables and negations are at level 0 (that

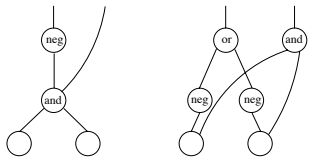

Figure 5.1: Using de Morgan and replicating gates.

is, layer 0 consists of $x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n$). Now choose if you want to start the alternation at level 1 with `And` or `Or` gates; say we start with `And` gates. Consider all gates of depth 1 (these will be gates connected directly to the inputs, negated or unnegated). If a depth-1 gate is an `And` gate, do nothing. If it is an `Or` gate, then put an `And` gate on each one of its wires (see figure 5.2). Now consider gates at level 2, and repeat. We may frequently have to collapse several `And`'s (or several `Or`s) into one `And` (or `Or`), since we are allowed unbounded fan-in. □

Lemma 5.1.7 *Addition of two integers is in* AC^0.

PROOF: Let $a_{n-1}, \ldots, a_0, b_{n-1}, \ldots, b_0$ be the two inputs (n-bit integers, where a_0, b_0 are the least significant bits).

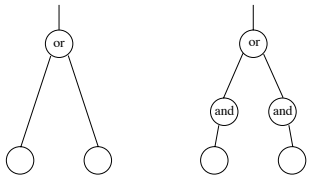

Figure 5.2: Putting `And` gates on the wires of an `Or`.

Algorithm 5.1.8 (Classical Binary Addition)
On input $a_{n-1} \dots a_0$ & $b_{n-1} \dots b_0$

1. $y_0 := a_0 \oplus b_0$
2. $\text{carry}_0 := a_0 \wedge b_0$
3. for $i = 1, \dots, n-1$
4. $\quad y_i := a_i \oplus b_i \oplus \text{carry}_{i-1}$
5. $\quad \text{carry}_i := (a_i \wedge b_i) \vee (a_i \wedge \text{carry}_{i-1}) \vee (b_i \wedge \text{carry}_{i-1})$
6. $y_n := \text{carry}_{n-1}$

Suppose we were to implement the classical algorithm with a circuit. Since carry_i depends on carry_{i-1}, the depth of the resulting circuit would be at least $O(n)$, and hence not an AC^0 circuit.

Instead, note that the i-th bit of the sum is $a_i \oplus b_i \oplus \text{carry}_i$, where carry_i is the carry bit left over after summing the $(i-1)$ first bits. Compute carry_i differently; note that the i-th carry is 1 iff $\exists j \in \{0, \dots, i-1\}$ such that $a_j \wedge b_j$ and $\forall k \in \{j+1, \dots, i-1\}$ it is the case that $a_k \vee b_k$. This can be stated with:

$$\text{carry}_i := \bigvee_{0 \le j < i} \left[(a_j \wedge b_j) \wedge \bigwedge_{j < k < i} (a_k \vee b_k) \right].$$

Note that this is a circuit of bounded depth. □

Lemma 5.1.9 *Repeated addition of integers is in* NC^1.

Proof: Assume that we want to add n n-bit numbers. Using the previous result directly, we can only show that repeated addition is in AC^1, which is a weaker claim. To show that it is in NC^1, we first design an NC^0 circuit which takes as input three integers, and returns two integers whose sum is the same as the sum of the original three: given a, b, c, we produce x, y as

follows:

$$a_i, b_i, c_i \mapsto x_i = \underbrace{a_i \oplus b_i \oplus c_i}_{\text{sum } \textit{without} \text{ carries}} \quad , \quad y_{i+1} = \underbrace{(a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i)}_{\text{the carry}}.$$

Each step reduces the number of integers to be added by 1/3. So in a logarithmic number of steps we are left with two integers. We now add those two with an AC^0 circuit, as was shown possible in lemma 5.1.7. □

We say that a Boolean function is *symmetric* if it only depends on the number of 1s in the input, and not on their order. More formally, if $f = \{f_n\}$, then f is symmetric if for all n, given any permutation $\pi \in S_n$, $f_n(x_1, x_2, \ldots, x_n) = f_n(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$.

Examples of symmetric functions are the *parity function*, $\mathrm{Par} = \{\mathrm{Par}_n\}$, where $\mathrm{Par}_n(x_1, \ldots, x_n)$ is 1 if the number of 1s in its input is odd, and the *majority function*, $\mathrm{Maj} = \{\mathrm{Maj}_n\}$, where $\mathrm{Maj}_n(x_1, \ldots, x_n)$ is 1 if at least half of its inputs are 1, and a generalization of majority, namely the *threshold function*, $\mathrm{Th} = \{\mathrm{Th}_{k,n}\}$ which outputs 1 iff at least k of its n variables are 1.

Lemma 5.1.10 *All symmetric Boolean functions can be computed with* NC^1 *circuits.*

PROOF: Notice that if $f = \{f_n\}$ is a symmetric Boolean function, then the value of f_n depends only on the number of inputs that are set to 1. To construct the circuit that calculates f_n, we treat all inputs as 1-bit numbers and construct a circuit that adds them, thus counting the number of 1's in the input.

From lemma 5.1.9, we know that this can be done using an NC^1 circuit. The output of this circuit is a $\log(n)$ bit number. Now look at those $\log(n)$ many bits as the input to a circuit which gives us the value of f_n; this circuit can be given in CNF, since each clause will have $\log(n)$ many literals, and there are $O(2^{\log(n)})$ many clauses, and thus the CNF circuit has size $O(n \log(n))$ (and depth 2 when the fan-in is unbounded, and depth $\log(n)$ when the fan-in is required to be 2).

Altogether we obtain an NC^1 circuit. □

In the footnote on page 7 we defined regular languages in terms of TM; recall that a deterministic finite automaton (DFA) is a TM that reads the input left-to-right, changing states as it goes, and never using any extra space[1].

[1]We defined regular expressions on page 44; they also turn out to capture exactly

Lemma 5.1.11 *The set of regular languages is contained in* NC^1.

PROOF: Let L be a regular language decided by a DFA M. Define M_a, $a \in \Sigma$, to be the Boolean matrix with a 1 in position (i,j) iff on symbol a machine M moves from state q_i to state q_j.

Given $a \in \{0,1\}^*$, $a = a_1 a_2 \dots a_n$, consider the iterated Boolean matrix product $\mathbf{M} := M_{a_1} M_{a_2} \cdots M_{a_n}$ (Boolean product means that instead of $+$ we have $\vee$ and instead of $\times$ we have $\wedge$). This product can be clearly computed with an NC^1 circuit as each M_a is of constant size (i.e., $|Q| \times |Q|$).

Let the output be the entry of $\mathbf{M}$ in the row corresponding to the initial state, in the column corresponding to the accepting state. □

Exercise 5.1.12 *Prove by induction on n that the (i,j)-th entry of matrix $M_{a_1} M_{a_2} \dots M_{a_n}$ is 1 iff when started in state q_i machine M reaches state q_j after reading $a_1 a_2 \dots a_n$.*

Lemma 5.1.13 *For uniform circuit classes,* $\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{NC}^2$.

PROOF: The first containment follows from the fact that an NC^1 circuit can be evaluated by a depth-first manner using only logspace. On the other hand, the circuits are uniform, so on an input $x \in \{0,1\}^n$, C_n can be computed in logspace. However, $|C_n|$ is polynomial in n, so it will not fit in logspace; we can cope with that by only generating the parts of the circuit we need at a particular step of the depth-first evaluation.

The second containment is by definition.

The third follows from the fact that DIRECTREACH is in NC^2: matrix product is in NC^1, and therefore iterated matrix product is in NC^2, and so transitive closure can be computed in NC^2. □

5.2 Shannon's lower bound

In the 1960s, Shannon showed that most Boolean functions require large circuits, but today, for concrete Boolean functions, the best lower bounds we have are linear. Let B_n be the set of Boolean functions on n variables ($|B_n| = 2^{2^n}$), and in this section consider circuits with gates $\{\wedge, \vee, \neg, 0, 1\}$ of fan-in exactly two except for fan-in one negations at input level only. Let

regular languages (see [Sip06]).

the size of a circuit be the number of $\{\wedge, \vee\}$ gates (so the negation gates at the input level do not count for size).

Claim 5.2.1 *The number of circuits with n variables and size s is bounded above by $(2 \cdot (s + 2n + 2)^2)^s$.*

Exercise 5.2.2 *Prove claim 5.2.1.*

Claim 5.2.3 *For $s = 2^n/(10n)$ the value of the expression $(2\cdot(s+2n+2)^2)^s$ is bounded above by $2^{2^n/5}$, and so "almost all" Boolean functions in B_n require circuits of size $\Omega(2^n/n)$.*

PROOF: Setting $s = 2^n/(10n)$, and taking the log of the expression in $(2 \cdot (s + 2n + 2)^2)^s$, we obtain

$$2^n/(10n) \cdot \underbrace{2 \log \left(\sqrt{2} \, (2^n/10n + 2n + 2) \right)}_{(*)} \tag{5.1}$$

and for sufficiently large n, $(*) < n$, so for sufficiently large n (5.1) is less than $2^n/(10n) \cdot 2n$, which is $2^n/5$. □

We know that $n2^n$ many gates are sufficient to compute any Boolean function in B_n, since $n2^n$ is the size of its CNF (or DNF) representation. It is also easy to give a slightly better construction, from which it follows that every Boolean function can be computed with $O(2^n)$ gates. We do this by induction on the number of inputs. Let

$$f = (\neg x_n \wedge f|_{x_n=0}) \vee (x_n \wedge f|_{x_n=1})$$

where $f \in B_n$ and $f|_{x_n=0}, f|_{x_n=1} \in B_{n-1}$. Let $s(f)$ be the size of the smallest circuit computing the function f. Then

$$s(f) \leq 2 \cdot \max\{s(f|_{x_n=0}), s(f|_{x_n=1})\} + 3.$$

The claim follows from this.

In fact we can get an even tighter upper bound showing that the lower bound $\Omega(2^n/n)$ in claim 5.2.3 is very exact.

Claim 5.2.4 *Every Boolean function f in B_n can be computed with circuits of size $O(2^n/(n - \log n)) \subseteq O(2^n/(n - \frac{1}{2}n)) \subseteq O(2^n/n)$.*

PROOF: Observe that for any k there is a circuit with multiple outputs and size $O(2^{2^k})$ such that for every f in B_k there is an output computing f. This is just a CNF-like circuit, with all the possible 2^k different clauses at level 1, and at level 2 it has `Or`'s connected to all the possible subsets of clauses at level 1.

For $f \in B_n$, and for any $k \leq n$, we have that $f(x_1, \ldots, x_n)$ equals

$$\bigvee_{a_1,\ldots,a_k \in \{0,1\}} (x_1^{a_1} \wedge \cdots \wedge x_k^{a_k}) \wedge f(a_1, \ldots, a_k, x_{k+1}, \ldots, x_n)$$

where x^a is x if $a = 1$, and $\neg x$ if $x = 0$. Now construct a circuit with outputs for all functions $x^{a_1} \wedge \ldots \wedge x^{a_k}$ where $a_1, \ldots, a_k \in \{0,1\}^k$. Building this circuit inductively on k, layer by layer, results in a circuit of size $O(2^k)$. On the other hand, all the functions $f(a_1, \ldots, a_k, x_{k+1}, \ldots, x_n)$, for $a_1, \ldots, a_k \in \{0,1\}^k$, can be computed with a circuit of size $O(2^{2^{n-k}})$. Putting it all together, we have a circuit of size $O(2^k) + O(2^{2^{n-k}})$ computing f. Select k so that $n - k = \log(n - \log n)$. □

As we know that most Boolean functions require $\Omega(2^n/n)$ many gates, we can "construct" a hard Boolean function by enumeration: for every n, examine all Boolean functions in B_n, and for each such Boolean function examine all circuits of size up to $\Omega(2^n/n)$, and pick the first Boolean function not computable by such a circuit. This way, we obtain a family $f = \{f_n\}$. The trouble with this "construction" is that it tells us very little about the nature of hard Boolean functions (not to mention the infeasibility of the procedure, as it requires EXPSPACE).

An open question is: can we give a natural Boolean function $f = \{f_n\}$ that is provably hard? By "natural" we mean for example a function $f = \{f_n\}$ which computes if a graph of a given size has a clique; i.e., a Boolean function related to a natural combinatorial problem that requires large circuits.

So far, the best we can do for natural functions is the following linear lower bound.

Claim 5.2.5 *$Th_{2,n}$ requires circuit size at least $2n - 4$.*

PROOF: We do the proof by induction on n. For $n = 2$ and $n = 3$ it is easy. Otherwise, let C be an optimal circuit for $\mathrm{Th}_{2,n}$, and suppose that the bottom most gate acts on variables x_i, x_j, where $i \neq j$. Under the

four possible settings to x_i, x_j, the function $\mathrm{Th}_{2,n}$ has three possible subfunctions: $\mathrm{Th}_{0,n-2}, \mathrm{Th}_{1,n-2}, \mathrm{Th}_{2,n-2}$. It follows that either x_i or x_j fans out to another gate in C, for otherwise C would have only two inequivalent subcircuits under the settings of x_i, x_j. Suppose that it is x_i that fans-out to another gate. Setting x_i to 0 will eliminate the need for at least two gates from C. The resulting function is $\mathrm{Th}_{2,n-1}$, which by induction requires circuits of size $2(n-1)-4$. Adding the two eliminated gates to this bound shows that C has at least $2n-4$ gates, which completes the induction. □

5.3 The probabilistic method

In this section we give two different proofs of a super-polynomial lower bound for parity as computed by AC^0 circuits[2].

5.3.1 First proof of Parity $\notin \mathsf{AC}^0$

We show PARITY $\notin \mathsf{AC}^0$, where PARITY is the language of strings over $\{0,1\}$ with an odd number of 1s, i.e., PARITY $= \{x \in \{0,1\}^* | \mathrm{Par}_{|x|}(x) = 1\}$, where the function $\mathrm{Par} = \{\mathrm{Par}_n\}$ was defined on page 65.

All Boolean functions on n variables can be computed by circuits of depth 2 (CNF or DNF). In Figure 5.3 we have three circuits of depth 2, computing the same function. The second circuit is a *layered* version of the first (see lemma 5.1.6). The third circuit is a transformation of the first from an `And-Or` circuit (i.e., a circuit consisting of a single output `And` gate, a middle layer of `Or` gates, and an input layer of variables and their negations, and connections only between consecutive layers) into an `Or-And` circuit (defined analogously to an `And-Or` circuit, but the output gate is an `Or` gate). This is accomplished with the distributive laws:

$$\begin{aligned} (x \wedge (y \vee z)) &\equiv (x \wedge y) \vee (x \wedge z) \\ (x \vee (y \wedge z)) &\equiv (x \vee y) \wedge (x \vee z). \end{aligned} \tag{5.2}$$

Suppose that we have an `And-Or` circuit in which all the `Or` gates have fan-in c and the `And`-gate has fan-in d. We use the distributive laws (5.2) to transform our `And-Or` circuit into an `Or-And` circuit, where now the `And`s have fan-in d and the output `Or` has fan-in c^d; see Figure 5.4.

[2]Two other prominent lower bounds for circuits are Razborov's super-polynomial lower bound for monotone circuits computing CLIQUE (stated, but not proved, as theorem 6.4.8),

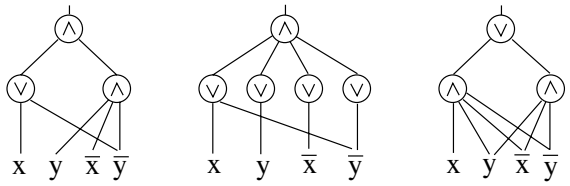

Figure 5.3: Layers, and `And-Or` as `Or-And`.

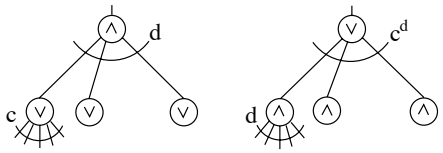

Figure 5.4: `And-Or` as `Or-And`.

Consider a polybounded family of circuits $C = \{C_n\}$, where each C_n is an `And-Or` circuit. Suppose that for every C_n, the fan-in of the `Or` gates (in the middle layer) is bounded by c, and the output `And`-gate depends on constantly many input variables, meaning that if you collect all the literals connected to an output `And`-gate (by a path from the input layer, passing through the layer of `Or`s) you get a set of variables whose size is always bounded by some constant e.

The following claim holds for such a C (as described in the previous paragraph).

Claim 5.3.1 *C can be converted to an* `Or-And` *family C', still of polynomial size, of constant input fan-in. Also, the claim still holds if we change the definition of "depends on constantly many input variables" to mean a* semantic dependence*. That is, each output* `And`*-gate might be connected to an arbitrary number of literals, but its value depends on the value of only constantly many input variables.*

PROOF: If the `And` gate is connected (through the `Or` gates) to no more than e variables, then there are at most 3^e different (in terms of input literals) `Or` gates (for each variable x the `Or` gate either uses x, its negation, or none of them). Deleting duplicate input connections we can lower the fan-in of the

and the Razborov-Smolensky results showing that for p, q different primes $\mathsf{AC}^0[p] \neq \mathsf{AC}^0[q]$.

`And` gate to be at most 3^e. The fan-in of the `Or` gates is, of course, bounded by e. Thus, using distributive laws to exchange the `And` and `Or` levels, we will create a single `Or` gate connected to no more than e^{3^e} `And` gates, which is a constant number!

If we consider semantic dependence instead, then if we set all but the e variables to, say, 0, the circuit would compute the same Boolean function, and the output `And` would depend syntactically on no more than e variables. This finishes the proof. □

Note that the last word on the type of results presented in claim 5.3.1 is the Håstad switching lemma; see [Bea94].

Claim 5.3.2 *Suppose that we have a circuit for $\overline{Par}_n$ of depth t and size g. Then there is a circuit of same depth and size for Par_n.*

PROOF: Just note that $\overline{\mathrm{Par}}_n(x_1, x_2, \ldots, x_n) = \mathrm{Par}_n(\overline{x}_1, x_2, \ldots, x_n)$ □

Claim 5.3.3 *Suppose that we have a circuit C for Par_n, and we set a variable x_i to 0 (or 1) and simplify. Then the resulting circuit C' computes Par_{n-1} or $\overline{Par}_{n-1}$. Consequently, the same applies if we set any number of variables.*

PROOF: This follows immediately from the observation that

$$\mathrm{Par}_n(0, x_2, \ldots, x_n) = \mathrm{Par}_{n-1}(x_2, \ldots, x_n)$$
$$\mathrm{Par}_n(1, x_2, \ldots, x_n) = \overline{\mathrm{Par}}_{n-1}(x_2, \ldots, x_n)$$

(We used x_1, but obviously the same works for any x_i.) □

Claim 5.3.4 *Par_n can be computed with a circuit of polynomial size and depth $O(\log(n))$. Thus,* PARITY $\in \mathsf{NC}^1$.

PROOF: This has already been shown in lemma 5.1.10. But this can also be checked directly, as

$$\mathrm{Par}_n(x_1, \ldots, x_n) = \mathrm{Par}_{\lfloor \frac{n}{2} \rfloor}(x_1, \ldots, x_{\lfloor \frac{n}{2} \rfloor}) \oplus \mathrm{Par}_{\left(n - \lfloor \frac{n}{2} \rfloor\right)}(x_{(\lfloor \frac{n}{2} \rfloor + 1)}, \ldots, x_n).$$

Using this fact we can compute Par_n with a tree of XORs. Note that each $\oplus$ can be replaced (locally) using $x \oplus y \equiv (x \wedge \bar{y}) \vee (\bar{x} \wedge y)$, and note that all the gates in this circuit have fan-in 2. □

Theorem 5.3.5 PARITY $\notin \mathsf{AC}^0$.

The rest of this section is a proof of this theorem. We start with a claim.

Claim 5.3.6 PARITY *cannot be decided with a polysize* `Or-And` *circuit.*

PROOF: Suppose that Par_n has an `Or-And` circuit. If one of the `And` gates has as inputs both x_i and $\bar{x}_i$, then eliminate this `And` since its contribution to the output `Or` is zero. If an `And` gate is missing a variable x_i as input (that is, neither x_i nor $\bar{x}_i$ is an input to this `And`) then by setting all the other literal inputs to this `And` to 1, we set the circuit to 1, *regardless* of the value of this x_i, which means that the circuit is not computing the parity correctly. Summing up, this `Or-And` circuit is such that to each `And` gate each variable comes in as an input (either as itself or negated). But this means that each `And` corresponds to "one row" of the truth table for Par_n, i.e., this `Or-And` circuit is the conjunctive normal form of Par_n. Since there are 2^{n-1} rows with value 1, there must be 2^{n-1} `And` gates, and so Par_n requires at least 2^{n-1} gates. □

Exercise 5.3.7 *Show that* PARITY *cannot be computed with a polysize* `And-Or` *circuit.*

Theorem 5.3.5 follows directly from the next claim.

Claim 5.3.8 $\forall t, \forall c, \forall p(n)$, PARITY *cannot be computed using a depth* t *circuit family of size* $p(n)$ *that has input-level fan-in* $\leq c$.

Note that theorem 5.3.5 follows from this claim because we can always insert a dummy layer of gates at the input level (all of fan-in 1), and increase the depth to $(t+1)$.

The case $t = 2$ was already done (from claim 5.3.6 and exercise 5.3.7 we know that PARITY cannot be computed with a polysize circuit of depth 2, let alone one where there is a restriction on the input-level fan-in).

For $t > 2$ we are going to use the following strategy: let t be the least such depth, and suppose that $S_1, S_2, S_3, \ldots$ is a family of circuits of depth t and polysize computing PARITY. Then we are going to produce $S'_1, S'_2, S'_3, \ldots$, of depth $(t-1)$ and polysize, thereby getting a contradiction.

To produce S'_n we are going to take S_{4n^2} and set $(4n^2 - n)$ of its variables to 0 and 1 in such a way that we can use the distributive laws (5.2) on levels 1 and 2 to exchange these two levels *without increasing the size of the circuit exponentially.* To do this, it is enough to have a situation where *each gate*

on level 2 depends on only a constant number of variables (see claim 5.3.1 and the discussion leading up to it). Then, levels 2 and 3 can be collapsed to a single layer leaving a circuit of depth $(t-1)$.

How can we come up with the right restriction mapping S_{4n^2} to S'_n? We are going to show one exists without explicitly constructing it: we use a *random restriction* and show that the probability of getting an appropriate substitution is positive, so we can conclude that one exists.

The kind of reasoning just described is a very powerful technique in combinatorics, and it goes under the name of a *probabilistic argument*; *"My thoughts and my discourse as madmen's are, at random from the truth vainly express'd"* (Shakespeare, Sonnet CXLVII).

We shall use a family of random restrictions $r = \{r_n\}$, where

$$\Pr[x_i^{r_n} = x_i] = \frac{1}{\sqrt{n}}$$

$$\Pr[x_i^{r_n} = 0] = \Pr[x_i^{r_n} = 1] = \frac{1 - \frac{1}{\sqrt{n}}}{2}.$$

Then, $S_n^{r_n}$ is the result of applying the random restriction r_n to the variables $x_1, x_2, \ldots, x_n$, and simplifying S_n accordingly, so $S_n^{r_n}$ is Par_m or $\overline{\mathrm{Par}}_m$ where $m \leq n$.

Since it will be understood that we are applying r_n to S_n, we are going to drop the subscript of r_n, and write r.

Exercise 5.3.9 *As a warm up to the probability that we are going to be using in this section (and in chapter 7) show the* Markov inequality*:*

$$\Pr[X \geq a] \leq \frac{E(X)}{a}, \tag{5.3}$$

and the Chebyshev inequality*:*

$$\Pr[|X - E(X)| \geq a] \leq \frac{V(X)}{a^2}. \tag{5.4}$$

Claim 5.3.10 *Pr[there are fewer than $\frac{\sqrt{n}}{2}$ vars in S_n^r] $= O(\frac{1}{\sqrt{n}})$.*

PROOF: Consider r acting on S_n. The probability of setting a variable to 0 or 1 is $q = 1 - \frac{1}{\sqrt{n}}$, and the probability of leaving it unset is $p = \frac{1}{\sqrt{n}} = 1 - q$. This is done independently for each variable, and so it is a *binomial*

distribution with probability of having exactly k successes (i.e., of leaving exactly k variables unset) at $\binom{n}{k}p^k q^{n-k}$.

Let X be the random variable counting the number of successes (i.e., counting the number of unset variables) in n trials. We want to compute $E(X)$ and $\mathrm{Var}(X)$. Let

$$X_i = \begin{cases} 1 & \text{if } i\text{-th trial is a success, i.e., } x_i^r = x_i \\ 0 & \text{otherwise} \end{cases}$$

then $E(X_i) = 1 \cdot p + 0 \cdot q = p$, and $E(X) = E(\sum_{i=1}^n X_i) = \sum_{i=1}^n E(X_i) = n \cdot p$ and

$$\begin{aligned} \mathrm{Var}(X) &= E(X^2) - E(X)^2 = E((\sum_{i=1}^n X_i)^2) - (n \cdot p)^2 \\ &= E(\sum_{i=1}^n \sum_{j=1}^n X_i X_j) - (n \cdot p)^2 \\ &= n(n-1)p^2 + \underbrace{n \cdot p}_{(*)} - (n \cdot p)^2 = np - np^2 = np(1-p) \end{aligned}$$

where $(*)$ is for when $i = j$, because $E(X_i X_i) = p$.

We are now ready to prove the claim.

$$\begin{aligned} &\Pr[\text{ fewer than } \frac{\sqrt{n}}{2} \text{ vars remain in } S_n^r \,] \\ \leq &\Pr[X \leq \frac{\sqrt{n}}{2}] = \Pr[\sqrt{n} - X \geq \sqrt{n} - \frac{\sqrt{n}}{2}] = \Pr[E(X) - X \geq \frac{\sqrt{n}}{2}] \\ \leq &\Pr[|X - E(X)| \geq \frac{\sqrt{n}}{2}] \end{aligned}$$

Using Chebyshev inequality, the last line can be bounded by

$$\leq \frac{\mathrm{Var}(X)}{\left(\frac{\sqrt{n}}{2}\right)^2} = \frac{\sqrt{n}-1}{\left(\frac{n}{4}\right)} \leq 4 \cdot \frac{\sqrt{n}}{n} = 4 \cdot \frac{1}{\sqrt{n}}$$

which finishes the proof of the claim. □

We are now going to show that after a random restriction r, the `And` gates on the second level depend on a constant number of variables with high probability. In other words, we prove by induction on c that the `And` gates depend on "too many variables" with probability $O(\frac{1}{n^k})$.

The basis case is $c = 1$, where fan-in 1 into the `Or` gates (at level 1) means that these `Or` gates can be disregarded, and the inputs connected directly to the `And` gates (this is depicted in Figure 5.5).

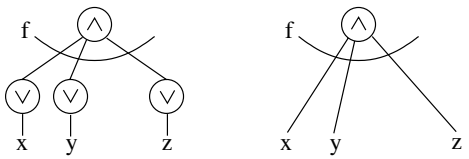

Figure 5.5: Basis case: $c = 1$.

Exercise 5.3.11 *In this section we sedulously use the logarithm function; as a warm up, prove (i)* $\log_c(a) = 1/\log_a(c)$*, and (ii)* $\log_a(b)/\log_a(c) = \log_c(b)$.

We now consider two possibilities:

1. The `And`-gate has a large fan-in ($f \geq 4 \cdot k \cdot \ln(n)$),
2. the `And`-gate has a small fan-in ($f < 4 \cdot k \cdot \ln(n)$).

Claim 5.3.12 *In the first case (large fan-in), we have:*

$$\Pr[\ \texttt{And}\text{-}\textit{gate is not } 0\ ] = O\left(\frac{1}{n^k}\right)$$

PROOF: This is because it is very likely that a random r would have set one of the `And`'s inputs to 0. So with high probability, the `And`-gate depends on no variables. We now give the details.

$$\begin{aligned} &\Pr[\texttt{And}\text{-gate not } 0] \\ \leq &\Pr[\text{all inputs not } 0] \leq \Pr[\text{a fixed input not } 0]^{4k\ln(n)} \\ \leq &\left(\frac{1+\frac{1}{\sqrt{n}}}{2}\right)^{4k\ln(n)} \end{aligned}$$

and for $n \geq 4$,

$$\leq \left(\frac{3}{4}\right)^{4k\ln(n)} = n^{4k\ln(\frac{3}{4})} \leq n^{-k}$$

which proves the claim. □

Claim 5.3.13 *In the second case (small fan-in), we have:*

$$\Pr[\ \texttt{And}\text{-gate depends on more than } 18k \text{ inputs }] = O\left(\frac{1}{n^k}\right)$$

Exercise 5.3.14 *Show the following upper bound for the binomial distribution:*

$$\Pr[X \geq a] = \sum_{i=a}^{n} \binom{n}{i} p^i (1-p)^{n-i} \leq p^a 2^n. \tag{5.5}$$

PROOF: (of Claim 5.3.13.)

$$\begin{aligned} &\Pr[\texttt{And}\text{-gate depends on more than } a \text{ variables}] \\ \leq &\sum_{i=a}^{4k\ln(n)} \binom{4k\ln(n)}{i} (1/\sqrt{n})^i (1-1/\sqrt{n})^{4k\ln(n)-i} \end{aligned}$$

and using (5.5),

$$\leq (1/\sqrt{n})^a 2^{4k\ln(n)} = n^{-a/2} n^{8k} = n^{8k-a/2}$$

Now letting $a = 18k$ completes the proof. □

In the induction step assume the result holds for $(c-1)$. Again, there are two cases:

1. Before the random restriction, the `And`-gate has many `Or`-gates below it with *disjoint input variables* (at least $d \cdot \ln(n)$, where $d = k \cdot 4^c$),
2. before the random restriction, the `And`-gate has few `Or`-gates below it with *disjoint input variables* (less than $d \cdot \ln(n)$).

In the first case (many `Or`-gates below `And`), it is very likely that after the random restriction one of the `Or`-gates will have all of its inputs set to 0, and so the `And`-gate is 0, and so it depends on no variables:

$$\begin{aligned} &\Pr[\ \texttt{And}\text{-gate is } \textit{not} = 0\ ] \\ \leq &\Pr[\ \text{none of the } \texttt{Or}\text{-gates is } 0\ ] \\ \leq &(\Pr[\ \text{a fixed } \texttt{Or}\text{-gate is not } 0\ ])^{d\ln n} \\ \leq &(1-4^{-c})^{d\cdot\ln n} \quad \text{for } n \geq 4 \\ \leq &n^{d\cdot\ln(1-4^{-c})} \\ \leq &n^{-d\cdot 4^{-c}} \quad \text{since } \ln(1-x) \leq -x \\ = &n^{-k}. \end{aligned}$$

In the second case (few `Or`-gates below `And`), choose a *maximal* set of `Or`-gates with *disjoint input variables*. Let H be the set of the variables that occur in these `Or` gates.

For example, in the circuit depicted in Figure 5.6, a maximal set of `Or`s would consist of gates (numbered from left) $\{1, 3, 4\}$ and $H = \{x, y, u, z, v\}$.

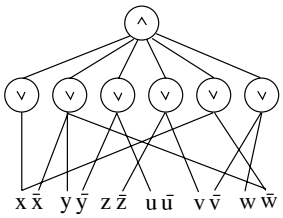

Figure 5.6: Few `Or` gates below `And`.

Since the `Or`-gates' fan-in is c, we know that $|H| \leq c \cdot d \cdot \ln(n)$. Let $\{\tau_1, \tau_2, \ldots, \tau_l\}$, $l = 2^{|H|}$, be all the truth assignments to variables in H. Let $A_i = C^{\tau_i}$, after simplifying, where C represents the whole `And-Or` circuit. Since each of the `Or`-gates has a variable in H, each `Or`-gate in A_i either disappears or "loses" an input. So the input fan-in in each A_i is at most $(c-1)$. By the induction hypothesis, the probability that A_i^r depends on more than e_{c-1} variables is bounded above by $O(\frac{1}{n^k})$.

Consider the Boolean function f_C,

$$f_C = \bigvee_{i=1}^{l} \text{clause}_H(\tau_i) \wedge A_i. \tag{5.6}$$

where $\text{clause}_H(\tau_i)$ is a conjunction of literals over variables in H, in such a way that if $x \in H$, and $\tau_i(x) = 0$, then x appears as $\bar{x}$, and if $\tau_i(x) = 1$, then x appears as x. Therefore, the probability that f_C depends on more than $l \cdot e_{c-1}$ many variables is bounded above by $l \cdot O(\frac{1}{n^k})$. So, if we can show that with high probability $|H|$ is constant, we will have that with high probability l is constant as well, and then the claim follows.

Let h be the number of variables in H after a random restriction. We show that:

$$\Pr[\ h > 4cd + 2k\ ] = O\left(\frac{1}{n^k}\right).$$

To see this, note that $|H| \leq cd\ln(n)$, so once again using (5.5) we get that

$$\Pr[h > a] \leq 2^{cd\ln(n)} \cdot \left(\frac{1}{\sqrt{n}}\right)^a \leq n^{2cd} \cdot n^{-a/2} = n^{2cd-a/2},$$

and solving for a in $2cd - a/2 = k$ we get that $a = 4cd + 2k$.

Thus, with high probability (which always means $O(1 - \frac{1}{n^k})$) $2^h \leq 2^{4cd+2k}$. Hence, with high probability, $l \leq 2^{4cd+2k}$ in (5.6), and so, with high probability,

$$e_c = 2^{4cd+2k} \cdot e_{c-1} + (4cd + 2k),$$

where e_c is the constant bounding the number of variables on which f_C depends.

5.3.2 Second proof of Parity $\notin \mathsf{AC}^0$

We are going to give a different proof of PARITY $\notin \mathsf{AC}^0$, which also uses the probabilistic argument, but with a dash of linear algebra, and employing the idea of arithmetization that was introduced in section 3.3.2 (see page 35).

The gate $\texttt{And}(x_1, \ldots, x_n)$ can be represented by the multivariable polynomial $\Pi_{i=1}^n x_i = x_1 x_2 \ldots x_n$, over $\mathbb{Z}$. Using $(1 - x)$ to represent `Not`, and de Morgan laws, we give the polynomial representation of the `Or` function: $1 - \Pi_{i=1}^n (1 - x_i)$. Note that the polynomials thus obtained have degree n; we are interested in lowering this degree, and to that end we use the probabilistic method. We construct a random polynomial as follows: let $S_0 = \{1, \ldots, n\}$. Let $S_i \supseteq S_{i+1}$, for $i \in \{0, \ldots, \log(n) + 1\}$, where S_{i+1} is chosen randomly so that for all $j \in S_i$, $\Pr[j \in S_{i+1}] = \frac{1}{2}$.

Let $q_i = \sum_{j \in S_i} x_j$ be a random polynomial (of degree 1)[3].

Let $p = \Pi_{i=0}^{\log(n)+2}(1 - q_i)$, so it is a polynomial of degree $O(\log(n))$.

Claim 5.3.15 *If* $\texttt{Or}(x_1, \ldots, x_n) = 0$, *then* $1 - p = 0$.

PROOF: If $\texttt{Or}(x_1, \ldots, x_n) = 0$, then $x_1 = \cdots = x_n = 0$, so $q_i = 0$ for all i, so $1 - q_i = 1$ for all i, so their product p is 1, so $1 - p = 0$. □

[3]The degree of a multivariate polynomial is calculated by writing it out as a sum of monomials—unique up to order of summation—and find the monomial with the highest degree, where the degree of a monomial $(x_1^{a_1} \cdots x_l^{a_l})$ is $a_1 + \cdots + a_l$.

Claim 5.3.16 *If* $\texttt{Or}(x_1, \ldots, x_n) = 1$*, then there is at least one* $x_i = 1$*. In this case, the probability is* $\geq \frac{1}{2}$ *that one of the polynomials* q_i *has the value exactly 1, and so* $\Pr[1 - p = 1] \geq \frac{1}{2}$*.*

PROOF: Let T be a set of variables of an `Or` that are set to true; we need to argue that for any choice of a non-empty $T \subseteq S_0$, the probability is at least $\frac{1}{2}$ that there is at least one $i \in \{0, 1, \ldots, \log(n) + 2\}$ such that the size of $T \cap S_i$ is exactly 1. We accomplish this by considering the probability of two cases.

Case 1. For all $i \in \{0, 1, \ldots, \log(n) + 2\}$, we have that $|T \cap S_i| > 1$.

Since the S_i form a non-ascending chain of subsets, it follows that for all i, $|T \cap S_i| > 1$ iff $|T \cap S_{\log(n)+2}| > 1$, which is true if at least two variables "survive" all the way from S_0 to $S_{\log(n)+2}$.

The probability of this happening is bounded by $\binom{n}{2} 4^{-(\log(n)+2)} < \frac{1}{16}$, since there are $\binom{n}{2}$ ways to choose a pair of variables, and then there is a probability of $\frac{1}{2^{\log(n)+2}}$ that each "survives" until the end. Note that we could have given a tighter bound, since we are overcounting (these pairs intersect). We could have accomplished that with the *Inclusion-Exclusion Principle*, but we do not need such tight bounds in this case.

In fact, the following (even coarser) bound will do just fine for us:

$$\Pr[|T \cap S_{\log(n)+2}| > 1] \leq \Pr[|T \cap S_{\log(n)+2} \geq 1] \leq \binom{n}{1} 2^{-(\log(n)+2)} = \frac{1}{4}.$$

Case 2. There is an $i \in \{0, 1, \ldots, \log(n) + 2\}$ with $|T \cap S_i| \leq 1$.

Suppose that $|T \cap S_0| = 1$; then we are set. Otherwise, $|T \cap S_0| = |T| > 1$, and let i be such that $|T \cap S_{i-1}| > 1$ and $|T \cap S_i| \leq 1$. Let $t = |T \cap S_{i-1}|$. The probability that $|T \cap S_i| = 1$ *under the assumption* that $t > 1$ and $|T \cap S_i| \leq 1$ is given by

$$\frac{t2^{-t}}{2^{-t} + t2^{-t}} = \frac{t}{t+1} \geq \frac{2}{3}$$

since $t2^{-t}$ is the probability that one of the t variables "survives", and 2^{-t} is the probability that none of the t variables "survive."

We are now going to put case 1 and 2 together, to obtain a lower bound for $\Pr[\exists i \text{ s.t. } |T \cap S_i| = 1]$. Case 1 does not occur with probability $\frac{3}{4}$, and in case 2 we get what we want with probability $\geq \frac{2}{3}$, and multiplying the two

values[4] we obtain $\geq \frac{1}{2}$. □

So the polynomial $(1-p)$ approximates the `Or`, but with a success rate of only $\frac{1}{2}$. But we can improve this by selecting independent polynomials $p_1, p_2, \ldots, p_t$, and then using $P = 1 - \Pi_{i=1}^{t} p_i$, which has degree $O(t \log(n))$. The error probability of P is $\geq \frac{1}{2^t}$. To get the error probability below a given constant ε, we want $\frac{1}{2^t} < \varepsilon$, so $t > \log(\varepsilon^{-1})$.

The polynomial for the $\texttt{And}(x_1, \ldots, x_n)$ is just the product of the p_i's with x_j replaced by $(1 - x_j)$, so it is just the dual case of the analysis of the `Or`.

We want to simulate an AC^0 circuit family of size $s = s(n)$ and depth d using our polynomials, so that the error probability is at most ε. We replace all the gates with our polynomials, making sure that the error probability for each gate is $\leq \frac{\varepsilon}{s}$. The degree of the polynomial approximating the `And` and `Or` gates with error $\leq \frac{\varepsilon}{s}$ is $O(\log(\frac{s}{\varepsilon}) \log(n))$. So the degree of the polynomial approximating such a circuit of depth d is $O(\log^d(\frac{s}{\varepsilon}) \log^d(n)) = O(\log^{2d}(\frac{s}{\varepsilon}))$, since $s = s(n)$ is a polynomial in n.

Thus, for every $f \in \mathsf{AC}^0$, we can generate a polynomial p of polylogarithmic degree (in n), with the property that for any $(a_1, \ldots, a_n) \in \{0,1\}^n$ the probability is at least $(1 - \varepsilon)$ that $f(a_1, \ldots, a_n) = p(a_1, \ldots, a_n)$.

Claim 5.3.17 *There exists polynomial $\mathcal{P}$ s.t. $f(a_1, \ldots, a_n) = \mathcal{P}(a_1, \ldots, a_n)$ for all $(a_1, \ldots, a_n)$ in some S, where $|S| \geq (1-\varepsilon)2^n$.*

Proof: For a random polynomial thus generated, the expectation of the number of inputs for which it computes the circuit correctly is $(1-\varepsilon)2^n$. There has to be a polynomial $\mathcal{P}$ that meets this expectation. □

Identify true with -1 and false with $+1$. The linear function that maps 0 to 1 and 1 to -1 is $x \mapsto 1 - 2x$; its inverse is $x \mapsto \frac{(1-x)}{2}$. Apply this linear function to the polynomial $\mathcal{P}$ that correctly simulates f on $(1-\varepsilon)2^n$-many

[4] This is a little bit more subtle than that. You can apply the rule $\Pr[A \wedge B] = \Pr[A] \cdot Pr[B]$ only if the events are independent. In case they are dependent, you can apply the rule (which is *always* valid): $\Pr[A \wedge B] = \Pr[A] \cdot \Pr[B|A]$ where $\Pr[B|A]$ is the conditional probability (i.e., what is the probability of B taking place, given that A has taken place). Let A be the event: "not for all i, $|T \cap S_i| > 1$", and we know from Case 1. that $\Pr[A] > (3/4)$. Let B be the event: "for some i, $|T \cap S_i| = 1$", and note that in case 2, we are doing our analysis of B taking place, *under* the assumption that event A has taken place, i.e., we computed a lower bound for $\Pr[B|A]$, i.e., $\Pr[B|A] > (2/3)$. In fact, our events A and B are *not* independent events; B is a "subset" of event A. It follows that $\Pr[B] = \Pr[A \wedge B] = \Pr[A] \cdot \Pr[B|A] > (3/4)(2/3) = (1/2)$.

inputs to obtain a polynomial $\mathcal{Q}$ of the same degree—which now correctly simulates f transformed to use $\{-1,+1\}^n$, again over $(1-\varepsilon)2^n$-many inputs.

Suppose that the parity function is in AC^0. Then there must be such a polynomial $\mathcal{Q}$ for parity. For $(1-\varepsilon)2^n$-many input strings (in $\{-1,+1\}^n$), $\mathcal{Q}(y_1,\ldots,y_n) = \Pi_{i=1}^n y_i$, i.e., $\mathcal{Q}$ corresponds exactly to multiplication: if the number of 1s is odd, $\mathcal{Q}$ is -1, and if it is even it is $+1$.

Claim 5.3.18 *There is no polynomial $\mathcal{Q}$ of degree $\frac{\sqrt{n}}{2}$ that correctly represents the function $\Pi_{i=1}^n y_i$ for $(1-\varepsilon)2^n$ strings in $\{-1,+1\}^n$.*

PROOF: We first give the proof outline, and then fill in the details. Let

$$S = \{(y_1,\ldots,y_n) \in \{-1,+1\}^n | \Pi_{i=1}^n y_i = \mathcal{Q}(y_1,\ldots,y_n)\},$$

where $\mathcal{Q}$ is a polynomial of degree $\frac{\sqrt{n}}{2}$ that correctly represents $\Pi_{i=1}^n y_i$ on $(1-\varepsilon)2^n$ many strings in $\{-1,+1\}^n$. Thus, $|S| \geq (1-\varepsilon)2^n$. We can assume that $\mathcal{Q}$ is *multilinear*, that is, no variable has exponent larger than 1. Let $L(S)$ be the vector space (over $\mathbb{R}$) of functions $f : S \longrightarrow \mathbb{R}$. The dimension of $L(S)$, $\dim(L(S))$, is $|S|$. On the other hand, let POL be the set of n-variable multi-linear polynomials of degree $\frac{(n+\sqrt{n})}{2}$. Then, POL is also a vector space (over $\mathbb{R}$), with the usual polynomial addition and multiplication by scalars in $\mathbb{R}$. Then, $\dim(\text{POL})$ is $\sum_{i=0}^{\frac{(n+\sqrt{n})}{2}} \binom{n}{i}$, and using the Stirling approximation we can show that this is strictly smaller than $|S|$. On the other hand, $\dim(L(S)) \leq \dim(\text{POL})$; contradiction.

We now provide some details. The natural basis for $L(S)$ is B given by the set of functions $f_s : S \longrightarrow \mathbb{R}$ where $f_s(s) = 1$ and $f_s(s') = 0$ for $s' \neq s$. Now, any function in $L(S)$ can be written as a linear combination of functions in B.

Note that any f_s can be represented by an n-degree multi-variate multi-linear polynomial as follows:

$$f_{(s_1,\ldots,s_n)}(x_1,x_2,\ldots,x_n) \mapsto \frac{1}{2^n}(1-s_1\cdot x_1)(1-s_2\cdot x_2)\cdots(1-s_n\cdot x_n),$$

and the expression on the right-hand side can be written out as a sum of monomials. Each such monomial (without the constant coefficient multiplying it in front) is of the form $x_{i_1}x_{i_2}\cdots x_{i_k}$ with $k \leq n$.

Now consider such monomials in the representation of a given f_s. If a monomial has at most $n/2$ many variables, i.e., $k \leq n/2$, then leave it as it

is. Otherwise, $k > n/2$, and replace this monomial by the polynomial

$$g = \mathcal{Q}(x_1, x_2, \ldots, x_n) x_{j_1} x_{j_2} \cdots x_{j_{n-k}}$$

where $\{x_{j_1}, x_{j_2}, \ldots, x_{j_{n-k}}\} = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}^c$. Two observations about the polynomial g: first, it is of degree $(\sqrt{n}+n)/2$, and second, on S it takes on the same values as the original monomial.

Thus, we just showed that any f_s can be represented with a multi-variate multi-linear polynomial of degree at most $(\sqrt{n}+n)/2$. We are not quite done yet; to show that $\dim(L(S)) \leq \dim(\text{POL})$, we need to show that this mapping ($f_s \stackrel{h}{\mapsto} p \in \text{POL}$, where we extend h from basis$(L(S))$ to all of $L(S)$ in the natural way to obtain a vector space homomorphism) is such that $h(S)$ is linearly *independent*.

Suppose that $h(S)$ is linearly dependent, so there are $p_1, p_2, \ldots, p_k \in h(S)$ such that $c_1 p_1 + c_2 p_2 + \cdots + c_k p_k = 0$ (assume all $c_i \neq 0$), i.e., it is the 0 polynomial. We now examine the pre-images of these p_i's in $L(S)$, i.e., we look at $f_{s_1}, f_{s_2}, \ldots, f_{s_k}$, where $h(f_{s_i}) = p_i$ (note that these pre-images are unique, i.e., $|S| = |h(S)|$).

It follows that the function $f = c_1 f_{s_1} + c_2 f_{s_2} + \cdots + c_k f_{s_k}$ is mapped by h to the zero polynomial. It follows therefore that $f = 0$, i.e., f is the zero function. But $f(s_1) = c_1 f_{s_1}(s_1) = c_1 \neq 0$, contradiction. □

We showed that the degree of a polynomial approximating a bounded depth circuit is poly-logarithmic. On the other hand, we need degree higher than $\frac{\sqrt{n}}{2}$ to approximate parity. Thus PARITY $\notin \mathsf{AC}^0$.

5.4 Computation with advice

In this section we consider a nonuniform version of TMs which capture the nonuniformity of circuits.

Suppose that our TMs have an extra read-only input tape called the *advice tape*, and let $A(n)$ be a function mapping integers to strings in Σ^*. We say that machine M decides language L with advice $A(n)$ if $x \in L$ implies $M(x, A(|x|)) =$ "yes", and $x \notin L$ implies $M(x, A(|x|)) =$ "no". That is, the advice $A(n)$, specific to the length of the input, helps M decide all strings of length n correctly.

Let $f(n)$ be a function mapping nonnegative integers to nonnegative integers. We say that $L \in \mathcal{C}/f(n)$ if there is an advice function $A(n)$, where

$|A(n)| \leq f(n)$ for all $n \geq 0$, and a TM M running in complexity $\mathcal{C}$, such that M decides L with advice A. We write $\mathcal{C}/\text{poly}$ to indicate TMs working in complexity $\mathcal{C}$ with an advice bounded by some fixed polynomial.

Claim 5.4.1 *$L \in \mathsf{P}/\text{poly}$ iff L has polysize circuits.*

PROOF: If $L \in \mathsf{P}/n^k$ then it can be decided by a deterministic machine with advice. But then for each length n the advice is fixed, so we can encode the whole computation of this machine using a polynomial size circuit. If, on the other hand, L has polynomial circuits, we can construct the following machine with advice to decide it: treat the advice as a description of a circuit, simulate this circuit on your input, and accept iff the result was 1. It is clear that this is a polytime machine which, when given descriptions of circuits for L as the advice, decides L. □

Claim 5.4.2 *If* SAT $\in \mathsf{P}/\log(n)$, *then* $\mathsf{P} = \mathsf{NP}$.

Exercise 5.4.3 *Prove claim 5.4.2.*

Let $G = (V, E)$ be a connected undirected regular graph of degree exactly d, i.e., that for each $u \in V$, $|\{(u, v) : \text{ for some } v \in V \text{ s.t. } (u, v) \in E \}| = d$.

Assume that for any given u we have ordered the edges incident upon u in some way. A string $U = l_1 l_2 \ldots l_m \in \{1, 2, \ldots, d\}^*$ and a node u induce a traversal of the graph as follows: we start at u, and we take edge l_1 out of node u and arrive at node u_1. Then we take edge l_2 out of node u_1 and arrive at node u_2, etc. (We may visit a node more than once). We say that U traverses G if it visits every node of G.

Exercise 5.4.4 *Use a probabilistic argument to show that for each n there is a traversal sequence of length $O(d \cdot n^4)$ that works for all graphs with n nodes and degree d (i.e., a* universal traversal sequence*). What does this tell you about the complexity of undirected reachability?*

5.5 Answers to selected exercises

Exercise 5.2.2. Note that this number is a gross overestimate. We just count graphs with s nodes labeled by a gate with two inputs, not being concerned about the fact that many such graphs are not really circuits because they may contain cycles. Each gate is a $\wedge$ or a $\vee$, and its inputs are two

other nodes. Each input can be either a gate (s many choices), a literal ($2n$ choices), or a constant (2 choices). Compounding these choices for s gates gives us the result.

Exercise 5.3.9. The proof of (5.3) is:

$$E(X) = \sum_x x \cdot \Pr[X = x] \geq \sum_{x \geq a} x \cdot \Pr[X = x] \geq a \sum_{x \geq a} \Pr[X = x] = a \Pr[X \geq a].$$

Note that $\mathrm{Var}(X) = E((X - E(X))^2) = E(X^2) - (E(X))^2$ which is the average of the square of the distance of each data point from the mean (the second expression is more useful in practice). The proof of (5.4) is:

$$\Pr[|X - E(X)| \geq a] = \Pr[(X - E(X))^2 \geq a^2] \leq \frac{E((X - E(X))^2)}{a^2} = \frac{\mathrm{Var}(X)}{a^2}.$$

Exercise 5.3.11. For (i), note that $c^{\log_c(a)} = a$, so $c^{\log_c(a)+1} = ac$, so $\log_a(c^{\log_c(a)+1}) = \log_c(ac)$, so $(\log_c(a) + 1)\log_a(c) = 1 + \log_a(c)$, and so $\log_c(a)\log_a(c) = 1$. For (ii), note $a^{\log_a(b)} = b$, so $\log_c(a^{\log_a(b)}) = \log_c b$, so $\log_a(b)\log_c(a) = \log_c(b)$, and so $\log_a(b)/\log_a(c) = \log_c(b)$, where we used (i) to derive the last step.

Exercise 5.4.4. Check in [Pap94] that the probability that a fixed node v is *not* visited by a random walk of length dn^2 starting at some node u is $\leq 1/2$. If we repeat this "experiment" m times, or, equivalently, increase the random walk to dn^2m many steps, the probability of not visiting v becomes $\leq 1/2^m$. So the probability of missing some node during the random walk is $\leq n/2^m$. The number of d-regular graphs with n nodes is (grossly) bounded above by n^{dn}, so we want to choose an m such that $(n/2^m)(n^{dn}) < 1$, assuring that there is a universal traversal sequence of all d-regular graphs with n nodes. Choose $m = n^2$. This tells us two things about undirected reachability: (i) it is in RL (randomized logspace, with no false positives), and (ii) it is in L/poly, i.e., logspace with polynomial advice.

5.6 Notes

The proof of lemma 5.1.2 is based on [Pap94, Proposition 11.2]. Section 5.3.1 is based on [SP95, chapter 11], and section 5.3.2 is based on [SP95, chapter 12]. Section 5.4 is based on [Pap94, exercise 11.5.24]; originally, computation with advice was introduced in [KL80]. Claim 5.4.1 is [Pap94, exr. 11.5.24(a)]. Claim 5.4.2 is [Pap94, exr. 11.5.24(b)].