CHAPTER 9

Circuit Complexity

The circuit complexity of a binary function is measured by the size or depth of the smallest or shallowest circuit for it. Circuit complexity derives its importance from the corollary to Theorem 3.9.2; namely, if a function has a large circuit size over a complete basis of fixed fan-in, then the time on a Turing machine required to compute it is large. The importance of this observation is illustrated by the following fact. For $n \geq 1$, let $f_L^{(n)}$ be the characteristic function of an **NP**-complete language L, where $f_L^{(n)}$ has value 1 on strings of length n in L and value 0 otherwise. If $f_L^{(n)}$ has super-polynomial circuit size for all sufficiently large n, then $\mathbf{P} \neq \mathbf{NP}$.

In this chapter we introduce methods for deriving lower bounds on circuit size and depth. Unfortunately, it is generally much more difficult to derive good lower bounds on circuit complexity than good upper bounds; an upper bound measures the size or depth of a particular circuit whereas a lower bound must rule out a smaller size or depth for all circuits. As a consequence, the lower bounds derived for functions realized by circuits over complete bases of bounded fan-in are often weak.

In attempting to understand lower bounds for complete bases, researchers have studied monotone circuits over the monotone basis and bounded-depth circuits over the basis {AND, OR, NOT} in which the first two gates are allowed to have unbounded fan-in. Formula size, which is approximately the size of the smallest circuit with fan-out 1, has also been studied. Lower bounds to formula size also produce lower bounds to circuit depth, a measure of the parallel time needed for a function.

Research on these restricted circuit models has led to some impressive results. Exponential lower bounds on circuit size have been derived for monotone functions over the monotone basis and functions such as parity when realized by bounded-depth circuits. Unfortunately, the methods used to obtain these results may not apply to complete bases of bounded fan-in. Fortunately, it has been shown that the *slice functions* have about the same circuit size over both the monotone and standard (non-monotone) bases. This may help resolve the $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ question, since there are **NP**-complete slice problems.

Despite the difficulty of deriving lower bounds, circuit complexity continues to offer one of the methods of highest potential for distinguishing between **P** and **NP**.

9.1 Circuit Models and Measures

In this section we characterize types of logic circuits by their bases and the fan-in and fan-out of basis elements. We consider bases that are complete and incomplete and that have bounded and unbounded fan-in. We also consider circuits in which the fan-out is restricted and unrestricted. Each of these factors can affect the size and depth of a circuit.

9.1.1 Circuit Models

The (general) **logic circuit** is the graph of a straight-line program in which the variables have value 0 or 1 and the operations are Boolean functions $g : \mathcal{B}^p \mapsto \mathcal{B}$, $p \geq 1$. (Boolean functions have one binary value. Logic circuits are defined in Section 1.2 and discussed at length in Chapter 2.) The vertices in a logic circuit are labeled with Boolean operations and are called **gates**; the set of different gate types used in a circuit is called the **basis** (denoted Ω) for the circuit. The **fan-in of a basis** is the maximal fan-in of any function in the basis. A circuit **computes** the binary function $f : \mathcal{B}^n \mapsto \mathcal{B}^m$, which is the mapping from the n circuit inputs to the m gate outputs designated as circuit outputs.

The **standard basis**, denoted Ω_0, is the set $\{\text{AND}, \text{OR}, \text{NOT}\}$ in which AND and OR have fan-in 2. The **full two-input basis**, denoted B_2, consists of all two-input Boolean functions. The **dyadic unate basis**, denoted U_2, consists of all Boolean functions of the form $(x^a \wedge y^b)^c$ for constants a, b, c in $\mathcal{B}$. Here $x^1 = x$ and $x^0 = \overline{x}$.

A **basis** Ω is **complete** if every binary function can be computed by a circuit over Ω. The bases Ω_0, B_2, and U_2 are complete, as is the basis consisting of the NAND gate computing the function $x \text{ NAND } y = \overline{x \wedge y}$. (See Problem 2.5.)

The bounded fan-out circuit model specifies a bound on the fan-out of a circuit. As we shall see, the **fan-out-1 circuit** plays a special role related to circuit depth. Each circuit of fan-out 1 corresponds to a **formula** in which the operators are the functions associated with vertices of the circuit. Figure 9.1 shows an example of a circuit of fan-out 1 over the standard basis and its associated formula. (See also Problem 9.9.) Although each input variable appears once in this example, Boolean functions generally require multiple instances of variables (have fan-out greater than 1). Formula size is studied at length in Section 9.4.

To define the monotone circuits, we need an ordering of binary n-tuples. Two such tuples, $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$, are in the relation $\boldsymbol{x} \leq \boldsymbol{y}$ if for all $1 \leq i \leq n$, $x_i \leq y_i$, where $0 \leq 0$, $1 \leq 1$, and $0 \leq 1$, but $1 \not\leq 0$. (Thus, $001011 \leq 101111$, but $011011 \not\leq 101111$.)

A **monotone circuit** is a circuit over the monotone basis $\Omega_{\text{mon}} = \{\text{AND}, \text{OR}\}$ in which the fan-in is 2. There is a direct correspondence between monotone circuits and monotone functions. A **monotone function** is a function $f : \mathcal{B}^n \mapsto \mathcal{B}^m$ that is either **monotone increasing**, that is, for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{B}^n$, if $\boldsymbol{x} \leq \boldsymbol{y}$, then $f(\boldsymbol{x}) \leq f(\boldsymbol{y})$, or is **monotone decreasing**, that is, for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{B}^n$, if $\boldsymbol{x} \leq \boldsymbol{y}$, then $f(\boldsymbol{x}) \geq f(\boldsymbol{y})$. Unless stated explicitly, a monotone function will be understood to be a monotone increasing function.

A monotone Boolean function has the following expansion on the first variable, as the reader can show. (See Problem 9.10.) A similar expansion is possible on any variable.

$$f(x_1, x_2, \ldots, x_n) = f(0, x_2, \ldots, x_n) \vee (x_1 \wedge f(1, x_2, \ldots, x_n))$$

By applying this expansion to every variable in succession, we see that each monotone function can be realized by a circuit over the monotone basis. Furthermore, the **monotone basis** Ω_{mon}

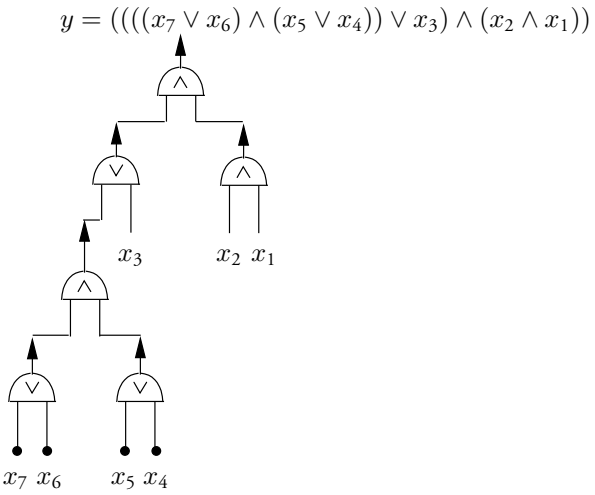


Figure 9.1 A circuit of fan-out 1 over a basis with fan-in 2 and a corresponding formula. The value y at the root is the AND of the value $(((x_7 \vee x_6) \wedge (x_5 \vee x_4)) \vee x_3)$ of the left subtree with the value $(x_2 \wedge x_1)$ of the right subtree.

is complete for the monotone functions, that is, every monotone function can be computed by a circuit over the basis Ω_{mon}. (See Problem 2.)

In Section 9.6 we show that some monotone functions on n variables require monotone circuits whose size is exponential in n. In particular, some monotone functions requiring exponential-size monotone circuits can be realized by polynomial-size circuits over the standard basis Ω_0. Thus, the absence of negation can result in a large increase in circuit size.

The **bounded-depth circuit** is a circuit over the standard basis Ω_0 where the fan-in of AND and OR gates is allowed to be unbounded, but the circuit depth is bounded. The conjunctive and disjunctive normal forms and the product-of-sums and sum-of-products normal forms realize arbitrary Boolean functions by circuits of depth 2 over Ω_0. (See Section 2.3.) In these normal forms negations are used only on the input variables. Note that any circuit over the standard basis can be converted to a circuit in which the NOT gates are applied only to the input variables. (See Problem 9.11.)

9.1.2 Complexity Measures

We now define the measures of complexity studied in this chapter. The **depth of a circuit** is the number of gates of fan-in 2 or more on the longest path in the circuit. (Note that NOT gates do not affect the depth measure.)

DEFINITION 9.1.1 *The* **circuit size** *of a binary function $f : \mathcal{B}^n \mapsto \mathcal{B}^m$ with respect to the basis Ω, denoted $C_\Omega(f)$, is the smallest number of gates in any circuit for f over the basis Ω. The* **circuit size with fan-out s**, *denoted $C_{s,\Omega}(f)$, is the circuit size of f when the circuit fan-out is limited to at most s.*

The **circuit depth** *of a binary function $f : \mathcal{B}^n \mapsto \mathcal{B}^m$ with respect to the basis Ω, $D_\Omega(f)$, is the depth of the smallest depth circuit for f over the basis Ω. The* **circuit depth with fan-out s**, *denoted $D_{s,\Omega}(f)$, is the circuit depth of f when the circuit fan-out is limited to at most s.*

The **formula size** *of a Boolean function $f : \mathcal{B}^n \mapsto \mathcal{B}$ with respect to a basis Ω, $L_\Omega(f)$, is the minimal number of input vertices in any circuit of fan-out 1 for f over the basis Ω.*

It is important to note the distinction between formula and circuit size: in the former the number of input vertices is counted, whereas in the latter it is the number of gates. A relationship between the two is shown in Lemma 9.2.2.

9.2 Relationships Among Complexity Measures

In this section we explore the effect on circuit complexity measures of a change in either the basis or the fan-out of a circuit. We also establish relationships between circuit depth and formula size.

9.2.1 Effect of Fan-Out on Circuit Size

It is interesting to ask how the circuit size and depth of a function change as the maximal fan-out of a circuit is reduced. This issue is important in understanding these complexity measures and in the use of technologies that limit the fan-out of gates. The following simple facts about trees are useful in comparing complexity measures. (See Problem 9.2.)

LEMMA 9.2.1 *A rooted tree of maximal fan-in r containing k vertices has at most $k(r-1)+1$ leaves and a rooted tree with l leaves and fan-in r has at most $l-1$ vertices with fan-in 2 or more and at most $2(l-1)$ edges.*

From the above result we establish the following connection between circuit size with fan-out 1 and formula size.

LEMMA 9.2.2 *Let Ω be a basis of fan-in r. For each $f : \mathcal{B}^n \mapsto \mathcal{B}$ the following inequalities hold between formula size, $L_\Omega(f)$, and fan-out-1 circuit size, $C_{1,\Omega}(f)$:*

$$(L_\Omega(f)-1)/(r-1) \leq C_{1,\Omega}(f) \leq 3L_\Omega(f)-2$$

Proof The first inequality follows from the definition of formula size and the first result stated in Lemma 9.2.1 in which $k = C_{1,\Omega}(f)$. The second inequality also follows from Lemma 9.2.1. A tree with $L_\Omega(f)$ leaves has at most $L_\Omega(f)-1$ vertices with fan-in of 2 or more and at most $2(L_\Omega(f)-1)$ edges between vertices (including the leaves). Each of these edges can carry a NOT gate, as can the output gate, for a total of at most $2L_\Omega(f)-1$ NOT gates. Thus, a circuit of fan-out 1 has at most $3L_\Omega(f)-2$ gates. ■

As we now show, circuit size increases by at most a constant factor when the fan-out of the circuit is reduced to s for $s \geq 2$. Before developing this result we need a simple fact about a complete basis Ω, namely, that at most two gates are needed to compute the **identity function** $i(x) = x$, as shown in the next paragraph. If a basis contains AND or OR gates, the identity function can be obtained by attaching both of their inputs to the same source.

We are done if Ω contains a function such that by fixing all but one variable, $i(x)$ is computed. If not, then we look for a non-monotone function in Ω. Since some binary

functions are non-monotone ($\overline{x}$, for example), some function g in a complete basis Ω is non-monotone. This means there exist tuples $\boldsymbol{x}$ and $\boldsymbol{y}$ for g, $\boldsymbol{x} \leq \boldsymbol{y}$, such that $g(\boldsymbol{x}) = 1 > g(\boldsymbol{y}) = 0$. Let $\boldsymbol{u}$ and $\boldsymbol{v}$ be the largest and smallest tuples, respectively, satisfying $\boldsymbol{x} \leq \boldsymbol{u} \leq \boldsymbol{v} \leq \boldsymbol{y}$ and $g(\boldsymbol{u}) = 1$ and $g(\boldsymbol{v}) = 0$. Then $\boldsymbol{u}$ and $\boldsymbol{v}$ differ in at most one position. Without loss of generality, let that position be the first and let the values in the remaining positions in both tuples be $(c_2, \ldots, c_n)$. It follows that $g(1, c_2, \ldots, c_n) = 0$ and $g(0, c_2, \ldots, c_n) = 1$ or $g(x, c_2, \ldots, c_n) = \overline{x}$. If $l(\Omega)$ is the number of gates from Ω needed to realize the identity function, then $l(\Omega) = 1$ or 2.

THEOREM 9.2.1 *Let Ω be a complete basis of fan-in r and let $f : \mathcal{B}^n \mapsto \mathcal{B}^m$. The following inequalities hold on $C_{s,\Omega}(f)$:*

$$C_\Omega(f) \leq C_{s+1,\Omega}(f) \leq C_{s,\Omega}(f) \leq C_{1,\Omega}(f)$$

Furthermore, $C_{s,\Omega}(f)$ has the following relationship to $C_\Omega(f)$ for $s \geq 2$:

$$C_{s,\Omega}(f) \leq C_\Omega(f) \left(1 + \frac{l(\Omega)(r-1)}{s-1}\right)$$

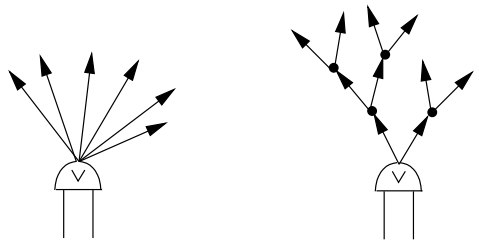
Proof The first set of inequalities holds because a smallest circuit with fan-out s is no smaller than a smallest circuit with fan-out $s+1$, a less restrictive type of circuit.

The last inequality follows by constructing a tree of identity functions at each gate whose fan-out exceeds s. (See Fig. 9.2.) If a gate has fan-out $\phi > s$, reduce the fan-out to s and then attach an identity gate to one of these s outputs. This increases the fan-out from s to $s + s - 1$. If ϕ is larger than this number, repeat the process of adding an identity gate k times, where k is the smallest integer such that $s + k(s-1) \geq \phi$ or is the largest integer such that $s + (k-1)(s-1) < \phi$. Thus, $k < (\phi - 1)/(s-1)$.

Let ϕ_i denote the fan-out of the ith gate in a circuit for f of potentially unbounded fan-out and let k_i be the largest integer satisfying the following bound:

$$k_i < \frac{\phi_i - 1}{s-1}$$

Then at most $\sum_i (k_i l(\Omega) + 1)$ gates are needed in the circuit of fan-out s to realize f, one for the ith gate in the original circuit and $k_i l(\Omega)$ gates for the k_i copies of the identity



(a) (b)

Figure 9.2 Conversion of a vertex with fan-out more than s to a subtree with fan-out s, illustrated for $s = 2$.

function at the ith gate. Note that $\sum_i \phi_i$ is the number of edges directed away from gates in the original circuit. But since each edge directed away from a gate is an edge directed into a gate, this number is at most $rC_\Omega(f)$ since each gate has fan-in at most r.

It follows that the smallest number of gates in a circuit with fan-out s for f satisfies the following bound:

$$C_{s,\Omega}(f) \leq C_\Omega(f) + l(\Omega) \sum_{i=1}^{C_\Omega(f)} \left(\frac{\phi_i - 1}{s-1} \right) \leq C_\Omega(f) \left(1 + \frac{l(\Omega)(r-1)}{s-1} \right)$$

which demonstrates that circuit size with a fan-out $s \geq 2$ differs from the unbounded fan-out circuit size by at most a constant factor. ■

With the construction employed in Theorem 9.2.1, an upper bound can be stated on $D_{s,\Omega}(f)$ that is proportional to the product of $D_\Omega(f)$ and $\log C_\Omega(f)$. (See Problem 9.12.) The upper bound stated above on $C_{s,\Omega}(f)$ can be achieved by a circuit that also achieves an upper bound on $D_{s,\Omega}(f)$ that is proportional to $D_\Omega(f)$ and $\log_r s$ [138].

9.2.2 Effect of Basis Change on Circuit Size and Depth

We now consider the effect of a change in basis on circuit size and depth. In the next section we examine the relationship between formula size and depth, from which we deduce the effect of a basis change on formula size.

LEMMA 9.2.3 *Given two complete bases, Ω_a and Ω_b, and a function $f : \mathcal{B}^n \mapsto \mathcal{B}^m$, the circuit size and depth of f in these two bases differ by at most constant multiplicative factors.*

Proof Because each basis is complete, every function in Ω_a can be computed by a fixed number of gates in Ω_b, and vice versa. Given a circuit with basis Ω_a, a circuit with basis Ω_b can be constructed by replacing each gate from Ω_a by a fixed number of gates from Ω_b. This has the effect of increasing the circuit size by at most a constant factor. It follows that $C_{\Omega_a}(f) = \Theta(C_{\Omega_b}(f))$. Since this construction also increases the depth by at most a constant factor, it follows that $D_{\Omega_a}(f) = \Theta(D_{\Omega_b}(f))$. ■

9.2.3 Formula Size Versus Circuit Depth

A logarithmic relationship exists between the formula size and circuit depth of a function, as we now show. If a formula is represented by a balanced tree, this result follows from the fact that the circuit fan-in is bounded. However, since we cannot guarantee that each formula corresponds to a balanced tree, we must find a way to balance an unbalanced tree.

To balance a formula and provide a bound on the circuit depth of a function in terms of formula size, we make use of the multiplexer function $f^{(n)}_{\text{mux}} : \mathcal{B}^{2^n+n} \mapsto \mathcal{B}$ on three inputs $f^{(1)}_{\text{mux}}(a, y_1, y_0)$. Here the value of a determines which of the two other values is returned.

$$f^{(1)}_{\text{mux}}(a, y_1, y_0) = \begin{cases} y_0 & a = 0 \\ y_1 & a = 1 \end{cases}$$

This function can be realized by

$$f^{(1)}_{\text{mux}}(a, y_1, y_0) = (\overline{a} \wedge y_0) \vee (a \wedge y_1)$$

The measure $d(\Omega)$ of a basis Ω defined below is used to obtain bounds on the circuit depth of a function in terms of its formula size.

DEFINITION 9.2.1 *Given a basis Ω of fan-in r, the constant $d(\Omega)$ is defined as follows:*

$$d(\Omega) = \left(D_\Omega\left(f^{(1)}_{\text{mux}}\right) + 1\right) / \log_r \left(\frac{r+1}{r}\right)$$

Over the standard basis Ω_0, $d(\Omega_0) = 3.419$.

We now derive a **separator theorem for trees**. This is a theorem stating that a tree can be decomposed into two trees of about the same size by removing one edge. We begin by establishing a property about trees that implies the separator theorem.

LEMMA 9.2.4 *Let T be a tree with n internal (non-leaf) vertices. If the fan-in of every vertex of T is at most r, then for any k, $1 \leq k \leq n$, T has a vertex v such that the subtree T_v rooted at v has at least k leaves but each of its children $T_{v_1}, T_{v_2}, \ldots, T_{v_p}$, $p \leq r$, has fewer than k leaves.*

Proof If the property holds at the root, the result follows. If not, move to some subtree of T that has at least k leaves and apply the test recursively. Because a leaf vertex has one leaf vertex in its subtree, this process terminates on some vertex v at which the property holds. If it terminates on a leaf vertex, each of its children is an empty tree. ■

COROLLARY 9.2.1 *Let T be a tree of fan-in r with n leaves. Then T has a subtree T_v rooted at a vertex v such that T_v has at least $\lceil n/(r+1) \rceil$ leaves but at most $\lfloor rn/(r+1) \rfloor$.*

Proof Let v be the vertex of Lemma 9.2.4 and let $k = \lceil n/(r+1) \rceil$. Since T_v has at most r subtrees each containing no more than $\lceil n/(r+1) \rceil - 1 \leq n/(r+1)$ leaves, the result follows. ■

We now apply this decomposition of trees to develop bounds on formula size.

THEOREM 9.2.2 *Let Ω be a complete basis of fan-in r. Any function $f : \mathcal{B}^n \mapsto \mathcal{B}$ with formula size $L_\Omega(f) \geq 2$ has circuit depth $D_\Omega(f)$ satisfying the following bounds:*

$$\log_r L_\Omega(f) \leq D_\Omega(f) \leq d(\Omega) \log_r L_\Omega(f)$$

Proof The lower bound follows because a rooted tree of fan-in r with depth d has at most r^d leaves. Since $L_\Omega(f)$ leaves are needed to compute f with a tree circuit over Ω, the result follows directly.

The derivation of the upper bound is by induction on formula size. We first establish the basis for induction: that $D_\Omega(f) \leq d(\Omega) \log_r L_\Omega(f)$ for $L_\Omega(f) = 2$. To show this, observe that any function f with $L_\Omega(f) = 2$ depends on at most two variables. There are 16 functions on two variables (which includes the functions on one variable), of which 10 have the property that both variables affect the output. Each of these 10 functions can be realized from a circuit for $f^{(1)}_{\text{mux}}$ by adding at most one NOT gate on one input and one NOT on the output. (See Problem 9.13.) But, as seen from the discussion preceding Theorem 9.2.1, every complete basis contains a non-monotone function all but one of whose inputs can be fixed so that the functions computes the NOT of its one remaining input. Thus, a circuit with depth $D_\Omega\left(f^{(1)}_{\text{mux}}\right) + 2$ suffices to realize a function with $L_\Omega(f) = 2$.

The basis for induction is that $D_\Omega\left(f_{\text{mux}}^{(1)}\right) + 2 \leq d(\Omega)\log_r L_\Omega(f)$ for $L_\Omega(f) = 2$, which we now show.

$$\begin{aligned} d(\Omega)\log_r L_\Omega(f) &= \left(D_\Omega\left(f_{\text{mux}}^{(1)}\right) + 1\right)(\log_r 2)/\log_r\left(\frac{r+1}{r}\right) \\ &= \left(D_\Omega\left(f_{\text{mux}}^{(1)}\right) + 1\right)/\log_2\left(\frac{r+1}{r}\right) \\ &\geq 1.7\left(D_\Omega\left(f_{\text{mux}}^{(1)}\right) + 1\right) \geq D_\Omega(f_{\text{mux}}^{(1)}) + 2 \end{aligned}$$

since $(r+1)/r \leq 1.5$ and $D_\Omega\left(f_{\text{mux}}^{(1)}\right) \geq 1$.

The inductive hypothesis is that any function f with a formula size $L_\Omega(f) \leq L_0 - 1$ can be realized by a circuit with depth $d(\Omega)\log_r L_\Omega(f)$.

Let T be the tree associated with a formula for f of size L_0. The value computed by T can be computed from the function $f_{\text{mux}}^{(1)}$ using the values produced by three trees, as suggested in Fig. 9.3. The tree T_v of Corollary 9.2.1 and two copies of T from which T_v has been removed and replaced by 0 in one case (the tree T_0) and 1 in the other (the tree T_1) are formed and the value of T_v is used to determine which of T_0 and T_1 is the value T. Since T_v has at least $\lceil L_0/(r+1)\rceil$ and at most $\lfloor rL_0/(r+1)\rfloor \leq L_0 - 1$ leaves, each of T_0 and T_1 has at most $L_0 - \lceil L_0/(r+1)\rceil = \lfloor rL_0/(r+1)\rfloor$ leaves. (See Problem 9.1.) Thus, all trees have at most $\lfloor rL_0/(r+1)\rfloor \leq L_0 - 1$ leaves and the inductive hypothesis applies. Since the depth of the new circuit is the depth of $f_{\text{mux}}^{(1)}$ plus the maximum of the depths of the three trees, f has the following depth bound:

$$D_\Omega(f) \leq D_\Omega\left(f_{\text{mux}}^{(1)}\right) + d(\Omega)\log_r \frac{rL_\Omega(f)}{(r+1)}$$

The desired result follows from the definition of $d(\Omega)$. ■

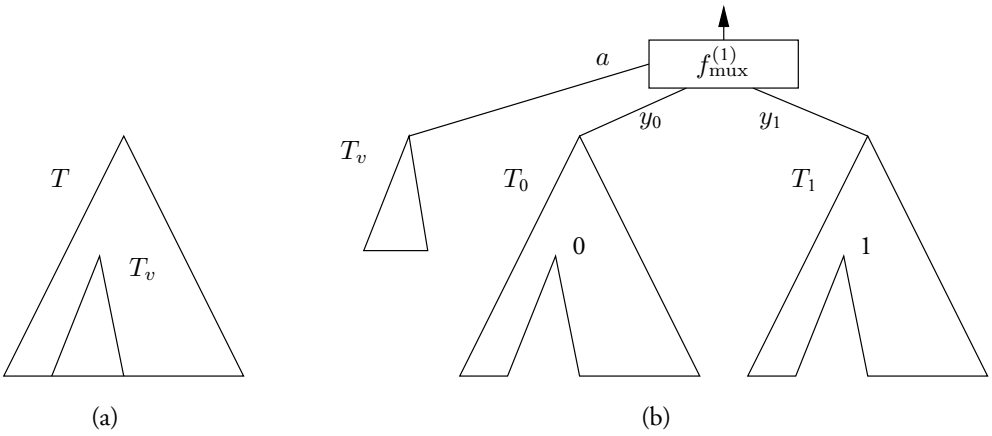


Figure 9.3 Decomposition of a tree circuit T for the purpose of reducing its depth. A large subtree T_v is removed and its value used to select the value computed by two trees formed from the original tree by replacing the value of T_v alternately by 0 and 1.

Combining this result with Lemma 9.2.3, we obtain a relationship between the formula sizes of a function over two different complete bases.

THEOREM 9.2.3 *Let Ω_a and Ω_b be two complete bases with fan-in r_a and r_b, respectively. There is a constant α such that the formula size of a function $f : \mathcal{B}^n \mapsto \mathcal{B}$ with respect to these bases satisfies the following relationship:*

$$L_{\Omega_a}(f) \leq [L_{\Omega_b}(f)]^{\alpha}$$

Proof Let $D_{\Omega_a}(f)$ and $D_{\Omega_b}(f)$ be the depth of f over the bases Ω_a and Ω_b, respectively. From Theorem 9.2.2, $\log_{r_a} L_{\Omega_a}(f) \leq D_{\Omega_a}(f)$ and $D_{\Omega_b}(f) \leq d(\Omega_b) \log_{r_b} L_{\Omega_b}(f)$.

From Lemma 9.2.3 we know there is a constant $d_{a,b}$ such that if a function $f : \mathcal{B}^n \mapsto \mathcal{B}$ has depth $D_{\Omega_b}(f)$ over the basis Ω_b, then it has depth $D_{\Omega_a}(f)$ over the basis Ω_a, where

$$D_{\Omega_a}(f) \leq d_{a,b} D_{\Omega_b}(f)$$

The constant $d_{a,b}$ is the depth of the largest-depth basis element of Ω_b when realized by a circuit over Ω_a.

Combining these facts, we have that

$$\begin{aligned} L_{\Omega_a}(f) &\leq (r_a)^{D_{\Omega_a}(f)} \leq (r_a)^{d_{a,b} D_{\Omega_b}(f)} \\ &\leq (r_a)^{d_{a,b} d(\Omega_b) \log_{r_b} L_{\Omega_b}(f)} \\ &\leq L_{\Omega_b}(f)^{d_{a,b} d(\Omega_b) (\log_{r_b} r_a)} \end{aligned}$$

Here we have used the identity $x^{\log_y z} = z^{\log_y x}$. ■

This result can be extended to the monotone basis. (See Problem 9.14.) We now derive a relationship between circuit size and depth.

9.3 Lower-Bound Methods for General Circuits

In Chapter 2 upper bounds were derived for a variety of functions, including logical, arithmetic, shifting, and symmetric functions as well as encoder, decoder, multiplexer, and demultiplexer functions. We also established lower bounds on size and depth of the most complex Boolean functions on n variables. In this section we present techniques for deriving lower bounds on circuit size and depth for particular functions when realized by general logic circuits.

9.3.1 Simple Lower Bounds

A function $f : \mathcal{B}^n \mapsto \mathcal{B}$ on n variables is **dependent on its ith variable**, x_i, if there exist values $c_1, c_2, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n$ such that

$$f(c_1, c_2, \ldots, c_{i-1}, 0, c_{i+1}, \ldots, c_n) \neq f(c_1, c_2, \ldots, c_{i-1}, 1, c_{i+1}, \ldots, c_n)$$

This simple property leads to lower bounds on circuit size and depth that result from the connectivity that a circuit must have to compute a function depending on each of its variables.

THEOREM 9.3.1 *Let $f : \mathcal{B}^n \mapsto \mathcal{B}$ be dependent on each of its n variables. Then over each basis Ω of fan-in r, the size and depth of f satisfies the following lower bounds:*

$$C_\Omega(f) \geq \left\lceil \frac{n-1}{r-1} \right\rceil$$
$$D_\Omega(f) \geq \lceil \log_r n \rceil$$

Proof Consider a circuit of size $C_\Omega(f)$ for f. Since it has fan-in r, it has at most $rC_\Omega(f)$ edges between gates. After we show that this circuit also has at least $C_\Omega(f) + n - 1$ edges, we observe that $rC_\Omega(f) \geq C_\Omega(f) + n - 1$, from which the conclusion follows.

Since f depends on each of its n variables, there must be at least one edge attached to each of them. Similarly, because the circuit has minimal size there must be at least one edge attached to each of the $C_\Omega(f)$ gates except possibly for the output gate. Thus, the circuit has at least $C_\Omega(f) + n - 1$ edges and the conclusion follows.

The depth lower bound uses the fact that a circuit with depth d and fan-in r with the largest number of inputs is a tree. Such trees have at most r^d leaves (input vertices). Because f depends on each of its variables, a circuit for f of depth d has at least n and at most r^d leaves, from which the depth lower bound follows. ■

This lower bound is the best possible given the information used to derive it. To see this, observe that the function $f(x_1, x_2, \ldots, x_n) = x_1 \wedge x_2 \wedge \cdots \wedge x_n$, which depends on each of its variables, has circuit size $\lceil (n-1)/(r-1) \rceil$ and depth $\lceil \log_r n \rceil$ over the basis containing the r-input AND gate. (See Problem 9.15.)

9.3.2 The Gate-Elimination Method for Circuit Size

The search for methods to derive large lower bounds on circuit size for functions over complete bases has to date been largely unsuccessful. The largest lower bounds on circuit size that have been derived for explicitly defined functions are linear in n, the number of variables on which the functions depend. Since most Boolean functions on n variables have exponential size (see Theorem 2.12.1), functions do exist that have high complexity. Unfortunately, this fact doesn't help us to show that any particular problem has high circuit size. In particular, it does not help us to show that **P** $\neq$ **NP**.

In this section we introduce the gate-elimination method for deriving linear lower bounds. When applied with care, it provides the strongest known lower bounds for complete bases. The **gate-elimination method** uses induction on the properties of a function f on n variables to show two things: a) a few variables of f can be assigned values so that the resulting function is of the same type as f, and b) a few gates in any circuit for f can be eliminated by this assignment of values. After eliminating all variables by assigning values to them, the function is constant. Since the number of gates in the original circuit cannot be smaller than the number removed during this process, the original circuit has at least as many gates as were removed.

We now apply the gate-elimination method to functions in the class $Q_{2,3}^{(n)}$ defined below. Functions in this class have at least three different subfunctions when any pair of variables ranges through all four possible assignments.

DEFINITION 9.3.1 *A Boolean function $f : \mathcal{B}^n \mapsto \mathcal{B}$ belongs to the class $Q_{2,3}^{(n)}$ if for any two variables x_i and x_j, f has at least three distinct subfunctions as x_i and x_j range over all possible*

values. Furthermore, for each variable x_i there is a value c_i such that the subfunction of f obtained by assigning x_i the value c_i is in $Q_{2,3}^{(n-1)}$.

The class $Q_{2,3}^{(n)}$ contains the function $f_{\text{mod }3,c}^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}$, as we show. Here $z \bmod a$ is the remainder of z after removing all multiples of a.

LEMMA 9.3.1 *For $n \geq 3$ and $c \in \{0, 1, 2\}$, the function $f_{\text{mod }3,c}^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}$ defined below is in $Q_{2,3}^{(n)}$:*

$$f_{\text{mod }3,c}^{(n)}(x_1, x_2, \ldots, x_n) = ((y + c) \bmod 3) \bmod 2$$

where $y = \sum_{i=1}^{n} x_i$ and $\sum$ and $+$ denote integer addition.

Proof We show that the functions $f_{\text{mod }3,c}^{(n)}$, $c \in \{0, 1, 2\}$, are all distinct when $n \geq 1$. When $n = 1$, the functions are different because $f_{\text{mod }3,0}^{(1)}(x_1) = x_1$, $f_{\text{mod }3,1}^{(1)}(x_1) = \overline{x}_1$, and $f_{\text{mod }3,2}^{(1)}(x_1) = 0$. For $n = 2$, y can assume values in $\{0, 1, 2\}$. Because the functions $f_{\text{mod }3,0}^{(2)}(x_1, x_2)$, $f_{\text{mod }3,1}^{(2)}(x_1, x_2)$, and $f_{\text{mod }3,2}^{(2)}(x_1, x_2)$ have value 1 only when $y = x_1 + x_2 = 1, 0, 2$, respectively, the three functions are different.

The proof of membership of $f_{\text{mod }3,c}^{(n)}$ in $Q_{2,3}^{(n)}$ is by induction. The base case is $n = 3$, which holds, as shown in the next paragraph. The inductive hypothesis is that for each $c \in \{0, 1, 2\}$, $f_{\text{mod }3,c}^{(n-1)} \in Q_{2,3}^{(n-1)}$.

To show that for $n \geq 3$, $f_{\text{mod }3,c}^{(n)}$ has at least three distinct subfunctions as any two of its variables range over all values, let y^* be the sum of the $n - 2$ variables that are not fixed and let c^* be the sum of c and the values of the two variables that are fixed. Then the value of the function is $((y^* + c^*) \bmod 3) \bmod 2 = (((y^* \bmod 3) + (c^* \bmod 3)) \bmod 3) \bmod 2$. Since $(y^* \bmod 3)$ and $(c^* \bmod 3)$ range over the values 0, 1, and 2, the three functions are different, as shown in the first paragraph of this proof.

To show that for any variable x_i there is an assignment c_i such that $f_{\text{mod }3,c}^{(n)}$ is in $Q_{2,3}^{(n-1)}$, let $c = 0$. ■

We now derive a lower bound on the circuit size of functions in the class $Q_{2,3}^{(n)}$.

THEOREM 9.3.2 *Over the basis of all Boolean functions on two inputs, Ω, if $f \in Q_{2,3}^{(n)}$ for $n \geq 3$, then*

$$C_\Omega(f) \geq 2n - 3$$

Proof We show that f depends on each of its variables. Suppose it does not depend on x_i. Then, pick x_i and a second variable x_j and let them range over all four possible values. Since the value of x_i has no effect on f, f has at most two subfunctions as x_i and x_j range over all values, contradicting its definition.

We now show that some input vertex x_i of a circuit for f has fan-out of 2 or more. Consider a gate g in a circuit for f whose longest path to the output gate is longest. (See Fig. 9.4.) Since the circuit does not have loops and no other vertex is farther away from the output, both of g's input edges must be attached to input vertices. Let x_i and x_j be the two inputs to this gate. If the fan-out of both of these input vertices is 1, they influence the value of f only through the one gate to which they are connected. Since this gate has at most two

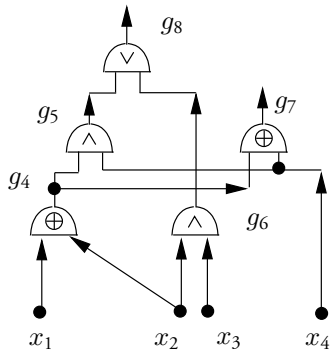


Figure 9.4 A circuit in which gates g_4 has maximal distance from the output gate g_8. The input x_2 has fan-out 2.

values for the four assignments to inputs, f has at most two subfunctions, contradicting the definition of f.

If $n = 3$, this fact demonstrates that the fan-out from the three inputs has to be at least 4, that is, the circuit has at least four inputs. From Theorem 9.3.1 it follows that $C_\Omega(f) \geq 2n - 3$ for $n = 3$. This is the base case for a proof by induction.

The inductive hypothesis is that for any $f^* \in Q_{2,3}^{(n-1)}$, $C_\Omega(f^*) \geq 2(n-1) - 3$. From the earlier argument it follows that there is an input vertex x_i in a circuit for $f \in Q_{2,3}^{(n)}$ that has fan-out 2. Let x_i have that value that causes the subfunction f^* of f to be in $Q_{2,3}^{(n-1)}$. Fixing x_i eliminates at least two gates in the circuit for f because each gate connected to x_i either has a constant output, computes the identity, or computes the NOT of its input. The negation, if any, can be absorbed by the gate that precedes or follows it. Thus,

$$C_\Omega(f) \geq C_\Omega(f^*) + 2 \geq 2(n-1) - 3 + 2 = 2n - 3$$

which establishes the result. ■

As a consequence of this theorem, the function $f_{\text{mod }3,c}^{(n)}$ requires at least $2n - 3$ gates over the basis B_2. It can also be shown to require at most $3n + O(1)$ gates [86].

We now derive a second lower-bound result using the gate-elimination method. In this case we demonstrate that the upper bound on the complexity of the multiplexer function $f_{\text{mux}}^{(n)} : \mathcal{B}^{2^n+n} \mapsto \mathcal{B}$ introduced in Section 2.5.5, which is $2^{n+1} + O(n\sqrt{2^n})$, is optimal to within an additive term of size $O(n\sqrt{2^n})$. (The multiplexer function is also called the **storage access function**.) We generalize the storage access function $f_{\text{SA}}^{(n,k)} : \mathcal{B}^{n+k} \mapsto \mathcal{B}$ slightly and write it in terms of a k-bit address $\boldsymbol{a}$ and an n-tuple $\boldsymbol{x}$, as shown below, where $|\boldsymbol{a}|$ denotes the integer represented by the binary number $\boldsymbol{a}$ and $2^k \geq n$.:

$$f_{\text{SA}}^{(n,k)}(a_{k-1}, \ldots, a_1, a_0, x_{n-1}, \ldots, x_0) = x_{|\boldsymbol{a}|}$$

Thus, $f_{\text{mux}}^{(m)} = f_{\text{SA}}^{(2^m,m)}$.

To derive a lower bound on the circuit size of $f_{\text{SA}}^{(n,k)}$ we introduce the class $F_s^{(n,k)}$ of Boolean functions on $n + k$ variables defined below.

DEFINITION 9.3.2 *A Boolean function $f : \mathcal{B}^{n+k} \mapsto \mathcal{B}$ belongs to the class $F_s^{(n,k)}$, $2^k \geq n$, if for some set $S \subseteq \{0, 1, \ldots, n-1\}$, $|S| = s$,*

$$f(a_{k-1}, \ldots, a_1, a_0, x_{n-1}, \ldots, x_0) = x_{|\boldsymbol{a}|}$$

for $|\boldsymbol{a}| \in S$.

Clearly, $f_{\text{SA}}^{(n,k)}$ is a member of $F_n^{(n,k)}$. We now show that every function in $F_s^{(n,k)}$ has circuit size that is at least $2s - 2$.

In the proof of Theorem 9.3.2 the gate-elimination method replaced variables with constants. In the following proof this idea is extended to replacing variables by functions. Applying this result, we have that $C_\Omega(f_{\text{mux}}^{(n)}) \geq 2^{n+1} - 1$.

THEOREM 9.3.3 *Let $f : \mathcal{B}^{n+k} \mapsto \mathcal{B}$ belong to $F_s^{(n,k)}$, $2^k \geq n$. Then over the basis B_2 the circuit size of f satisfies the following bound:*

$$C_\Omega(f) \geq 2s - 2$$

Proof In the proof of Theorem 9.3.2 we used the fact that some input variable has fan-out 2 or more, as deduced from a property of functions in $Q_{2,3}^{(n)}$. This fact does not hold for the storage access function (multiplexer), as can be seen from the construction in Section 2.5.5. Thus, our lower-bound argument must explicitly take into account the fact that the fan-out from some input can be 1.

The following proof uses the fact that the basis B_2 contains functions of two kinds, AND-type and parity-type functions. The former compute expressions of the form $(x^a \wedge y^b)^c$ for Boolean constants a, b, c, where the notation x^c denotes x when $c = 1$ and $\overline{x}$ when $c = 0$. Parity-type functions compute expressions of the form $x \oplus y \oplus c$ for some Boolean constant c. (See Problem 9.19.)

The proof is by induction on the value of s. In the base case $s = 1$ and the lower bound is trivially 0. The inductive hypothesis assumes that for $s = s' - 1$, $C_\Omega(f) \geq 2(s' - 1) - 2$. We let $s = s'$ and consider the following mutually exclusive cases:

a) For some $i \in S$, x_i has fan-out 2. Replacing x_i by a constant allows elimination of at least two gates, replaces S by $S - \{i\}$, which has size $s' - 1$, and reduces f to $f^* \in F_{s'-1}^{(n,k)}$, from which we conclude that

$$C_\Omega(f) \geq 2 + C_\Omega(f^*) \geq 2s' + 2 = 2s - 2$$

b) For some $i \in S$, x_i has fan-out 1, its unique successor is a gate G of AND-type, and G computes the expression $(x_i^a \wedge g^b)^c$ for some function g of the inputs. Setting $x_i = \overline{a}$ sets $x_i^a = \overline{a}^a = 0$, thereby causing the expression to have value 0^c, which is a constant. Since G cannot be the output gate, this substitution allows the elimination of G and at least one successor gate, reduces f to $f^* \in F_{s'-1}^{(n,k)}$, and replaces S by $S - \{i\}$, from which the lower bound follows.

c) For some $i \in S$, x_i has fan-out 1, its unique successor is a gate G of parity-type, and G computes the expression $x_i \oplus g \oplus c$ for some function g of the inputs. Replace S by $S - \{i\}$. Since we ask that the output of the circuit be $x_{|\boldsymbol{a}|}$ for $\boldsymbol{a} \in S - \{i\}$, this output

cannot depend on the value of G because a change in x_i would cause the value of G to change. Thus, G is not the output gate and when $\boldsymbol{a} \in S - \{i\}$ we can set its value to any function without affecting the value computed by the circuit. In particular, setting $x_i = g$ causes G to have value c, a constant. This substitution allows the elimination of G and at least one successor gate, and reduces f to $f^* \in F_{s'-1}^{(n,k)}$, from which the lower bound follows.

Thus, in all cases, $C_\Omega(f) \geq 2s' - 2$. ■

The lower bounds given above are derived for two functions over the basis B_2. The best circuit-size lower bound that has been derived for this basis is $3(n-1)$. When the basis is restricted, larger lower bounds may result, as mentioned in the notes and illustrated by Problems 9.22 and 9.23.

9.4 Lower-Bound Methods for Formula Size

Since formulas correspond to circuits of fan-out 1, the formula size of a function may be much larger than its circuit size. In this section we introduce two techniques for deriving lower bounds on formula size that illustrate this point. Each leads to bounds that are quadratic or nearly quadratic in the number of inputs. The first, due to Nečiporuk [230], applies to any complete basis. The second, due to Krapchenko [174], applies to the standard basis Ω_0.

To fix ideas about formula size, we construct a circuit of fan-out 1 for the **indirect storage access function** $f_{\text{ISA}}^{(k,l)} : \mathcal{B}^{k+lK+L} \mapsto \mathcal{B}$, where $K = 2^k$ and $L = 2^l$:

$$f_{\text{ISA}}^{(k,l)}(\boldsymbol{a}, \boldsymbol{x}_{K-1}, \ldots, \boldsymbol{x}_0, \boldsymbol{y}) = y_{|\boldsymbol{x}_{|\boldsymbol{a}|}|}$$

Here $\boldsymbol{a}$ is a k-tuple, $\boldsymbol{x}_j = (x_{j,l-1}, \ldots, x_{j,0})$ is an l-tuple for $0 \leq j \leq K-1$, and $\boldsymbol{y} = (y_{L-1}, \ldots, y_0)$ is an L-tuple. The value of $f_{\text{ISA}}^{(k,l)}$ is computed by indirection; that is, the value of $\boldsymbol{a}$ is treated as a binary number with value $|\boldsymbol{a}|$ that is used to select the $|\boldsymbol{a}|$th l-tuple $\boldsymbol{x}_{|\boldsymbol{a}|}$; this, in turn, is treated as a binary number and its value is used to select the $|\boldsymbol{x}_{|\boldsymbol{a}|}|$th variable in $\boldsymbol{y}$.

A circuit realizing $f_{\text{ISA}}^{(k,l)}$ from multiple copies of the multiplexer (direct storage access function) $f_{\text{mux}}^{(n)} : \mathcal{B}^{2^n+n} \mapsto \mathcal{B}$ is shown schematically in Fig. 9.5. This circuit uses l copies of $f_{\text{mux}}^{(k)} : \mathcal{B}^{2^k+k} \mapsto \mathcal{B}$ and one copy of $f_{\text{mux}}^{(l)} : \mathcal{B}^{2^l+l} \mapsto \mathcal{B}$. The copies of $f_{\text{mux}}^{(k)}$ produce the $|\boldsymbol{a}|$th l-tuple, which is supplied to the copy of $f_{\text{mux}}^{(l)}$ to select a variable from $\boldsymbol{y}$. Since, as shown in Lemma 2.5.5, the function $f_{\text{mux}}^{(k)}$ can be realized by a circuit of size linear in 2^k, a circuit for $f_{\text{ISA}}^{(k,l)}$ can be constructed that is also linear in the size of its input.

A formula for $f_{\text{ISA}}^{(k,l)}$ has fan-out of 1 from every gate. The circuit sketched in Fig. 9.5 has fan-out 1 if and only if the fan-out within each multiplexer circuit is also 1. To construct a formula from this circuit, we first construct one for $f_{\text{mux}}^{(l)}$. The total number of times that address bits appear in a formula for $f_{\text{mux}}^{(l)}$ determines the number of copies of the formula for $f_{\text{mux}}^{(k)}$ that are used in the formula for $f_{\text{ISA}}^{(k,l)}$. A proof by induction can be developed to show that a formula for $f_{\text{mux}}^{(p)}$ can be constructed of size $32^p - 2$ in which address bits occur $2(2^p - 1)$ times. (See Problem 9.24.) Since each occurrence of an address bit in $f_{\text{mux}}^{(l)}$ corresponds to a copy of the formula for $f_{\text{mux}}^{(k)}$, by choosing $L = 2^l = n$ and k the smallest integer such that

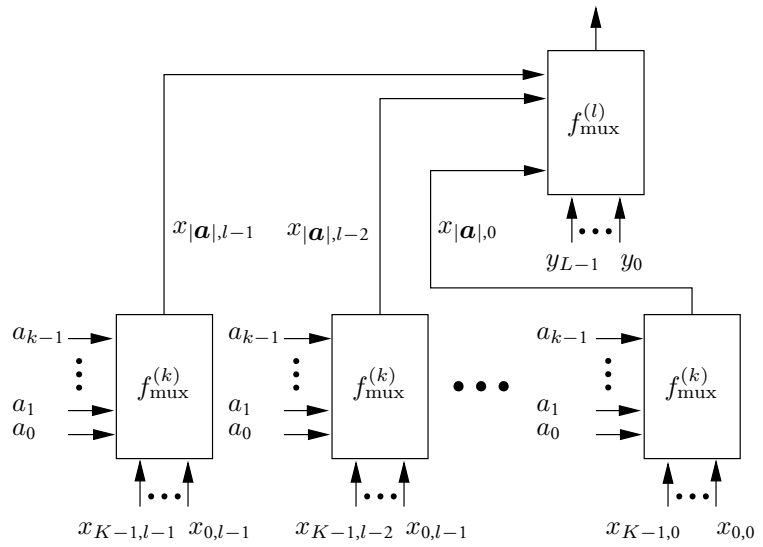


Figure 9.5 The schema used to construct a circuit of fan-out 1 for the indirect storage access function $f^{(k,l)}_{\text{ISA}}$.

$K = 2^k \geq n/l$ we see that $f^{(k,l)}_{\text{ISA}}$ has $2^l + l2^k + k = O(n)$ variables and that its formula size is $2(2^l - 1)L_\Omega\left(f^{(k)}_{\text{mux}}\right) + L_\Omega\left(f^{(l)}_{\text{mux}}\right)$, which is $O(n^2/\log_2 n)$, as summarized in Lemma 9.4.1.

LEMMA 9.4.1 *Let $2^l = n$ and $k = \lceil \log_2 n/l \rceil$. Then the formula size of $f^{(k,l)}_{\text{ISA}} : \mathcal{B}^{k+lK+L} \mapsto \mathcal{B}$ satisfies the following bound:*

$$L_\Omega\left(f^{(k,l)}_{\text{ISA}}\right) = O(n^2/\log_2 n)$$

We now introduce Nečiporuk's method, by which it can be shown that this bound for $f^{(k,l)}_{\text{ISA}}$ is optimal to within a constant multiplicative factor.

9.4.1 The Nečiporuk Lower Bound

The Nečiporuk lower-bound method uses a partition of the variables $X = (x_1, x_2, \ldots, x_n)$ of a Boolean function $f^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}$ into disjoint sets $X_1, X_2, \ldots, X_p$. That is, $X = \bigcup_{i=1}^p X_i$ and $X_i \cap X_j = \emptyset$ for $i \neq j$. The lower bound on the formula size of f is stated in terms of $r_{X_j}(f)$, $0 \leq j \leq p$, the **number of subfunctions of** f when restricted to variables in X_j. That is, $r_{X_j}(f)$ is the number of different subfunctions of f in the variables in X_j obtained by ranging over all values for variables in $X - X_j$.

We now describe Nečiporuk's lower bound on formula size. We emphasize that the strength of the lower bound depends on which partition $X_1, X_2, \ldots, X_p$ of the variables X is chosen. After the proof we apply it to the indirect storage access function. The method cannot provide a lower bound that is larger than $O(n^2/\log n)$ for a function on n variables. (See Problem 9.25.)

THEOREM 9.4.1 *For every complete basis Ω there is a constant c_Ω such that for every function $f^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}$ and every partition of its variables X into disjoint sets $X_1, X_2, \ldots, X_p$, the formula size of f with respect to Ω satisfies the following lower bound:*

$$L_\Omega(f) \geq c_\Omega \sum_{j=1}^{p} \log_2 r_{X_j}(f)$$

Proof Consider T, a minimal circuit of fan-out 1 for f. Let n_j be the number of instances of variables in X_j that are labels for leaves in T. Then by definition $L_\Omega(f) = \sum_{i=1}^{p} n_j$. Let d be the fan-in of the basis Ω.

For each j, $1 \leq j \leq p$, we define the subtree T_j of T consisting of paths from vertices with labels in X_j to the output vertex, as suggested by the heavy lines in Fig. 9.6. We observe that some vertices in such a subtree have one input from a vertex in the subtree T_j (called **controllers** — shaded vertices in Fig. 9.6) whereas others have more than one input from a vertex in T_j (**combiners** — black vertices in Fig. 9.6). Each type of vertex typically has inputs from vertices other than those in T_j, that is, from vertices on paths from input vertices in $X - X_j$.

When the variables $X - X_j$ are assigned values, the output of a controller or combiner vertex depends only on the inputs it receives from other vertices in T_j. The function computed by a controller is a function of its one input y in T_j and can be represented as $(a \wedge y) \oplus b$ for some values of the constants a and b. These constants are determined by the values of inputs in $X - X_j$. We assume without loss of generality that each chain of controllers with no intervening combiners is compressed to one controller. The combiner is also some function of its inputs from other vertices in T_j. Since the number of such inputs is as least 2, a combiner (with fan-in at most d) has at most $d - 2$ inputs determined by variables in $X - X_j$.

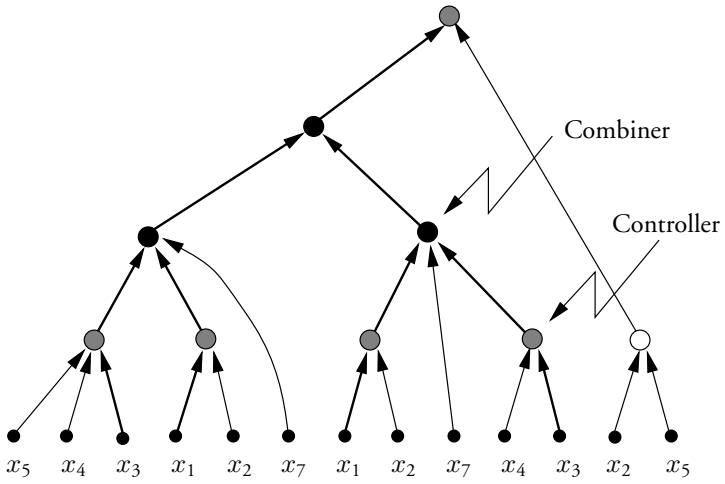


Figure 9.6 The subtree T_j of the tree T is identified by heavy edges on paths from input vertices in the set $X_j = \{x_1, x_3\}$. Vertices in T_j that have one heavy input edge are controller vertices. Other vertices in T_j are combiner vertices.

By Lemma 9.2.1, since T_j has n_j leaves, the number of vertices with fan-in of 2 or more (combiners) is at most $n_j - 1$. Also, by Lemma 9.2.1, T_j has at most $2(n_j - 1)$ edges. Since T_j may have one controller at the output and at most one per edge, T_j has at most $2n_j - 1$ controllers.

The number of functions computed by a combiner is at most one of 2^{d-2} since at most $d - 2$ of its inputs are determined by variables in $X - X_j$. At most four functions are computed by a controller since there are at most four functions on one variable. It follows that the tree T_j associated with the input variables in X_j containing n_j leaves computes r_{X_j} different functions where r_{X_j} satisfies the following upper bound. This bound is the product of the number of ways that each of the controllers and combiners can compute functions.

$$r_{X_j}(f) \leq 2^{(d-2)(n_j-1)} \left(4^{(2n_j-1)}\right) \leq 2^{(d+2)n_j}$$

Thus, $(d+2)n_j \geq \log_2 r_{X_j}(f)$. Since $L_\Omega(f) = \sum_{i=1}^{p} n_j$, the theorem holds for $c_\Omega = 1/(d+2)$. ■

Applying Nečiporuk's lower bound to the indirect storage access function yields the following result, which demonstrates that the upper bound given in Lemma 9.4.1 for the indirect storage access function is tight.

LEMMA 9.4.2 *Let $2^l = n$ and $k = \lceil \log_2(n/l) \rceil$. The formula size of $f_{\text{ISA}}^{(k,l)} : \mathcal{B}^{k+lK+L} \mapsto \mathcal{B}$ satisfies the following bound:*

$$L_\Omega\left(f_{\text{ISA}}^{(k,l)}\right) = \Omega\left(\frac{n^2}{\log_2 n}\right)$$

Proof Let $p = K = 2^k$ and let X_j contain $\boldsymbol{x}_j$. If X_j contains other variables, these are assigned fixed values, which cannot increase $r_{X_j}(f)$. For $0 \leq j \leq K-1$, set $|\boldsymbol{a}| = j$. f has at least 2^L restrictions since for each of the 2^L assignments to $(y_{L-1}, \ldots, y_0)$ the restriction of f is distinct; that is, if two different such L-tuples are supplied as input, they can be distinguished by some assignment to $\boldsymbol{x}_j$. Thus $r_{X_j}(f) \geq 2^L$. Hence, the formula size of $f_{\text{ISA}}^{(k,l)}$, $L_\Omega\left(f_{\text{ISA}}^{(k,l)}\right) \geq c_\Omega KL$, which is proportional to $n^2/\log n$. ■

9.4.2 The Krapchenko Lower Bound

Krapchenko's lower bound applies to the standard basis Ω_0 or any complete subset, namely $\{\wedge, \neg\}$ and $\{\vee, \neg\}$. It provides a lower bound on formula size that can be slightly larger than that given by Nečiporuk's method.

We apply Krapchenko's method to the parity function $f_{\oplus}^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}$, where $f_{\oplus}^{(n)}(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \cdots \oplus x_n$, to show that its formula size is quadratic in n. Since the parity function on two variables can be expressed by the formula

$$f_{\oplus}^{(2)}(x_1, x_2) = (x_1 \wedge \overline{x}_2) \vee (\overline{x}_1 \wedge x_2)$$

it is straightforward to show that the formula size of $f_{\oplus}^{(n)}$ is at most quadratic in n. (See Problem 9.26.)

DEFINITION 9.4.1 *Given two disjoint subsets $A, B \subseteq \{0,1\}^n$ of the set of the Boolean n-tuples, the* **neighborhood** *of A and B, $\mathcal{N}(A,B)$, is the set of pairs of tuples $(\boldsymbol{x},\boldsymbol{y})$, $\boldsymbol{x} \in A$ and $\boldsymbol{y} \in B$, such that $\boldsymbol{x}$ and $\boldsymbol{y}$ agree in all but one position.*

The neighborhood of $A = \{0\}$ and $B = \{1\}$ is the pair $\mathcal{N}(A,B) = \{(0,1)\}$. Also, the neighborhood of $A = \{000, 101\}$ and $B = \{111, 010\}$ is the set of pairs $\mathcal{N}(A,B) = \{(000,010),(101,111)\}$.

Given a function $f : \mathcal{B}^n \mapsto \mathcal{B}$, we use the notation $f^{-1}(0)$ and $f^{-1}(1)$ to denote the sets of n-tuples that cause f to assume the values 0 and 1, respectively.

THEOREM 9.4.2 *For any $f : \mathcal{B}^n \mapsto \mathcal{B}$ and any $A \subseteq f^{-1}(0)$ and $B \subseteq f^{-1}(1)$, the following inequality holds over the standard basis Ω_0:*

$$L_{\Omega_0}(f) \geq \frac{|\mathcal{N}(A,B)|^2}{|A||B|}$$

Proof Consider a circuit for f of fan-out 1 over the standard basis that has the minimal number of leaves, namely $L_{\Omega_0}(f)$. Since the fan-in of each gate is either 1 or 2, by Lemma 9.2.1 the number of leaves is one more than the number of gates of fan-in 2. Each fan-in-2 gate is an AND or OR gate with suitable negation on its inputs and outputs.

Consider a minimal formula for f. Assume without loss of generality that the formula is written over the basis $\{\wedge, \neg\}$. We prove the lower bound by induction, the base case being that of a function on one variable. If the function is constant, $|\mathcal{N}(A,B)| = 0$ and its formula size is also 0. If the function is non-constant, it is either x or $\overline{x}$. (If $f(x) = x$, $f^{-1}(1) = \{1\}$ and $f^{-1}(0) = \{0\}$.) In both cases, $|\mathcal{N}(A,B)| = 1$ since the neighborhood has only one pair. (In the first case $\mathcal{N}(A,B) = \{(0,1)\}$.) Also, $|A| = 1$ and $|B| = 1$, thereby establishing the base case.

The inductive hypothesis is that $L_{\Omega_0}(f^*) \geq |\mathcal{N}(A,B)|/|A||B|$ for any function f^* whose formula size $L_{\Omega_0}(f^*) \leq L_0 - 1$ for some $L_0 \geq 2$. Since the occurrences of NOT do not affect the formula size of a function, apply DeMorgan's theorem as necessary so that the output gate of the optimal (minimal-depth) formula for f is an AND gate. Then we can write $f = g \wedge h$, where g and h are defined on the variables appearing in their formulas. Since the formula for f is optimal, so are the formulas for g and h.

Let $A \subseteq f^{-1}(0)$ and $B \subseteq f^{-1}(1)$. Thus, $f(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \in A$ and $f(\boldsymbol{x}) = 1$ for $\boldsymbol{x} \in B$. Since $f = g \wedge h$, if $f(\boldsymbol{x}) = 1$, then both $g(\boldsymbol{x}) = 1$ and $h(\boldsymbol{x}) = 1$. That is, $f^{-1}(1) \subseteq g^{-1}(1)$ and $f^{-1}(1) \subseteq h^{-1}(1)$. (See Fig. 9.7.) It follows that $B \subseteq g^{-1}(1)$ and $B \subseteq h^{-1}(1)$. Let $B_1 = B_2 = B$. Let $A_1 = A \cap g^{-1}(0)$ (which implies $A_1 \subseteq g^{-1}(0)$) and let $A_2 = A - A_1$. Since $f(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \in A$, but $g(\boldsymbol{x}) = 1$ for $\boldsymbol{x} \in A_2$, as suggested in Fig. 9.7, it follows that $A_2 \subseteq h^{-1}(0)$. (Since $f = g \wedge h$, $f(\boldsymbol{x}) = 0$, and $g(\boldsymbol{x}) = 1$, it follows that $h(\boldsymbol{x}) = 0$.) Finally, observe that $\mathcal{N}(A_1,B_1)$ and $\mathcal{N}(A_2,B_2)$ are disjoint (A_1 and A_2 have no tuples in common) and that $|\mathcal{N}(A,B)| = |\mathcal{N}(A_1,B_1)| + |\mathcal{N}(A_2,B_2)|$.

Given the inductive hypothesis, it follows from the above that

$$\begin{aligned} L_{\Omega_0}(f) = L_{\Omega_0}(g) + L_{\Omega_0}(h) &\geq \frac{|\mathcal{N}(A_1,B_1)|^2}{|A_1||B_1|} + \frac{|\mathcal{N}(A_2,B_2)|^2}{|A_2||B_2|} \\ &= \frac{1}{|B|}\left(\frac{|\mathcal{N}(A_1,B_1)|^2}{|A_1|} + \frac{|\mathcal{N}(A_2,B_2)|^2}{|A_2|}\right) \end{aligned}$$

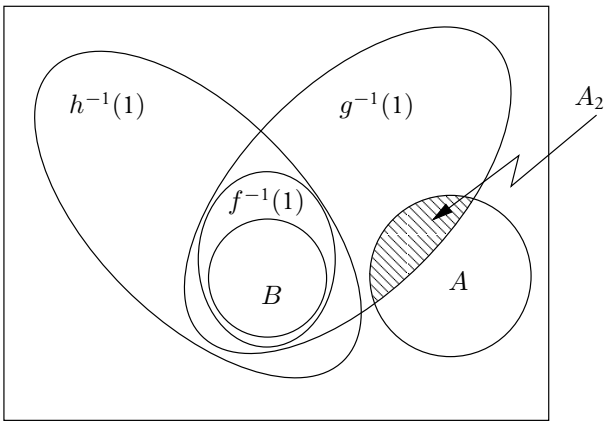


Figure 9.7 The relationships among the sets $f^{-1}(1)$, $g^{-1}(1)$, $h^{-1}(1)$, A_2, and $h^{-1}(0)$.

By the identity $n_1^2/a_1 + n_2^2/a_2 \geq (n_1 + n_2)^2/(a_1 + a_2)$, which holds for positive integers (see Problem 9.3), the desired result follows because $|A| = |A_1| + |A_2|$. ■

Krapchenko's method is easily applied to the parity function $f_{\oplus}^{(n)}$. We need only let A (B) contain n-tuples having an even (odd) number of 1's. ($|A| = |B| = 2^{n-1}$.) Then $|\mathcal{N}(A, B)| = n2^{n-1}$ because for any vector in A there are exactly n vectors in B that are neighbors of it. It follows that $L_{\Omega_0}\left(f_{\oplus}^{(n)}\right) \geq n^2$.

9.5 The Power of Negation

As a prelude to the discussion of monotone circuits for monotone functions in the next section, we consider the minimum number of negations necessary to realize an arbitrary Boolean function $f : \mathcal{B}^n \mapsto \mathcal{B}^m$. From Problem 2.12 on dual-rail logic we know that every such function can be realized by a monotone circuit in which both the variables $x_1, x_2, \ldots, x_n$ and their negations $\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n$ are provided as inputs. Furthermore, every such circuit need have only at most twice as many AND and OR gates as a minimal circuit over Ω_0, the standard basis. Also, the depth of the dual-rail logic circuit of a function is at most one more than the depth of a minimal-depth circuit, the extra depth being that to form $\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n$.

Let $f_{\text{NEG}}^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}^n$ be defined by $f_{\text{NEG}}^{(n)}(x_1, x_2, \ldots, x_n) = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$. As shown in Lemma 9.5.1, this function can be realized by a circuit of size $O(n^2 \log n)$ and depth $O(\log^2 n)$ over Ω_0 using $\lceil \log_2(n+1) \rceil$ negations. This implies that most Boolean functions on n variables can be realized by a circuit whose size and depth are within a factor of about 2 of their minimal values when the number of negations is $\lceil \log_2(n+1) \rceil$.

THEOREM 9.5.1 *Every Boolean function on n variables, $f : \mathcal{B}^n \mapsto \mathcal{B}^m$, can be realized by a circuit containing at most $\lceil \log_2(n+1) \rceil$ negations. Furthermore, the minimal size and depth of such circuits is at most $2C_{\Omega_0}(f) + O(n^2 \log n)$ and $D_{\Omega_0}(f) + O(\log^2 n)$, respectively, where $C_{\Omega_0}(f)$ and $D_{\Omega_0}(f)$ are the circuit size and depth of f over the standard basis Ω_0.*

Proof The proof follows directly from the dual-rail expansion of Problem 2.12 and the following lemma. ■

We now show that the function $f_{\text{NEG}}^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}^n$ defined by $f_{\text{NEG}}^{(n)}(x_1, x_2, \ldots, x_n) = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$ can be realized by circuit size of $O(n^2 \log n)$ over Ω_0 using $\lceil \log_2(n+1) \rceil$ negations.

LEMMA 9.5.1 *$f_{\text{NEG}}^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}^n$ can be realized with $\lceil \log_2(n+1) \rceil$ negations by a circuit over the standard basis that has size $O(n^2 \log n)$ and depth $O(\log n)$.*

Proof The **punctured threshold function** $\tau_{t,\neg i}^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}$, $1 \leq t, i \leq n$, is defined below.

$$\tau_{t,\neg i}^{(n)}(\boldsymbol{x}) = \begin{cases} 1 & \sum_{j=1, j\neq i}^{n} x_j \ \geq \ t \\ 0 & \text{otherwise} \end{cases}$$

This function has value 1 if t or more of the variables other than x_i have value 1. The standard threshold function $\tau_t^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}$ has value 1 when t or more of the variables have value 1. Since the function $(\tau_{0,\neg i}^{(n)}, \tau_{1,\neg i}^{(n)}, \ldots, \tau_{n-1,\neg i}^{(n)})$ is the result of sorting all but the ith input, we know from Theorem 6.8.3 that Batcher's bitonic sorting algorithm will produce this output with a circuit of size $O(n \log^2 n)$ and depth $O(\log^2 n)$ because max and min of a comparator unit compute AND and OR on binary inputs. Ajtai, Komlós, and Szemerédi [14] have improved this bound to $O(n \log n)$ but with a very large coefficient, and simultaneously achieve depth $O(\log n)$. Thus, all the functions $\{\tau_{t,\neg i}^{(n)} \mid 1 \leq t, i \leq n\}$ can be realized with $O(n^2 \log n)$ gates and depth $O(\log n)$ over Ω_0.

Observe that for input $\boldsymbol{x}$ there is some largest t, $t = t_0$, such that $\tau_{t_0}^{(n)}(\boldsymbol{x}) = 1$. If $\tau_{t_0,\neg i}^{(n)}(\boldsymbol{x}) = 1$, then $x_i = 0$; otherwise, $x_i = 1$. Let the **implication function** $a \Rightarrow b$ have value 1 when $a = 0$ or when $a = 1$ and $b = 1$ and value 0 otherwise. Then we can express the implication function by the formula $(a \Rightarrow b) = \overline{a} \vee b$. It follows that $\overline{x}_i = (\tau_{t_0}^{(n)}(\boldsymbol{x}) \Rightarrow \tau_{t_0,\neg i}^{(n)}(\boldsymbol{x}))$ because the implication function has value 1 exactly when $x_i = 0$.

We use an indirect method to compute t_0. Since $\tau_t^{(n)}(\boldsymbol{x}) = 0$ for $t > t_0$, $(\tau_t^{(n)}(\boldsymbol{x}) \Rightarrow \tau_{t,\neg i}^{(n)}(\boldsymbol{x})) = 1$ for $t > t_0$. Also, both $\tau_t^{(n)}(\boldsymbol{x})$ and $\tau_{t,\neg i}^{(n)}(\boldsymbol{x})$ have value 1 for $t < t_0$. Using $(x \Rightarrow y) = \overline{x} \vee y$, we can write $\overline{x}_i$ as follows:

$$\overline{x}_i = \left(\overline{\tau_0^{(n)}(\boldsymbol{x})} \vee \tau_{0,\neg i}^{(n)}(\boldsymbol{x})\right) \wedge \left(\overline{\tau_1^{(n)}(\boldsymbol{x})} \vee \tau_{1,\neg i}^{(n)}(\boldsymbol{x})\right) \wedge \cdots \wedge \left(\overline{\tau_{n-1}^{(n)}(\boldsymbol{x})} \vee \tau_{n-1,\neg i}^{(n)}(\boldsymbol{x})\right)$$

The circuit design is complete once a circuit for $\{\overline{\tau_t^{(n)}(\boldsymbol{x})} \mid 1 \leq t \leq n\}$ has been designed. We begin by using a binary sorting circuit that computes $\{\tau_t^{(n)}(\boldsymbol{x}) \mid 1 \leq t \leq n\}$ from $\boldsymbol{x}$, which, as stated above, can be computed with $O(n \log^2 n)$ gates over the standard basis. Let $s_t = \tau_t^{(n)}(\boldsymbol{x})$ for $1 \leq t \leq n$.

For $n = K - 1$, $K = 2^k$ and k an integer, we complete the design by constructing a circuit for the function $\nu^{(k)} : \mathcal{B}^n \mapsto \mathcal{B}^n$, which, given as input the decreasing sequence $s_1, s_2, \ldots, s_n$ $(s_i \geq s_{i+1})$, computes as its jth output $z_j = \overline{s}_j$, $1 \leq j \leq n$. (The case $n \neq 2^k - 1$ is considered below.) That is, $\nu^{(k)}(\boldsymbol{s}) = \boldsymbol{z}$, where $z_t = \overline{\tau_t^{(n)}(\boldsymbol{x})}$. We give a recursive construction of a circuit for $\nu^{(k)}$ whose correctness is established by induction.

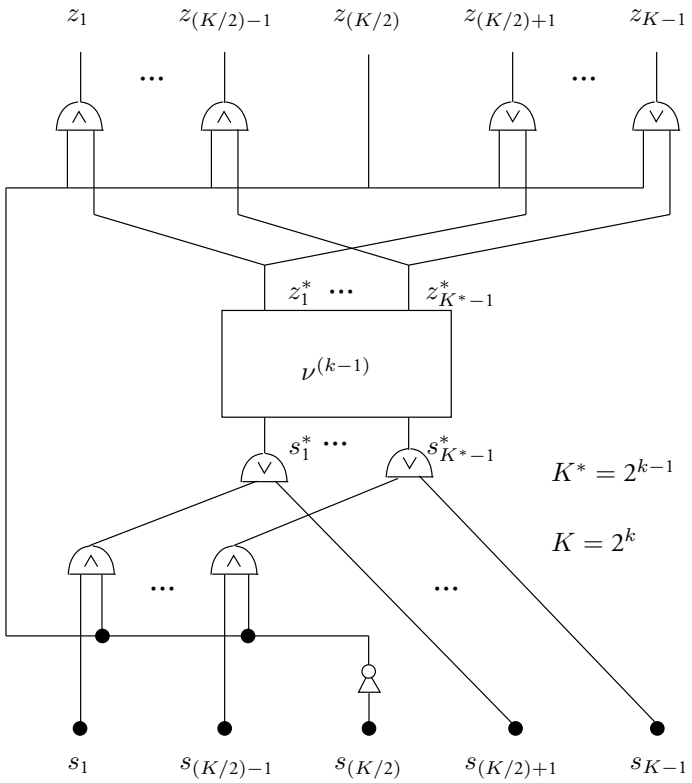


Figure 9.8 A circuit for $\nu^{(k)} : \mathcal{B}^n \mapsto \mathcal{B}^n$, $n = K - 1$, $K = 2^k$. It is given the sorted n-tuple $\boldsymbol{s}$ as input, where $s_j \geq s_{j+1}$ for $1 \leq j \leq n$, and produces as output $\boldsymbol{z}$, where $z_j = \overline{s}_j$.

The base case is a circuit for $\nu^{(1)}$. This circuit has one input, s_1, and one output, $z_1 = \overline{s}_1$, and can be realized by one negation and no other gates.

We construct a circuit for $\nu^{(k)}$ from one for $\nu^{(k-1)}$ using $2n$ additional gates and increasing the depth by three, as shown in Fig. 9.8. Let the inputs and outputs to the circuit for $\nu^{(k-1)}$ be s_i^* and z_i^*, $1 \leq i \leq K^* - 1$, where $K^* = K/2$. It follows that $s_i^* \geq s_{i+1}^*$ for $1 \leq i \leq (K/2) - 1$. By induction $z_i^* = \overline{s^*}_i$ for $1 \leq i \leq n$.

To show that the jth output of the circuit for $\nu^{(k)}$ is $z_j = \overline{s}_j$, we consider cases. If $s_{2^{k-1}} = 0$, then $s_j = 0$ for $j > K/2$. In this case the jth circuit output, $(K/2) < j \leq K - 1$, satisfies $z_j = 1$ (the corresponding output gate is OR), which is the correct value. Also, for $1 \leq j \leq (K/2) - 1$, $z_j = z_j^* = \overline{s}_j$ since the inputs to the circuit for $\nu^{(k-1)}$ are $s_1, s_2, \ldots, s_{(K/2)-1}$ ($s_j = 0$ for $j > K/2$) and its outputs are $\overline{s}_1, \overline{s}_2, \ldots, \overline{s}_{(K/2)-1}$. On the other hand, if $s_{K/2} = 1$, then $s_j = 1$ and $z_j = 0$ for $j \leq (K/2) - 1$ (the corresponding output gate is AND). Also, for $(K/2) + 1 \leq j \leq K - 1$, $z_j = z_j^* = \overline{s}_j$ since the inputs to the circuit for $\nu^{(k-1)}$ are $s_{(K/2)+1}, \ldots, s_{K-1}$ and its outputs are $\overline{s}_{(K/2)+1}, \ldots, \overline{s}_{(K/2)-1}$.

It follows that $k = \log_2(n+1)$ negations are used. The circuit for $\nu^{(k)}$ uses a total of $C(k) = C(k-1) + 2^{k+1} - 3$ gates, where $C(1) = 1$. The solution to this recurrence is $C(k) = 4(2^k) - 3k - 4 = 4n - 3\log_2 n - 4$. Also, the circuit for $\nu^{(k)}$ has depth

$D(k) = D(k-1)+4$, where $D(1) = 0$. The solution to this recurrence is $D(k) = 4(k-1)$. If n is not of the form $2^k - 1$, we increase n to the next largest integer of this form, which implies that $k = \lceil \log_2(n+1) \rceil$. Using the upper bounds on the size of circuits to compute $\tau^{(n)}_{t,\neg i}(\boldsymbol{x})$ for $1 \leq t, i \leq n$, we have the desired conclusion. ■

9.6 Lower-Bound Methods for Monotone Circuits

The best lower bounds that have been derived on the circuit size over complete bases of Boolean functions on n variables are linear in n. Similarly, the best lower bounds on formula size that have been derived over complete bases are at best quadratic in n. As a consequence, the search for better lower bounds has led to the study of monotone circuits (their basis is Ω_{mon}) for monotone functions. In one sense, this effort has been surprisingly successful. Techniques have been developed to show that some monotone functions have exponential circuit size. Since most monotone Boolean functions on n variables have circuit size $\Theta(2^n/n^{3/2})$, this is a strong result. On the other hand, the hope that such techniques would lead to strong lower bounds on circuit size for monotone functions over complete bases has not yet been realized.

Some monotone functions are very important. Among these are the **clique function** $f^{(n)}_{\text{clique},k} : \mathcal{B}^{n(n-1)/2} \mapsto \mathcal{B}$. $f^{(n)}_{\text{clique},k}$ is associated with a family of undirected graphs $G = (V, E)$ on $n = |V|$ vertices and $|E| \leq n(n-1)/2$ edges, where $V = \{1, 2, 3, \ldots, n\}$. The variables of $f^{(n)}_{\text{clique},k}$ are denoted $\{x_{i,j} \mid 1 \leq i < j \leq n\}$, where $x_{i,j} = 1$ if there is an edge between vertices i and j and $x_{i,j} = 0$ otherwise. The value of $f^{(n)}_{\text{clique},k}$ on these variables is 1 if G contains a ***k*-clique**, a set of k vertices such that there is an edge between every pair of vertices in the set. The value of $f^{(n)}_{\text{clique},k}$ is 0 otherwise. Clearly $f^{(n)}_{\text{clique},k}$ is monotone because increasing the value of a variable from 0 to 1 cannot decrease the value of the function.

As stated in Problem 8.24, the CLIQUE problem is **NP**-complete. Since an instance of CLIQUE on a graph with n vertices can be converted to the input format for $f^{(n)}_{\text{clique},k}$ in time polynomial in n, if the circuit size for $f^{(n)}_{\text{clique},k}$ over a complete basis can be shown to be superpolynomial, then from Corollary 3.9.1, **P** $\neq$ **NP**.

There are important similarities and differences between monotone and non-monotone functions. Every non-monotone function can be realized by a circuit over the standard basis Ω_0 in which negations are used only on inputs. (See Problem 9.11.) On the other hand, since circuits without negation compute only monotone functions (Problem 2), negations on inputs are essential.

The first results showing the existence of monotone functions such that their monotone and non-monotone circuit sizes are different were obtained for multiple-output functions. We illustrate this approach below for the n-input binary sorting function, $f^{(n)}_{\text{sort}}$, whose monotone circuit size is shown to be $\Theta(n \log n)$. As stated in Problem 2.17, this function can be realized by a circuit whose size over Ω_0 is linear in n.

We introduce the path method to show that a gap exists between the monotone and non-monotone circuit size of a family of functions. In Section 9.6.3 the approximation method is introduced and used to show that the clique function $f^{(n)}_{\text{clique},k}$ has exponential monotone circuit size.

9.6.1 The Path-Elimination Method

In this section we illustrate the **path-elimination method** for deriving lower bounds on circuit size for monotone functions. This method demonstrates that a path of gates in a monotone circuit can be eliminated by fixing one input variable. Thus, it is the monotone equivalent of the gate-elimination method for general circuits. We apply the method to two problems, binary sorting and binary merging.

Consider computing the binary sorting function $f_{\text{sort}}^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}^n$ introduced in Section 2.11. This function rearranges the bits in a binary n-input string into descending order. Thus, the first sorted output is 1 if one or more of the inputs is 1, the second is 1 if two or more of them are 1, etc. Consequently, we can write $f_{\text{sort}}^{(n)}(x_1, x_2, \ldots, x_n) = (\tau_1^{(n)}, \tau_2^{(n)}, \ldots, \tau_n^{(n)})$, where $\tau_t^{(n)}$ is the threshold function on n inputs with threshold t whose value is 1 if t or more of its inputs are 1 and 0 otherwise. Ajtai, Komlós, and Szemerédi [14] have shown the existence of a comparator-based sorting network on n inputs of size $O(n \log n)$. (The coefficient on this bound is so large that the bound has only asymptotic value.) Such networks can be converted to a monotone network by replacing the max and min operators in comparators with OR and AND, respectively.

THEOREM 9.6.1 *The monotone circuit size for $f_{\text{sort}}^{(n)}$ satisfies the following bounds:*

$$n\lceil \log_2 n \rceil - 2^{\lceil \log_2 n \rceil} \leq C_{\Omega_{\text{mon}}}\left(f_{\text{sort}}^{(n)}\right) = O(n \log n)$$

Proof To derive the lower bound, we show that in any circuit for $f_{\text{sort}}^{(n)}$ there is an input variable that can be set to 1, thereby allowing at least $\lceil \log_2 n \rceil$ gates along a path from it to the output $\tau_1^{(n)}$ to be removed from the circuit and converting the circuit to one for $f_{\text{sort}}^{(n-1)}$. As a result, we show the following relationship:

$$C_{\Omega_{\text{mon}}}\left(f_{\text{sort}}^{(n)}\right) \geq C_{\Omega_{\text{mon}}}\left(f_{sort}^{(n-1)}\right) + \lceil \log_2 n \rceil$$

A simple proof by induction and a little algebra show that the desired result follows from this bound and the fact that $C_\Omega(f_{\text{sort}}^{(2)}) = 2$, which is easy to establish.

Let $x_j = 0$ for $j \neq i$ but let x_i vary. The only functions computed at gates are 0, 1, or x_i. Also, the value of $\tau_1(\boldsymbol{x})$ on such inputs is equal to x_i. Consequently, there must be a path P from the vertex labeled x_i to τ_1 such that at each gate on the path the function x_i is computed. (See Fig. 9.9.) Thus, if we set $x_i = 1$ when $x_j = 0$ for $j \neq i$ the output of each of these gates is 1. Furthermore, since the circuit is monotone, each function computed at a gate is monotone (see Problem 2). Thus, if any other input is subsequently increased from 0 to 1, the value of τ_1 and of all the gates on the path P from x_i remain at 1 and can be removed. This setting of x_i also has the effect of reducing the threshold of all other output functions by 1 and implies that the circuit now computes the binary sorting function on one fewer variable.

Consider a minimal monotone circuit for $f_{\text{sort}}^{(n)}$. The shortest paths from each input to the output $\tau_1^{(n)}$ form a tree of fan-in 2. From Theorem 9.3.1 there is a path in this tree from some input, say x_r, to $\tau_1^{(n)}$ that has length at least $\lceil \log_2 n \rceil$. Consequently the shortest path from x_r to $\tau_1^{(n)}$ has length at least $\lceil \log_2 n \rceil$, implying that at least $\lceil \log_2 n \rceil$ gates can be removed if x_r is set to 1. ■

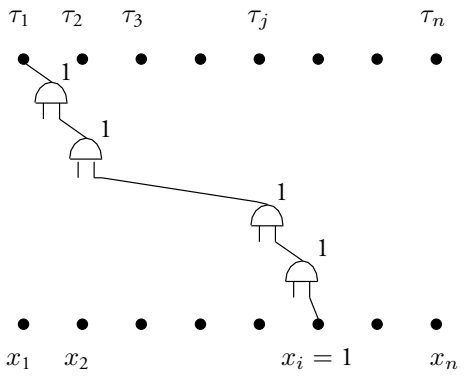


Figure 9.9 When $x_i = 1$ there is a path P to τ_1 such that each gate on P has value 1.

We now derive a stronger result: we show that every monotone circuit for binary merging has a size that is $\Omega(n \log n)$. **Binary merging** is realized by a function $f^{(n)}_{\text{merge}} : \mathcal{B}^n \mapsto \mathcal{B}^n$, $n = 2k$, defined as follows: given two sorted binary k-tuples $\boldsymbol{x}$ and $\boldsymbol{y}$, the value of $f^{(n)}_{\text{merge}}(\boldsymbol{x}, \boldsymbol{y})$ is the n-tuple that results from sorting the n-tuple formed by concatenating $\boldsymbol{x}$ and $\boldsymbol{y}$. Thus, a binary merging circuit can be obtained from one for sorting simply by restricting the values assumed by inputs to the sorting circuit. (Binary merging is a subfunction of binary sorting.) It follows that a lower bound on $C_{\Omega_{\text{mon}}}\left(f^{(n)}_{\text{merge}}\right)$ is a lower bound on $C_{\Omega_{\text{mon}}}\left(f^{(n)}_{\text{sort}}\right)$.

THEOREM 9.6.2 *Let n be even. Then the monotone circuit size for $f^{(n)}_{\text{merge}} : \mathcal{B}^n \mapsto \mathcal{B}^n$ satisfies the following bounds:*

$$(n/2)\log_2 n - O(n) \leq C_{\Omega_{\text{mon}}}\left(f^{(n)}_{\text{merge}}\right) = O(n \log n)$$

Proof The upper bound on $C_{\Omega_{\text{mon}}}\left(f^{(n)}_{\text{merge}}\right)$ follows from the construction given in Theorem 6.8.2 after max and min comparison operators are replaced by ANDs and ORs, respectively.

Let $k = n/2$. The function $f^{(n)}_{\text{merge}}$ operates on two k-tuples $\boldsymbol{x}$ and $\boldsymbol{y}$ to produce the merged result $f^{(n)}_{\text{merge}}(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x}$ and $\boldsymbol{y}$ are in descending order; that is, $x_1 \geq x_2 \geq \cdots \geq x_k$ and $y_1 \geq y_2 \geq \cdots \geq y_k$. As stated above for binary sorting, the output functions are $\tau_1, \tau_2, \ldots, \tau_n$.

Let $x_1 = x_2 = \cdots = x_{r-1} = 1$, $x_{r+1} = \cdots = x_k = 0$, $y_1 = y_2 = \cdots = y_s = 1$, and $y_{s+1} = \cdots = y_k = 0$. Let x_r be unspecified. Since the circuit is monotone, the value computed by each gate circuit is 0, 1, or x_r. Also,

$$\tau_t(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & t < r + s \\ x_r & t = r + s \\ 0 & t > r + s \end{cases}$$

It follows that there must be a path $P_r^{(r+s)}$ of gates from the input labeled x_r to the output labeled τ_{r+s} such that each gate output is x_r. If $x_r = 0$, since the components of $\boldsymbol{x}$

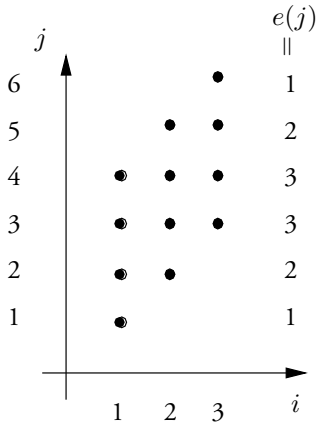


Figure 9.10 Let $f^{(n)}_{\text{merge}}(\boldsymbol{x}, \boldsymbol{y}) = (\tau_1, \ldots, \tau_n)$, where $\boldsymbol{x}$ and $\boldsymbol{y}$ are $(n/2)$-tuples. The dots in the jth row show the inputs on which τ_j depends. $e(j)$ is the number of dots in the jth row.

are sorted, $x_{r+1} = \cdots = x_k = 0$. On the other hand, if $x_r = 1$, by monotonicity the value of τ_{r+s} cannot change under variation of the values $x_{r+1}, \ldots, x_k$. Thus, τ_j is essentially dependent on x_i for i and j satisfying $1 \leq i \leq k$ and $i \leq j \leq i + k$. (See Fig. 9.10.) Let $e(j)$ denote the number of variables in $\boldsymbol{x}$ on which τ_j depends; then $e(j) = j$ for $j \leq k$ and $e(j) = 2k - j + 1$ for $j > k$.

We show by induction that there exist vertex-disjoint paths between x_1 and τ_{s+1}, x_2 and $\tau_{s+2}, \ldots, x_k$ and τ_{s+k} for $0 \leq s \leq k$. (See Fig. 9.11.) Thus, there are $k + 1$ sets of vertex-disjoint paths connecting the $k = n/2$ inputs in $\boldsymbol{x}$ and k consecutive outputs.

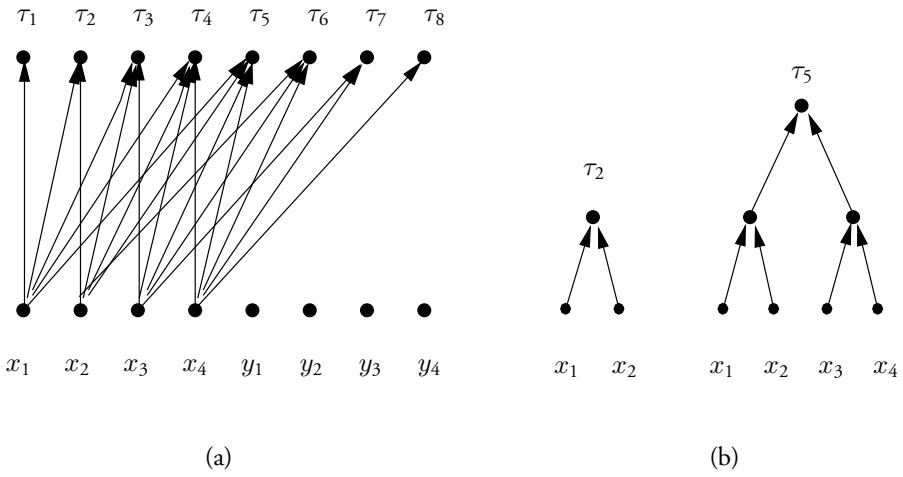


Figure 9.11 (a) In a monotone circuit for $f^{(n)}_{\text{merge}}$, $n = 2k$, $k+1$ sets of k disjoint paths exist between the k inputs $\boldsymbol{x}$ and k consecutive outputs. (b) The paths to an output τ_j form a binary tree.

To show the existence of the vertex-disjoint paths, let $y_1 = y_2 = \cdots = y_s = 1$, $y_{s+1} = \cdots = y_k = 0$ and $x_1 = x_2 = \cdots = x_{r-1} = 1$, but let $x_r, x_{r+1}, \ldots, x_k$ be unspecified. Then $\tau_{r+s} = x_r$ and, as stated above, there is a path $P_r^{(r+s)}$ of gates from an input labeled x_r to the output labeled τ_{r+s} such that each gate has value x_r. Set $x_r = 1$. Reasoning as before, there must be a path $P_{r+1}^{(r+1+s)}$ of gates from an input labeled x_{r+1} to the output labeled τ_{r+1+s} such that each gate has value x_{r+1}. Thus, $P_{r+1}^{(r+1+s)}$ and $P_r^{(r+s)}$ are vertex-disjoint. Extending this idea, we have the desired conclusion about disjoint paths.

We now develop a second fact about these paths that is needed in the lower bound. Let $P_r^{(r+s)}$ be a path from x_r to τ_{r+s}, as suggested in Fig. 9.11(a). Those paths connecting inputs to any one output form a binary tree, as suggested in Fig. 9.11(b). The number of inputs from which there is a path to τ_j is $e(j)$, the number of inputs on which τ_j depends.

To derive the lower bound on $C_{\Omega_{\text{mon}}}(f_{\text{merge}}^{(n)})$, let $d(i, j)$ denote the length (number of edges or non-input vertices (gates)) on the shortest path from an input labeled x_i to the output labeled τ_j. (Clearly, $d(i, j) = 0$ unless $i \leq j \leq i + k$.) Since the path from input x_i to output τ_j described above has a length at least as large as $d(i, j)$, it follows that $C_{\Omega_{\text{mon}}}\left(f_{\text{merge}}^{(n)}\right)$ satisfies the following bound:

$$C_{\Omega_{\text{mon}}}\left(f_{\text{merge}}^{(n)}\right) \geq \max\left\{\sum_{r=1}^{k} d(r, r+s) \mid 0 \leq s \leq k\right\}$$

Since the maximum of a set of integers is at least equal to the average of these integers, we have the following for $k = n/2 \geq 1$:

$$C_{\Omega_{\text{mon}}}\left(f_{\text{merge}}^{(n)}\right) \geq \frac{1}{k+1}\sum_{s=0}^{k}\sum_{r=1}^{k} d(r, r+s) = \frac{1}{k+1}\sum_{j=1}^{2k}\sum_{i=1}^{k} d(i, j)$$

The last identity follows by using the fact that $d(i, j) = 0$ unless $i \leq j \leq i + k$. But $\sum_{i=1}^{k} d(i, j)$ is the sum of the distances of the shortest paths from the relevant inputs of $\boldsymbol{x}$ to output τ_j, $1 \leq j \leq 2k$. Since these paths form a binary tree and τ_j depends on $e(j)$ inputs, this is the external path length of a tree with $e(j)$ leaves. The external path length is at least $e(j)\lceil \log_2 e(j)\rceil - 2^{\lceil \log_2 e(j)\rceil} + e(j)$ (see Problem 9.4). In turn, $x\lceil \log_2 x\rceil - 2^{\lceil \log_2 x\rceil} + x \geq x \log_2 x$, because $\lceil \log_2 x\rceil = (\log_2 x) + \delta$ for $0 \leq \delta < 1$ and $x\lceil \log_2 x\rceil - 2^{\lceil \log_2 x\rceil} + x = x \log_2 x + x(1 - 2^{\delta} + \delta)$, where $1 - 2^{\delta} + \delta$ is easily shown to be a concave function whose minimum value occurs at either $\delta = 0$ or $\delta = 1$, both of which are 0. Thus, $1 - 2^{\delta} + \delta \geq 0$ and the result follows. Thus, the size of smallest monotone circuit satisfies the following lower bound when $n = 2k$:

$$\begin{aligned} C_{\Omega_{\text{mon}}}\left(f_{\text{merge}}^{(n)}\right) &\geq \frac{1}{k+1}\sum_{j=1}^{2k}[e(j) \log_2 e(j)] \\ &= \frac{2}{k+1}\sum_{j=1}^{k}[j \log_2 j] \end{aligned}$$

The last equality uses the definition of $e(j)$ given above. By applying the reasoning in Problem 2.1 and captured in Fig. 2.23, it is easy to show that the above sum is at least as

large as $(2/(k+1))(\log_2 e)\int_{j=1}^{k} y \log_e y \; dy$, whose value is $(2/(k+1))[(k^2/2)\log_2 k - (1/4)k^2(\log_2 e) + 1/4]$. From this the desired conclusion follows, since $k = n/2$. ■

We now present lower bounds on the monotone circuit size of Boolean convolution and Boolean matrix multiplication, problems for which the gap between the monotone and non-monotone circuit size is much larger than for sorting and merging.

9.6.2 The Function Replacement Method

The **function replacement method** simplifies monotone circuits by replacing a function computed at an internal vertex by a new function without changing the function computed by the overall circuit. Since a replacement step eliminates gates and reduces a problem to a subproblem, the method provides a basis for establishing lower bounds on circuit complexity using proof by induction.

We describe two replacement rules and then apply them to Boolean convolution and Boolean matrix multiplication. These two problems are defined in the usual way except that variables assume Boolean values in $\mathcal{B}$ and the multiplication and addition operators are interpreted as AND and OR, respectively.

REPLACEMENT RULES A **replacement rule** is a rule that allows a function computed at a vertex of a circuit to be replaced by another without changing the function computed by the circuit. Before stating such rules for monotone functions, we introduce some terminology.

DEFINITION 9.6.1 *Let $\boldsymbol{x}$ denote the variables of a Boolean function $f : \mathcal{B}^n \mapsto \mathcal{B}$. An* **implicant** *of f is a product (AND), π, of a subset of the literals of f (the variables and their complements) such that if $\pi(\boldsymbol{x}) = 1$ on input n-tuple $\boldsymbol{x}$, then $f(\boldsymbol{x}) = 1$. (This is denoted $\pi \leq f$.) The* **set of implicants** *of a function f is denoted $I(f)$.*

An implicant π of a Boolean function f is a **prime implicant** *if there is no implicant π_1 different from π such that $\pi \leq \pi_1 \leq f$. The* **set of prime implicants** *of a function f is denoted $PI(f)$.*

A **monotone implicant** *(also called a* **monom***) of a monotone Boolean function $f : \mathcal{B}^n \mapsto \mathcal{B}$ is the product (AND) π of uncomplemented variables of f such that if $\pi(\boldsymbol{x}) = 1$ on input n-tuple $\boldsymbol{x}$, then $f(\boldsymbol{x}) = 1$. The* **empty monom** *has value 1. The* **set of monotone implicants** *of a function f is denoted $I_{\text{mon}}(f)$.*

A monotone implicant π of a Boolean function f is a **monotone prime implicant** *if there is no monotone implicant π_1 different from π such that $\pi \leq \pi_1 \leq f$. The* **set of monotone prime implicants** *of a function f is denoted $PI_{\text{mon}}(f)$.*

The products in the sum-of-products expansion (SOPE) are (non-monotone) implicants of a Boolean function. If a function is monotone, it has monotone implicants (monoms). The prime implicants of a Boolean function f define it completely; the OR of its prime implicants is a formula representing it. In the case of a monotone Boolean function, the prime implicants are monotone prime implicants. (See Problem 9.33.)

When it is understood from context that an implicant or prime implicant is monotone, we may omit the word "monotone" and use the subscript "mon." This will be the case in this section.

The function $c_{j+1} = (p_j \wedge c_j) \vee g_j$ used in the design of a full adder (see Section 2.7) is a monotone function of the variables p_j, c_j, and g_j. Its set of implicants is $I(c_{j+1}) = \{p_j \wedge c_j,\ g_j,\ p_j \wedge g_j,\ c_j \wedge g_j,\ p_j \wedge c_j \wedge g_j\}$. If any one of these products has value 1 then so does c_{j+1}. Its set of prime implicants is $PI(c_{j+1}) = \{p_j \wedge c_j, g_j\} \subseteq I(c_{j+1})$ because these are the smallest products for which c_{j+1} has value 1. Thus, c_{j+1} is defined by $PI(c_{j+1})$ and represented as $c_{j+1} = (p_j \wedge c_j) \vee g_j$.

We now present a replacement rule for monotone functions that captures the following idea: if a function g computed by a gate of a monotone circuit has a monom π that is not a monom of the function f computed by the complete circuit, then π can be removed from g without affecting the value of f. This idea is valid in monotone circuits because the absence of negation provides only one way to eliminate extra monoms, namely, by ORing them with products containing a subset of their variables. Taking the AND of a monom with another term creates a longer monom. Thus, since monoms that are not monoms of the function f computed by a circuit must be eliminated, there is no loss of generality in assuming that they are not produced in the first place.

DEFINITION 9.6.2 *Let $f : \mathcal{B}^n \mapsto \mathcal{B}$ and $g : \mathcal{B}^n \mapsto \mathcal{B}$ be two monotone functions. Let g be computed within a monotone circuit for f. The following is a replacement rule for g:*

a) Let $\pi_1 \in PI(g)$ and let h be defined by $PI(h) = PI(g) - \{\pi\}$. Replace the gate computing g by one computing h if for all monoms π' (including the empty monom), $\pi \wedge \pi' \notin PI(f)$.

We now show that any monom π satisfying Rule (a) can be removed from $PI(g)$ because it contributes nothing to the computation of f.

LEMMA 9.6.1 *Let $f : \mathcal{B}^n \mapsto \mathcal{B}$ and $g : \mathcal{B}^n \mapsto \mathcal{B}$ be two monotone functions and let $\pi \in PI(g)$ be such that for all monoms π' (including the empty monom), $\pi \wedge \pi' \notin PI(f)$. Let h be defined by $PI(h) = PI(g) - \{\pi\}$. If g is computed in some monotone circuit for f, the circuit obtained by replacing g by h also computes f.*

Proof Let $\mathcal{C}$ denote a circuit for f within which the function g is computed. Let $\mathcal{C}^*$ be the circuit obtained by replacing g by h under Rule (a). Since $h \leq g$ and the circuit is monotone, the function f^* computed by $\mathcal{C}^*$ satisfies $f^* \leq f$. We suppose that $f^* \neq f$ and show that a contradiction results.

If $f^* \neq f$, there is some input n-tuple $\boldsymbol{a} \in \mathcal{B}^n$ such that $f^*(\boldsymbol{a}) = 0$ but $f(\boldsymbol{a}) = 1$. Since the only change in the circuit occurred at the gate computing g, by monotonicity, on this tuple $g(\boldsymbol{a}) = 1$ but $h(\boldsymbol{a}) = 0$. It follows that $\pi(\boldsymbol{a}) = 1$. Let π' be a prime implicant of f for which $\pi'(\boldsymbol{a}) = 1$. We show that $\pi' = \pi \wedge \pi_1$ for some monom π_1, in contradiction to the condition of the lemma.

Let x_i be any variable of π. Then $a_i = 1$ since $\pi(\boldsymbol{a}) = 1$. Define the n-tuple $\boldsymbol{b}$ by $b_i = 0$ and $b_j = a_j$ for $j \neq i$. Since $\boldsymbol{b} \leq \boldsymbol{a}$ and $\pi(\boldsymbol{b}) = 0$, h and g both have the same value on $\boldsymbol{b}$. Thus, both circuits compute the same value, which must be 0 by monotonicity and the fact that $f^* = 0$ on $\boldsymbol{a}$. Since $\pi'(\boldsymbol{a}) = 1$ and $\pi'(\boldsymbol{b}) = 0$ but only one variable was changed, namely x_i, π' must contain x_i. Since x_i is an arbitrary variable of π, it follows that π' contains π as a sub-monom. ■

This last result implies that if a function f has no prime implicants containing more than l variables, then any monoms containing more than l variables can be removed where they

are first created. This will be useful later when discussing Boolean convolution and Boolean matrix multiplication, since each of their prime implicants depends on two variables.

BOOLEAN CONVOLUTION Convolution over commutative rings is defined in Section 6.7. In this section we introduce the Boolean version, which is defined by a monotone multiple-output function, and derive a lower bound of $n^{3/2}$ on its monotone circuit size. We also show that over a complete basis Boolean convolution can be realized by a circuit of nearly linear size.

DEFINITION 9.6.3 *The* **Boolean convolution function** $f_{\text{conv}}^{(n)} : \mathcal{B}^{2n} \mapsto \mathcal{B}^{2n-1}$ *maps Boolean* n*-tuples* $\boldsymbol{a} = (a_0, a_1, \ldots, a_{n-1})$ *and* $\boldsymbol{b} = (b_0, b_1, \ldots, b_{n-1})$ *onto a* $(2n-1)$*-tuple* $\boldsymbol{c}$*, denoted* $\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$*, where* c_j*,* $0 \leq j \leq 2n-2$*, is defined as*

$$c_j = \sum_{r+s=j} a_r \wedge b_s$$

Boolean convolution can be realized by a circuit over the standard basis Ω_0 for multiplying binary numbers (see Section 2.9) as follows. Represent $\boldsymbol{a}$ and $\boldsymbol{b}$ by the following integers where $q = \lceil \log_2 n \rceil + 1$:

$$a = \sum_{i=0}^{n-1} a_i 2^{qi}, \qquad b = \sum_{j=0}^{n-1} b_j 2^{qj}$$

That is, each bit in a and b is separated by $\lceil \log_2 n \rceil$ zeros. The formal product of a and b is

$$ab = \sum_{k=0}^{2n-2} \left(\sum_{i+j=k} a_i b_j \right) 2^{qk}$$

Because no inner sum in the above expression is more than $2n-1$, at most q bits suffice to represent it in binary notation. Consequently, there is no carry between any two inner sums. It follows that an inner sum is non-zero if and only if $c_k = 1$. Thus, the value of c_k can be obtained by forming the OR of the bits in positions $kq, kq+1, \ldots, kq+q-1$ of the product. Since two binary m tuples can be multiplied in the standard binary notation by a circuit of size $O\left(m(\log m)(\log\log m)\right)$ (see Section 2.9.3), the function $f_{\text{conv}}^{(n)}$ can be computed by a circuit of size $O\left(n(\log^2 n)(\log\log n)\right)$ since $m = nq = O(n \log n)$.

THEOREM 9.6.3 *The circuit size of* $f_{\text{conv}}^{(n)} : \mathcal{B}^{2n} \mapsto \mathcal{B}^{2n-1}$ *over the standard basis satisfies*

$$C_{\Omega_0}\left(f_{\text{conv}}^{(n)}\right) = O\left(n(\log^2 n)(\log\log n)\right)$$

Our goal is to use the function replacement method to show that every monotone circuit for Boolean convolution has size $\Omega(n^{3/2})$. As explained above, the method is designed to use induction to prove lower bounds on monotone circuit size. Each replacement step removes prime implicants from the function g computed at some gate and changes the function f computed by the circuit. If the new function f^* is in the same family as f, the gate-replacement process can continue and induction can be applied. Since the convolution function does not necessarily change to another instance of itself on fewer variables, we place this function in the class of semi-disjoint bilinear forms.

DEFINITION 9.6.4 *Let $f^{(n,m,p)} = (f_1, f_2, \ldots, f_p)$, where each $f_r : \mathcal{B}^{n+m} \mapsto \mathcal{B}$, $1 \leq r \leq p$, is a monotone function on n-tuple $\boldsymbol{x}$ and m-tuple $\boldsymbol{y}$; that is, $f_r(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{B}$. $f^{(n,m,p)}$ is a* **bilinear form** *if each prime implicant of each f_r, $1 \leq r \leq p$, contains one variable of $\boldsymbol{x}$ and one of $\boldsymbol{y}$. A function $f^{(n,m,p)}$ is a* **semi-disjoint bilinear form** *if in addition $PI(f_r) \cap PI(f_s) = \emptyset$ for $r \neq s$ and each variable is contained in at most one prime implicant of any one function.*

Before deriving a lower bound on the number of gates needed for a semi-disjoint bilinear form, we introduce a new replacement rule peculiar to these forms.

LEMMA 9.6.2 *No gate of a monotone circuit of minimal size for a semi-disjoint bilinear form $f^{(n,m,p)}$ computes a function g whose prime implicants include either two variables of $\boldsymbol{x}$ or of $\boldsymbol{y}$.*

Proof We suppose that a minimal monotone circuit does contain a gate g whose prime implicants contain either two variables of $\boldsymbol{x}$ or two of $\boldsymbol{y}$ and show that a contradiction results. Without loss of generality, assume that $PI(g)$ contains x_i and x_j, $i \neq j$. If there is a gate g satisfying this hypothesis, there is one that is closest to an input variable. This must be an OR gate because AND gates increase the length of prime implicants. Because the gate in question is closest to inputs, at least one of x_i and x_j is either an input to this OR gate or is the input to some OR gate that is on a path of OR gates to this gate. (See Fig. 9.12.)

A simple proof by induction on its circuit size demonstrates that if a circuit for $f^{(n,m,p)} = (f_1, \ldots, f_p)$ contains a gate computing g then f_r, $1 \leq r \leq p$, can be written as follows (see Problem 9.36):

$$f_r(\boldsymbol{x}, \boldsymbol{y}) = (p_r(\boldsymbol{x}, \boldsymbol{y}) \wedge g(\boldsymbol{x}, \boldsymbol{y})) \vee q_r(\boldsymbol{x}, \boldsymbol{y}) \tag{9.1}$$

Here $p_r(\boldsymbol{x}, \boldsymbol{y})$ and $q_r(\boldsymbol{x}, \boldsymbol{y})$ are Boolean functions. Of course, if for no r is f_r a function of g, then we can set $p_r(\boldsymbol{x}, \boldsymbol{y}) = 0$ and the circuit is not minimal.

If f_r depends on g, $p_r(\boldsymbol{x}, \boldsymbol{y}) \neq 0$. However, $p_r(\boldsymbol{x}, \boldsymbol{y}) \neq 1$ because otherwise both x_i and x_j are prime implicants of f_r, contradicting its definition. Also, $PI(p_r(\boldsymbol{x}, \boldsymbol{y}))$ cannot have any monoms containing one or more instances of a variable in $\boldsymbol{x}$ or two or more instances of variables in $\boldsymbol{y}$ because when ANDed with g they produce monoms that could be removed by Rule (a) of Definition 9.6.2 and the circuit would not be minimal. It follows that $PI(p_r(\boldsymbol{x}, \boldsymbol{y}))$ can contain only single variables of $\boldsymbol{y}$. But this implies that for some k, $y_k \wedge g \in I(f_r)$, which together with the fact that $x_i, x_j \in PI(g)$ implies that

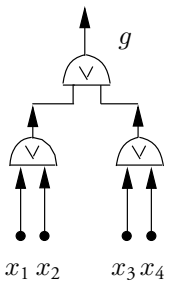


Figure 9.12 If $PI(g)$ for a gate g contains x_i and x_j, then either x_i or x_j is input to an OR gate on a path of OR gates to g.

$y_k \wedge x_i, y_k \wedge x_j \in I(f_r)$. But $y_k \wedge x_i$ and $y_k \wedge x_j$ cannot both be prime implicants of f_r because they violate the requirement that no two prime implicants of f_r contain the same variable. It follows that f_r does not depend on g. ■

The Boolean convolution function is a semi-disjoint bilinear form. Each implicant of each component of $\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$ contains one variable of $\boldsymbol{a}$ and one of $\boldsymbol{b}$. In addition, the prime implicants of c_i and c_j are disjoint if $i \neq j$. Finally, each variable appears in only one implicant of a component function, although it may appear in more than one such function.

THEOREM 9.6.4 *Let $f^{(n,m,p)} : \mathcal{B}^{n+m} \mapsto \mathcal{B}^p$, $f^{(n,m,p)} = (f_1, f_2, \ldots, f_p)$, be a semi-disjoint bilinear form, where $f_r(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{B}$. Let d_i be the number of functions in $\{f_1, f_2, \ldots, f_p\}$ that are essentially dependent on the input variable x_i, $1 \leq i \leq n$. Then the monotone circuit size of $f^{(n,m,p)}$ must satisfy the following lower bound:*

$$C_{\Omega_{\text{mon}}}\left(f^{(n,m,p)}\right) \geq \sum_{i=1}^{n} \sqrt{d_i}$$

Proof The proof is by induction. The basis for induction is the semi-disjoint bilinear form on two variables $f^{(1,1,1)}(x, y) = x \wedge y$. In this case $d_1 = 1$ and $C_{\Omega_{\text{mon}}}\left(f^{(1,1,1)}\right) = 1$. We assume that any semi-disjoint bilinear form in $n + m - 1$ or fewer variables satisfies the lower bound. We show that setting $x_i = 0$ produces another function that is a semi-disjoint bilinear form and allows the removal of at least $\sqrt{d_i}$ gates. The lower bound follows by induction. We consider only minimal circuits.

Let u_i denote the number of functions in $\{f_1, f_2, \ldots, f_p\}$ that are essentially dependent on x_i and have a single prime implicant (such as $c_0 = a_0 \wedge b_0$ and $c_{2n-2} = a_{n-1} \wedge b_{n-1}$ for convolution). Setting $x_i = 0$ eliminates the u_i AND gates at which these outputs are computed. We show that at least $\sqrt{d_i - u_i}$ OR gates can also be eliminated. Since $u_i + \sqrt{d_i - u_i} \geq \sqrt{d_i}$ (see Problem 9.8), we have the desired conclusion.

Let V_i denote those outputs that depend on x_i whose associated function has at least two prime implicants. Then $|V_i| = d_i - u_i$. There must be at least one OR gate on each path P from x_i to $f_r \in V_i$ because, if not, each path contains only ANDs and f_r has only one prime implicant that contains x_i, in contradiction to the definition of V_i.

We claim that on each path P from an input labeled x_i to some $f_r \in V_i$ there is an OR gate computing a function g_t such that $x_{i_t} \wedge y_{j_t} \in PI(g_t)$ for some $x_{i_t} \neq x_i$. Let $E_i = \{g_t\}$ be those OR gates closest to an input vertex x_i. Call E_i the **bottleneck for variable** x_i. We shall show that $|E_i| \geq \sqrt{d_i - u_i}$ and that each of the gates in E_i can be eliminated by setting $x_i = 0$.

If the claim is false, then there is a path P from input x_i to output $f_r \in V_i$ such that for each OR gate (let it compute g_t) on P there is no $x_{i_t} \neq x_i$ such that $x_{i_t} \wedge y_{j_t} \in PI(g_t)$. Therefore, either all monoms of $PI(g_t)$ a) contain x_i or b) are monoms that are not implicants of an output (they are not of the form $x_{i_t} \wedge y_{j_t}$). In case a), setting $x_i = 0$ causes the OR gates on P to have value 0, which forces the AND gates on P and f_r to have value 0, contradicting the definition of f_r (it has at least two prime implicants). In the second case under Rule (a) the monoms not containing x_i can be removed without changing the functions computed. Thus, when $x_i = 0$, the output of each OR gate on P has value 0, which contradicts the definition of f_r since it contains at least two prime implicants.

We now show that $|E_i| \geq \sqrt{d_i - u_i}$. Since each of the OR gates in E_i has a prime implicant $x_{i_t} \wedge y_{j_t}$ not containing x_i, their outputs can be set to 1 by setting $x_{i_t} = y_{j_t} = 1$

for $1 \leq t \leq |E_i|$. This eliminates all dependence of $f_r \in V_i$ on x_i. However, since inputs have only been assigned value 1 (and not 0), this dependence on x_i can be eliminated only if all functions in V_i have value 1; that is, at least one prime implicant of each of them is set to 1 by this assignment. Since each variable appears in at most one prime implicant of a function, the number of different variables x_{i_t} (and y_{i_t}) that are set to 1 is at most $|E_i|$. Thus, at most $|E_i|^2$ prime implicants can be assigned value 1 by this assignment. Thus, if $|E_i|^2 < (d_i - u_i)$, we have a contradiction since $|V_i| = (d_i - u_i)$.

We now show that $|E_i|$ OR gates can be eliminated by setting $x_i = 0$. Since each gate is a closest gate to an input labeled x_i with the stated property, there is an OR gate on the path to it with x_i as an input. Thus, setting $x_i = 0$ eliminates one of the two inputs to the OR gate and the need for the gate itself. ■

Since for each of the n input variables in $\boldsymbol{a}$ there are n output functions in $\boldsymbol{c} = \boldsymbol{a} \otimes \boldsymbol{b}$ that depend on it ($d_i = n$ for $1 \leq i \leq n$), the following corollary is immediate.

COROLLARY 9.6.1 *Let $f_{\text{conv}}^{(n)} : \mathcal{B}^{2n} \mapsto \mathcal{B}^{2n-1}$ be the Boolean convolution function. Then the monotone circuit size of $f_{\text{conv}}^{(n)}$ satisfies the following lower bound:*

$$C_{\Omega_{\text{mon}}}\left(f_{\text{conv}}^{(n)}\right) \geq n^{3/2}$$

Unfortunately, no upper bound on the monotone circuit size of $f_{\text{conv}}^{(n)}$ is known that matches this lower bound. A stronger statement can be made for Boolean matrix multiplication.

BOOLEAN MATRIX MULTIPLICATION Matrix multiplication over rings is discussed at length in Section 6.3. In this section we introduce the Boolean version. An $I \times J$ matrix $A = [a_{i,j}]$, $1 \leq i \leq I$ and $1 \leq j \leq J$, is a two-dimensional array of elements in which $a_{i,j}$ is the element in the ith row and jth column. We take the entries in a matrix to be Boolean variables.

DEFINITION 9.6.5 *Let $A = [a_{i,k}]$, $1 \leq i \leq n$ and $1 \leq k \leq m$, $B = [b_{k,j}]$, $1 \leq k \leq m$ and $1 \leq j \leq p$, and $C = [c_{i,j}]$, $1 \leq i \leq n$ and $1 \leq j \leq p$, be $n \times m$, $m \times p$, and $n \times p$ matrices, respectively. The* **product** *$C = A \times B$ of A and B is the function $f_{\text{MM}}^{(n,m,p)} : \mathcal{B}^{nm+mp} \mapsto \mathcal{B}^{np}$ whose value on the matrices A and B is the matrix C whose entry in row i and column j, $c_{i,j}$, is defined as*

$$c_{i,j} = \bigvee_{k=1}^{m} a_{i,k} \wedge b_{k,j}$$

In a more general context the AND operator $\wedge$ and the OR operator $\vee$ are replaced by the multiplication and addition operators over rings.

The above definition can be used as an algorithm to compute $c_{i,j}$, $1 \leq i \leq n$ and $1 \leq j \leq p$, from the entries in matrices A and B. We call this the **standard matrix-multiplication algorithm**. It uses nmp ANDs and $n(m-1)p$ ORs. We now show that every monotone circuit for matrix multiplication requires at least this many ANDs and ORs.

Clearly the matrix multiplication function is a bilinear form. We associate the entries in A with the tuple $\boldsymbol{x}$ and those in B with $\boldsymbol{y}$. We strengthen Theorem 9.6.4 to obtain a lower bound on the number of ORs needed to realize it in a monotone circuit.

LEMMA 9.6.3 *Every monotone circuit for Boolean matrix multiplication $f_{\text{MM}}^{(n,m,p)}$ requires at least $n(m-1)p$ OR gates.*

Proof In the proof of Theorem 9.6.4 we identified a set E_i of gates called the bottleneck associated with each input variable x_i. We demonstrated that each of these gates can be eliminated by setting $x_i = 0$ and that E_i has at least $\sqrt{d_i - u_i}$ gates, where $d_i - u_i = |V_i|$ is the number of circuit outputs that depend essentially on x_i and have at least two prime implicants. These results were shown by proving that all gates in E_i are OR gates and that the tth of these gates' associated function contains a prime implicant of the form $x_{i_t} \wedge y_{j_t}$ for $x_{i_t} \neq x_i$. We then demonstrated that the dependence of the outputs in V_i on the input x_i can be eliminated by setting $x_{i_t} = y_{j_t} = 1$ for $1 \leq t \leq E_i$ but that this contradicts the definition of a semi-definite bilinear form if $|E_i|^2 < |V_i|$. Finally, we proved that by setting $x_i = 0$ each of the gates in E_i could be eliminated. For this lemma, we need only strengthen the lower bound on E_i for matrix multiplication.

Consider a minimal circuit. The proof is by induction on m, with the base case being $m = 1$. In the base case $c_{i,j} = a_{i,1} \wedge b_{1,j}$ for $1 \leq i \leq n$ and $1 \leq j \leq p$ and no ORs are needed. As inductive hypothesis we assume that $f_{\text{MM}}^{(n,m-1,p)}$ requires at least $n(m-2)p$ OR gates. We show that setting any column of A in $f_{\text{MM}}^{(n,m,p)}$ to 0 eliminates np OR gates and reduces the problem to an instance of $f_{\text{MM}}^{(n,m-1,p)}$. It follows that $f_{\text{MM}}^{(n,m,p)}$ requires $n(m-1)p$ OR gates.

When $m \geq 2$, each output function $c_{i,j}$ has at least two prime implicants. We apply the bottleneck argument to this case. Consider the bottleneck $E_{i,k}$ associated with input variable $a_{i,k}$. We show that $|E_{i,k}| \geq p$, from which it follows that at least p OR gates can be eliminated by setting $x_{i,k} = 0$. This reduces the problem to another set of bilinear forms. Repeating this for $1 \leq i \leq n$, we eliminate np OR gates, one column of A, and one row of B. Let $V_{i,j} = \{c_{i,j} \mid 1 \leq j \leq p\}$ be the outputs that depend on $a_{i,k}$.

To show that $|E_{i,k}| \geq p$, let the tth gate of $E_{i,k}$ compute $x_{i_t} \wedge y_{j_t}$ for $x_{i_t} \neq a_{i,k}$. Here $x_{i_t} = a_{i_t,k_t}$ and $y_{j_t} = b_{l_t,j_t}$ for some i_t, k_t, l_t, and j_t. If we set all entries in $\{a_{i_t,k_t} \mid 1 \leq t \leq |E_{i,k}|\} \cup \{b_{l_t,j_t} \mid 1 \leq t \leq |E_{i,k}|\}$ to 1, we eliminate all dependence of outputs in $V_{i,k}$ on $a_{i,k}$. However, since $|V_{i,j}| = p$, the set $\{b_{l_t,j_t}\}$ must contain at least one variable used in $c_{i,j}$ for each $1 \leq j \leq p$. Thus, $|E_{i,k}| \geq p$. ■

We now derive a lower bound on the number of AND gates needed for Boolean matrix multiplication.

LEMMA 9.6.4 *Every monotone circuit for Boolean matrix multiplication $f_{\text{MM}}^{(n,m,p)}$ requires at least nmp AND gates.*

Proof Consider a minimal circuit. The proof is by induction on m, the base case being $m = 1$. In the base case $c_{i,j} = a_{i,1} \wedge b_{1,j}$ for $1 \leq i \leq n$ and $1 \leq j \leq p$ and np ANDs are needed, since np results must be computed, each requiring one AND, and all functions are different. As inductive hypothesis we assume that $f_{\text{MM}}^{(n,m-1,p)}$ requires at least $n(m-1)p$ AND gates. We show that setting any column of A in $f_{\text{MM}}^{(n,m,p)}$ to 1 and the corresponding row of B to 0 eliminates np AND gates and reduces the problem to an instance of $f_{\text{MM}}^{(n,m-1,p)}$. It follows that $f_{\text{MM}}^{(n,m,p)}$ requires nmp AND gates.

For arbitrary $1 \leq k \leq m$ let $G_{i,j}$ be a gate closest to inputs computing a function g such that $PI(g)$ contains $a_{i,k} \wedge b_{k,j}$. Since the gate associated with $c_{i,j}$ has $a_{i,k} \wedge b_{k,j}$ as a

prime implicant, there is such a gate $G_{i,j}$. Furthermore, $G_{i,j}$ must be an AND gate because OR gates cannot generate new prime implicants. Let G_1 and G_2 be gates generating inputs for $G_{i,j}$. Let them compute functions g_1 and g_2. It follows from the definition of $G_{i,j}$ that $a_{i,k} \in PI(g_1)$ and $b_{k,j} \in PI(g_2)$ or vice versa. Let the former hold. If $a_{i,k} = 1$, $g_1 = 1$ and $G_{i,j}$ can be eliminated. We now show that $G_{i,j} \neq G_{i',j'}$ for $(i,j) \neq (i',j')$. Suppose not. Since $i \neq i'$ or $j \neq j'$, there are at least three distinct variables among $a_{i,k}$, $a_{i',k}$, $b_{k,j}$, and $b_{k,j'}$. Therefore either g_1 or g_2 has at least two of these variables as prime implicants. By Lemma 9.6.2 this circuit is not minimal, a contradiction. ■

We summarize the results of this section below.

THEOREM 9.6.5 *The standard algorithm for $f_{\mathrm{MM}}^{(n,m,p)} : \mathcal{B}^{nm+mp} \mapsto \mathcal{B}^{np}$, the Boolean matrix multiplication function, is optimal. It uses nmp ANDs and $n(m-1)p$ ORs.*

We now show that the monotone circuit size of the clique function is exponential.

9.6.3 The Approximation Method

The approximation method is used to derive large lower bounds on the monotone circuit size for certain monotone Boolean functions. In this section we use it to derive an exponential lower bound on the size of the smallest monotone circuit for the clique function $f_{\mathrm{clique},k}^{(n)} : \mathcal{B}^{n(n-1)/2} \mapsto \mathcal{B}$. This method provides an interesting approach to deriving large lower bounds on circuit size. However, as mentioned in the Chapter Notes, it is doubtful that it can be used to obtain large lower bounds on circuit size over complete bases.

The **approximation method** converts a monotone circuit C computing a function f into an approximation circuit $\widehat{C}$ computing a function $\widehat{f}$. This is done by repeatedly replacing a previously unvisited gate farthest away from the output gate by an approximation gate that computes an approximation to the AND or OR gate it replaces. Each replacement operation changes the circuit and increases by a small amount the number of input tuples on which f and the function computed by the new circuit differ. When the entire replacement process is complete, the resulting circuit approximates f poorly; that is, $\widehat{f}$ and f differ on a large number of inputs. For this to happen, the original monotone circuit must have had many gates, each of which contributes a relatively small number of errors to the complete replacement process. This is the essence of the approximation method.

There are a number of ways to approximate AND and OR gates in a monotone circuit. Razborov [270], who introduced the approximation method, used an approximation for gates based on clique indicators, monotone functions associated with a subset of a set of vertices that has value 1 exactly when there is an edge between every pair of vertices in the subset. In this section gates are approximated in terms of the SOPE and POSE forms, a method used by Amano and Maruoka [20] to approximate the clique function.

It is not hard to show that the monotone circuit size of $f_{\mathrm{clique},k}^{(n)}$ is $O(n^n)$. (See Problem 9.37.) We now show that all monotone circuits for $f_{\mathrm{clique},k}^{(n)}$ have size $C_{\Omega_{\mathrm{mon}}}\left(f_{\mathrm{clique},k}^{(n)}\right) \geq \frac{1}{2}(1.8)^{\min(\sqrt{k-1}/2, n/(2k))}$, which is $2^{\Omega(n^{1/3})}$ for k proportional to $n^{2/3}$.

TEST CASES The quality of an approximation to the clique function $f^{(n)}_{\text{clique},k}$ is determined by providing positive and negative test inputs. A k-**positive test input** is a binary $n(n-1)/2$-tuple that describes a graph containing a single k-clique.

The negative test inputs, defined below, describe graphs that have many edges but not quite enough to contain a k-clique. A special set of negative test inputs is associated with balanced partitions of the vertices of an n-vertex graph $G = (V, E)$. A $(\boldsymbol{k-1})$**-balanced partition** of $V = \{v_1, \ldots, v_n\}$ is a collection of $k-1$ disjoint sets, $V_1, V_2, \ldots, V_{k-1}$, such that each set contains either $\lceil n/(k-10\rceil$ or $\lfloor n/(k-1)\rfloor$ elements. (By Problem 9.5 there are $w = n \bmod (k-1)$ sets of the first kind and $k-1-w$ sets of the second kind.) The graph associated with a particular $(k-1)$-balanced partition has an edge between each pair of vertices in different sets and no other edges. For each $(k-1)$-balanced partition, a k-**negative test input** is a binary $n(n-1)/2$-tuple $\boldsymbol{x}$ describing the graph G associated with that partition.

LEMMA 9.6.5 *There are τ_+ k-positive test inputs, where*

$$\tau_+ = \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

and τ_- k-negative test inputs, where for $w = n \bmod (k-1)$

$$\tau_- = \frac{n!}{(\lceil \frac{n}{k-1} \rceil !)^w (\lfloor \frac{n}{k-1} \rfloor !)^{k-1-w} w!(k-1-w)!}$$

Proof It is well known that $\tau_+ = \binom{n}{k}$. To derive the expression for τ_- we index each element of each set in a $(k-1)$-balanced partition. Such a partition has $w = n \bmod (k-1)$ sets containing $\lceil n/(k-1)\rceil$ elements and $k-1-w$ sets containing $\lfloor n/(k-1)\rfloor$ elements. The elements in the first w sets are indexed by the pairs $\{(i,1),(i,2),\ldots,(i,\lceil n/(k-1)\rceil)\}$ for $1 \le i \le w$. Those in the remaining $k-1-w$ sets are indexed by the pairs $\{(i,1),(i,2),\ldots,(i,\lfloor n/(k-1)\rfloor)\}$ for $w+1 \le i \le k-1$. (See Fig. 9.13.) Let $\mathcal{P}$ be the set of all such pairs. To define a k-negative graph, we assign each vertex in the set $V = \{1,2,\ldots,n\}$ to a unique pair. This partitions the vertices into $k-1$ sets. If vertices v_a and v_b are in the same set, the edge variable $x_{a,b} = 0$; otherwise $x_{a,b} = 1$. These assignments define the edges in a graph $G = (V, E)$. There are $n!$ assignments of vertices to pairs. Of these, there are $(\lceil n/(k-1)\rceil !)^w (\lfloor n/(k-1)\rfloor !)^{k-1-w} w!(k-1-w)!$ that

$(1,1)$	$(1,2)$	$(1,3)$	$(1,4)$	$(2,1)$	$(2,2)$	$(2,3)$	$(3,1)$	$(3,2)$	$(3,3)$
v_3	v_7	v_1	v_2	v_9	v_5	v_{10}	v_6	v_4	v_8
v_2	v_1	v_3	v_7	v_5	v_{10}	v_9	v_4	v_8	v_6

Figure 9.13 A set of pairs $\mathcal{P}$ indexing the elements of sets in a $(k-1)$-balanced partition of a set V of n vertices. In this example $n = 10$ and $k = 4$ and the partition has three sets, V_1, V_2, and V_3 containing four, three, and three elements, respectively. Shown are two assignments of variables to pairs in $\mathcal{P}$ that correspond to the same partition of V.

correspond to each graph. To see this, observe that there are $\lceil n/(k-1) \rceil !$ ways to permute the elements in each of the first w sets and $\lfloor n/(k-1) \rfloor !$ ways to permute the elements in each of the remaining $k-1-w$ sets. Also, each of the first w (the last $k-1-w$) sets have the same size and can be ordered in any of $w!$ $((k-1-w)!)$ ways without changing the graph. ■

APPROXIMATOR CIRCUITS It simplifies the development of lower bounds to assume that each AND gate in a circuit is followed by an OR gate and vice versa and that the output gate is an AND gate. This requirement can be met by interposing between successive AND (OR) gates an OR (AND) gate both of whose inputs are connected together. Since this transformation at most triples the number of gates, an exponential lower bound on the size of the transformed circuit yields an exponential lower bound on the size of the original circuit.

A monotone circuit for $f^{(n)}_{\text{clique},k}$ has (edge) variables drawn from the set $\{x_{i,j} \mid 1 \leq i < j \leq n\}$. The approximation to an input variable $x_{i,j}$ is $x_{i,j}$ itself. Gates in a circuit are successively replaced by approximator circuits starting with a gate that is at greatest distance from the root (output vertex) and continuing with previously unvisited gates at greatest distance from the root. Thus, when an AND or OR gate is replaced, its inputs have previously been replaced by functions f_l and f_r that approximate the functions g_l and g_r computed in the original circuit.

Approximations to AND ($\wedge$) and OR ($\vee$) gates are denoted $\widehat{\wedge}$ and $\widehat{\vee}$, respectively. As seen below, the approximation given to a gate is context dependent. Approximations are defined in terms of endpoint sets. Given a set of edge variables, for example $\{x_{1,2}, x_{1,3}, x_{2,3}, x_{1,4}\}$, its associated **endpoint set** is the set of vertex indices used to define the edge variables, which is $\{1,2,3,4\}$ in this example. Given a term t (a product (AND) or sum (OR) of edge variables), the endpoint set associated with it, $E(t)$, is the endpoint set of the edge variables appearing in the term. For example, if $t = x_{1,2} \wedge x_{1,3} \wedge x_{2,3} \wedge x_{1,4}$ or $t = x_{1,2} \vee x_{1,3} \vee x_{2,3} \vee x_{1,4}$, then $E(t) = \{1,2,3,4\}$. The **endpoint size** of a term t, denoted $|E(t)|$, is the number of indices in $E(t)$.

Consider a gate to be approximated. Let its two inputs be from gates that compute functions f_l and f_r. Like any function, f_r and f_l can be represented in either the monotone SOPE or POSE form. (All SOPEs and POSEs in this section are monotone.) The approximation rules for AND and OR gates are described below and denoted $\widehat{\wedge}$ and $\widehat{\vee}$, respectively. Here we let $p = \lfloor \sqrt{(k-1)}/2 \rfloor$ and $q = \lfloor n/(4k) \rfloor$.

$\widehat{\wedge}$: The approximation $f_l \widehat{\wedge} f_r$ to $f_l \wedge f_r$ is obtained by representing $f_l \wedge f_r$ in the sum-of-products expansion (SOPE) and eliminating all product terms whose endpoint set contains more than p vertices. It follows that $f_l \wedge f_r \geq f_l \widehat{\wedge} f_r$.

$\widehat{\vee}$: The approximation $f_l \widehat{\vee} f_r$ to $f_l \vee f_r$ is obtained by representing $f_l \vee f_r$ in the product-of-sums expansion (POSE) and eliminating all sum terms whose endpoint set contains more than q vertices. It follows that $f_l \vee f_r \leq f_l \widehat{\vee} f_r$.

Since $f_l \wedge f_r \geq f_l \widehat{\wedge} f_r$ and $f_l \vee f_r \leq f_l \widehat{\vee} f_r$, if a positive test input $\boldsymbol{x}$ causes the output of the approximated circuit to have value 0 when it should have value 1, then there is an approximated AND gate (including the output gate) that has value 0 on $\boldsymbol{x}$ when it should have value 1. Similarly, if there is a negative test input $\boldsymbol{x}$ that causes the approximated output to be 1 when it should be 0, there is an approximated OR gate that has value 1 on $\boldsymbol{x}$ when it should have value 0. We now examine the performance of approximator circuits.

PERFORMANCE OF APPROXIMATOR CIRCUITS We now show that when the approximation process is complete, the approximation circuit for $f^{(n)}_{\text{clique},k}$ makes a very large number of errors but that each gate approximation introduces a small number of errors. Thus, many gates must have been approximated to produce the large number of errors made by the fully approximated circuit. In fact, we show that the approximating circuit for $f^{(n)}_{\text{clique},k}$ either has output identically 0, thereby making one error on each of the $\tau_+ = \binom{n}{k}$ positive test inputs (it produces 0 when it should produce 1), or makes $\tau_-/2$ errors on the τ_- negative test inputs (it produces 1 when it should produce 0). On the other hand, we also show that approximating one AND or OR gate causes a small number of errors, at most e_{AND} errors per AND gate on positive test inputs and at most e_{OR} errors per OR gate on negative test inputs, quantities for which upper bounds are derived below. It follows that the original circuit for $f^{(n)}_{\text{clique},k}$ either has at least τ_+/e_{AND} AND gates or at least $\tau_-/(2e_{\text{OR}})$ OR gates. The lower bound on the monotone circuit size of $f^{(n)}_{\text{clique},k}$ is the larger of these two lower bounds.

LEMMA 9.6.6 *Let $k \leq n+1$. Then any approximation circuit for $f^{(n)}_{\text{clique},k}$ either computes a function that is identically zero or makes errors on half of the k-negative test inputs.*

Proof Let the approximation circuit for $f^{(n)}_{\text{clique},k}$ compute the function $\widehat{f^{(n)}_{\text{clique},k}}$. If this function is identically zero, we are done. Suppose not. Since the output gate in the original circuit is an AND gate, the function $\widehat{f^{(n)}_{\text{clique},k}}$ is represented by a SOPE in which each term is the product of variables whose endpoint set (the vertices involved) has size at most p. Because $f^{(n)}_{\text{clique},k}$ is not identically zero, there is a non-zero term t such that $\widehat{f^{(n)}_{\text{clique},k}} \geq t$. An error is made on a negative test input if $t = 1$. But this happens only if each of the endpoints in $E(t)$ is in a different set of the $(k-1)$-balanced partition defining the negative test input.

Let ϕ be the fraction of the negative test inputs on which $t = 1$. We derive a lower bound to ϕ by deriving an upper bound on the fraction χ of the $(k-1)$-balanced partitions with the property that two or more vertices in $E(t)$ fall into the same set. It follows that $\phi \geq 1 - \chi$.

To simplify bounding χ, we use the one-to-one correspondence developed in the proof of Lemma 9.6.5 between the n vertices in $V = \{1, 2, 3, \ldots, n\}$ and the pairs $\mathcal{P}$ associated with a $(k-1)$-balanced partition. Since $E(t)$ has at most p vertices, the number of ways to assign two vertices from $E(t)$ to pairs in $\mathcal{P}$ so that two of them fall into the same set, N_2, is at most the number of ways to choose two vertices from a set of p vertices, $p(p-1)/2$, times the number of ways of assigning these two vertices to pairs in $\mathcal{P}$, m_2, and the number of ways of assigning the remaining $n-2$ vertices, $(n-2)!$. Here m_2 is at most the product of the number of ways of choosing a pair for the first vertex, $(k-1)\lceil n/(k-1)\rceil$, and the number of ways of choosing a pair for the second from the same set, $\lceil n/(k-1)\rceil - 1$. Thus, N_2 is at most $(p(p-1)/2)(k-1)\lceil n/(k-1)\rceil(\lceil n/(k-1)\rceil - 1)(n-2)!$, which is at most $p^2\lceil n/(k-1)\rceil(n-1)!/2$. Since there are $n!$ assignments of vertices in V to pairs in $\mathcal{P}$, $\chi \leq p^2\lceil n/(k-1)\rceil/(2n)$. Because $p = \lfloor\sqrt{(k-1)/2}\rfloor$, χ is at most $1/4$ since $k-1 \leq n$. ■

We now derive upper bounds on the number of errors introduced through the approximation of individual AND and OR gates. Since we have assumed that AND and OR gates alternate on any path between inputs and outputs, it follows that the inputs f_l and f_r to an AND gate

are outputs of OR gates (and vice versa). Furthermore, by the approximation rules, if f_l and f_r are inputs to an AND (OR) gate, every sum (product) in their POSE (SOPE) has an endpoint set size of at most q (p). We now show that each replacement of a gate by its approximator introduces a relatively small number of errors. We begin by establishing this fact for OR gates.

LEMMA 9.6.7 *Let an* OR *gate* $\vee$ *and its approximation* $\widehat{\vee}$ *each be given as inputs the functions* f_l *and* f_r *whose SOPE contains product terms of endpoint size* p *or less. Then the number of* k*-negative test inputs for which* $\vee$ *and* $\widehat{\vee}$ *produce different outputs (*$\vee$ *has value* 0 *but* $\widehat{\vee}$ *has value* 1*) is at most* e_{OR} *where* $w = n \bmod (k-1)$*:*

$$e_{\text{OR}} = \frac{(n/2)^{q+1}(n-q-1)!}{(\lceil n/(k-1)\rceil !)^{w}(\lfloor n/(k-1)\rfloor !)^{k-1-w} w!(k-1-w)!}$$

Proof Let $f_{\text{correct}} = f_l \vee f_r$ and $f_{\text{approx}} = f_l \widehat{\vee} f_r$. Let $t_1, \ldots, t_l$ be the product terms in the SOPE for f_{correct}. Since the endpoint size of all terms in the SOPE of f_{correct} is at most p, each term is the product of at most $p(p-1)/2$ variables.

Using the association between $(k-1)$-balanced partitions and pairs of indices given in the proof of Lemma 9.6.5, we count N, the number of one-to-one mappings from V to $\mathcal{P}$ for which $f_{\text{correct}}(\boldsymbol{x}) = 0$ but $f_{\text{approx}}(\boldsymbol{x}) = 1$, after which we divide by D, the number of mappings corresponding to a single partition of the variables, to compute $e_{\text{OR}} = N/D$. From the proof of Lemma 9.6.5 we have that $D = (\lceil n/(k-1)\rceil !)^{w}(\lfloor n/(k-1)\rfloor !)^{k-1-w} w!(k-1-w)!$.

To derive an upper bound to N, observe that $f_{\text{approx}}(\boldsymbol{x})$ is obtained by converting the SOPE of f_{correct} to a POSE and deleting all sums in this POSE whose endpoint set size exceeds q. Thus, N is at most the number of ways to assign vertices to pairs in $\mathcal{P}$ that causes a deleted sum to be 0 because the new POSE may now become 1. But this can happen only if the endpoint set size of the deleted product is at least $q+1$. Thus, only if at least $q+1$ vertices in a sum are assigned values is it possible to have $f_{\text{correct}}(\boldsymbol{x}) = 0$ and $f_{\text{approx}}(\boldsymbol{x}) = 1$.

Below we show that each vertex can be assigned at most $n/2$ different pairs in $\mathcal{P}$. It follows there are at most $(n/2)^{q+1}(n-q-1)!$ ways to assign pairs to $q+1$ or more vertices because the first $q+1$ can be assigned in at most $(n/2)^{q+1}$ ways and the remaining $(n-q-1)$ vertices can be assigned in at most $(n-q-1)!$ ways. This is the desired upper bound on N.

We now show that every mapping from V to $\mathcal{P}$ that corresponds to a negative test input $\boldsymbol{x}$ assigns each vertex to at most $n/2$ pairs in $\mathcal{P}$.

Let $t_1, \ldots, t_l$ be product terms in the SOPE of f_{correct}. We examine these terms in sequence. Consider a partial mapping from V to $\mathcal{P}$ that assigns values to variables so that at least one variable in each of the products $t_1, \ldots, t_{i-1}$ is 0, thereby insuring that each product is 0. Consider now the ith product, t_i. If the partial mapping assigns value 0 to at least one of its variables, we move on to consider t_{i+1}. (It cannot set all variables in t_i to 1 because we are considering mappings causing all terms to be 0.)

Suppose that the partial mapping has not assigned value 0 to any of the variables of t_i. There are two cases to consider. For some variable $x_{a,b}$ of t_i either a) one or b) both of the vertices $v_a, v_b \in V$ has not been assigned a pair in $\mathcal{P}$. In the first case, assign the second vertex to the set containing the first, thereby setting $x_{a,b} = 0$. This can be done in at most $\lceil n/(k-1)\rceil - 1 \le n/(k-1)$ ways since the set contains at most $\lceil n/(k-1)\rceil$ elements and at least one of them has been chosen previously, namely the first vertex. In the second case the

two vertices can be assigned to at most $(k-1)(\lceil n/(k-1)\rceil)(\lceil n/(k-1)\rceil-1) \leq 2n^2/(k-1)$ pairs because the first can be assigned to $(k-1)$ sets each containing at most $\lceil n/(k-1)\rceil$ elements and the second must be assigned to one of the remaining elements in that set.

The number of ways to choose variables in t_i so that it has value 0 is the number of ways to choose a variable of each kind multiplied by the number of ways to assign values to it. Let α be the number of variables of t_i for which one vertex has previously been assigned a pair and let β be the number of variables for which neither vertex has been assigned a pair. ($\beta \leq p(p-1)/2 - \alpha$ since t_i has at most $p(p-1)/2$ variables.) Thus, a variable of the first kind can be assigned in at most $\alpha n/(k-1)$ ways and the number of ways of assigning the two vertices in variables of the second kind is at most $\beta 2n^2/(k-1)$. Since each vertex associated in such pairs can be assigned in the same number of ways, γ, it follows that $\gamma^2 \leq \beta 2n^2/(k-1)$. Thus, $\gamma \leq \sqrt{\beta 2n^2/(k-1)}$.

Summarizing, the variables in t_i can be assigned in at most the following number of ways so that t_i has value 0:

$$\alpha n/(k-1) + \sqrt{(p(p-1)/2 - \alpha)2n^2/(k-1)}$$

This quantity is largest when $\alpha = 0$ and is at most $n/2$ since $p = \lfloor\sqrt{(k-1)}/2\rfloor$, which is the desired conclusion. ■

We now derive an upper bound on the number of errors that can be made by AND gates on k-positive inputs.

LEMMA 9.6.8 *Let an* AND *gate* $\wedge$ *and its approximation* $\widehat{\wedge}$ *each be given as inputs the functions* f_l *and* f_r *whose POSE contains sum terms of endpoint size* q *or less. Then the number of* k*-positive test inputs for which* $\wedge$ *and* $\widehat{\wedge}$ *produce different outputs (*$\wedge$ *has value* 1 *but* $\widehat{\wedge}$ *has value* 0*) is at most* e_{AND}*:*

$$e_{\text{AND}} = \frac{(n/2)^{p+1}(n-p-1)!}{k!(n-k)!}$$

Proof The proof is similar to that of Lemma 9.6.7. Let $f_{\text{correct}} = f_l \wedge f_r$ and $f_{\text{approx}} = f_l \,\widehat{\wedge}\, f_r$. Let $c_1, \ldots, c_l$ be the sum terms in the POSE for f_{correct}. Since by induction the endpoint size of all terms in the POSE of f_l and f_r is at most q, each term in f_{correct} is the sum of at most $q(q-1)/2$ variables.

In this case we count the number of k-positive test graphs (they contain one k-clique) that cause $f_{\text{correct}}(\boldsymbol{x}) = 1$ but $f_{\text{approx}}(\boldsymbol{x}) = 0$. Since a k-positive test graph contains just those edges between a specified set of k vertices, we define each such graph by a one-to-one mapping from the vertices (endpoints) in V to the integers $\mathbb{N}(n) = \{1, 2, \ldots, n\}$, where we adopt the rule that vertices mapped to the first k integers are those in the clique associated with a particular test graph. It follows that each k-positive test graph corresponds to exactly $k!(n-k)!$ of these 1-1 mappings. Then, e_{AND} is the number of such 1-1 mappings for which $f_{\text{correct}}(\boldsymbol{x}) = 1$ but $f_{\text{approx}}(\boldsymbol{x}) = 0$ divided by $k!(n-k)!$.

We show that any mapping that results in $f_{\text{correct}}(\boldsymbol{x}) = 1$ assigns each endpoint to at most $n/2$ values from $\mathbb{N}(n)$. But $f_{\text{approx}}(\boldsymbol{x}) = 0$ for positive test inputs only if more than p endpoints are assigned values, because f_{approx} is obtained from f_{correct} by discarding product terms in its SOPE that contain more than p endpoints. It follows that at most $(n/2)^{p+1}(n-p-1)!$ of the positive test inputs result in an error by the approximate AND gate. Dividing by $k!(n-k)!$, we have the desired upper bound on e_{AND}.

To complete the proof we must show that each endpoint is assigned at most $n/2$ values from $\mathbb{N}(n)$. Consider the sum terms $c_1, \ldots, c_l$ in the POSE of f_{correct} in sequence and consider a partial mapping from V to $\mathbb{N}(n)$ that causes at least one variable in each of the sums $c_1, \ldots, c_{i-1}$ to be 1, thereby insuring that the value of each sum is 1. Now consider the ith sum, c_i. If the partial mapping assigns value 1 to at least one variable, we move on to c_{i+1}. (It cannot set all variables in c_i to 0 because we are considering mappings causing all terms to be 1.)

We now extend the mapping by considering the set C_i of variables of c_i that have not been assigned a value. A given variable $x_{a,b}$ in C_i has either one or no endpoints (vertices) previously mapped to an integer in $\mathbb{N}(n)$. If one endpoint, say a, has been assigned an integer, the other endpoint, b, can be assigned to at most one of $k-2$ integers that cause $x_{a,b} = 1$ because endpoint a was previously assigned a value in the range $\{1, 2, \ldots, k\}$ together with at least one other vertex and b must be different from them. Because there are most $q = \lfloor n/(4k) \rfloor$ variables of the first type, there are at most $q(k-2)$ ways to assign the one endpoint of a variable $x_{a,b}$ of the first type so that $x_{a,b} = 1$.

Consider now variables of the second type. There are at most $q(q-1)/2$ such variables and at most $(q(q-1)/2)k(k-1)$ ways to make assignments to both endpoints so that a variable has value 1. This follows because each endpoint is assigned to a distinct integer among the first k integers in $\mathbb{N}(n)$. Since each endpoint can be assigned in the same number of ways, this number is at most $\sqrt{(q(q-1)/2)k(k-1)}$.

It follows that the number of ways to assign an endpoint so that the correct and approximate functions differ is at most $q(k-2) + \sqrt{(q(q-1)/2)k(k-1)} \leq 2qk$, which is no more than $n/2$ since $q = \lfloor n/(4k) \rfloor$. This is the desired conclusion. ■

The desired result follows from the above lemmas.

THEOREM 9.6.6 *For $n \geq 13$ and $8 \leq k \leq n/2$, every monotone circuit for the clique function $f^{(n)}_{\text{clique},k} : \mathcal{B}^{n(n-1)/2} \mapsto \mathcal{B}$ has a circuit size satisfying the following lower bound:*

$$C_{\Omega_{\text{mon}}}\left(f^{(n)}_{\text{clique},k}\right) \geq \frac{1}{2}(1.8)^{\min(\sqrt{k-1}/2, n/(2k))}$$

The largest value for this lower bound is $C_{\Omega_{\text{mon}}}(f^{(n)}_{\text{clique},k}) = 2^{\Omega(n^{1/3})}$.

Proof From the discussion at the beginning of this section, we see that the monotone circuit size of $f^{(n)}_{\text{clique},k}$ is at least $\min(\tau_+/e_{\text{AND}}, \tau_-/(2e_{\text{OR}}))$. Thus,

$$\begin{aligned} C_{\Omega_{\text{mon}}}(f^{(n)}_{\text{clique},k}) &\geq \min\left(\frac{n!}{2(n/2)^{p+1}(n-p-1)!}, \frac{n!}{(n/2)^{q+1}(n-q-1)!}\right) \\ &\geq \min\left(\frac{(n-p)^{p+1}}{2(n/2)^{p+1}}, \frac{(n-q)^{q+1}}{(n/2)^{q+1}}\right) \end{aligned}$$

Let $8 \leq k \leq n/2$. It follows that $p = \lfloor \sqrt{(k-1)}/2 \rfloor \leq \sqrt{n}/(2\sqrt{2})$ and $q = \lfloor n/(2k) \rfloor \leq n/16$. Thus, $p, q \leq n/10$ if $n \geq 13$. Hence both $(n-p)$ and $(n-q)$ are at least $9n/10$, and

$$C_{\Omega_{\text{mon}}}\left(f^{(n)}_{\text{clique},k}\right) \geq \min\left(\frac{1}{2}(1.8)^{p+1}, (1.8)^{q+1}\right)$$

The desired conclusion follows from this and the observation that $p+1 \geq \sqrt{k-1}/2$ and $q+1 \geq n/(2k)$. That the maximum value of $\min(\sqrt{k-1}/2, n/(2k))$ is $\Omega(n^{1/3})$ under variation of k is left as a problem. (See Problem 9.38.) ■

9.6.4 Slice Functions

Although, as shown above, some monotone functions have exponential circuit size over the monotone basis, it is doubtful that the methods of analysis used to obtain this result can be extended to derive such bounds over the standard basis. (See the Chapter Notes.)

This section introduces a note of optimism by showing that the monotone circuit size of monotone slice functions can provide a strong lower bound on the circuit size of such functions over the standard basis. In addition, there are **NP**-complete languages whose characteristic functions are slice functions. Thus, if such functions can be shown to have super-polynomial monotone circuit size, **P** $\neq$ **NP**.

Let $|\boldsymbol{x}|$ denote the number of 1's in $\boldsymbol{x}$. We now define the slice functions.

DEFINITION 9.6.6 *A function $s : \mathcal{B}^n \mapsto \mathcal{B}$ is a* **slice function** *if there is an integer $0 \leq k \leq n$ such that $s(\boldsymbol{x}) = 0$ if $|\boldsymbol{x}| < k$ and $s(\boldsymbol{x}) = 1$ if $|\boldsymbol{x}| > k$. The* ***k*th slice of a function** *$\boldsymbol{f : \mathcal{B}^n \mapsto \mathcal{B}}$, $0 \leq k \leq n$, is the function $f^{[k]} : \mathcal{B}^n \mapsto \mathcal{B}$ defined below.*

$$f^{[k]}(\boldsymbol{x}) = \begin{cases} 0 & |\boldsymbol{x}| < k \\ f(\boldsymbol{x}) & |\boldsymbol{x}| = k \\ 1 & |\boldsymbol{x}| > k \end{cases}$$

It should be clear from this definition that slice functions are monotone. Below we show that if a Boolean function f on n variables has a large circuit size, then one of its slices has a circuit size that differs from the size of f by at most a multiplicative factor that is linear in n. Thus, a function f has a large circuit size if and only if one of its slice functions has a large circuit size.

We set the stage with a lemma that shows that the circuit size of a Boolean function is bounded above by the circuit size of its slices plus an additive term linear in its number of variables.

LEMMA 9.6.9 *Let Ω_0 be the standard basis and $f : \mathcal{B}^n \mapsto \mathcal{B}$. Then the following holds, where $C_{\Omega_0}(f^{[0]}, f^{[1]}, \ldots, f^{[n]})$ is the circuit size of all the slices simultaneously:*

$$C_{\Omega_0}(f) = C_{\Omega_0}(f^{[0]}, f^{[1]}, \ldots, f^{[n]}) + O(n)$$

Proof The goal is to construct a circuit for f given the input tuple $\boldsymbol{x}$ and a circuit for all the functions $f^{[0]}, f^{[1]}, \ldots, f^{[n]}$. This is easily done. We construct a circuit to count the number of 1's among the n inputs and represent the result in binary. We then supply this number as an address to a direct storage address function (multiplexer) where the other inputs are the values of the slice functions. If the address is $|\boldsymbol{a}|$, the output of the multiplexer is $f^{[|\boldsymbol{a}|]}$. Since, as shown in Lemma 2.11.1, the counting circuit can be realized with a circuit of size linear in n, and, as shown in Lemma 2.5.5, the multiplexer in question can be realized with a linear-size circuit, the result follows. ■

We now establish the connection between the circuit size of a function and that of one of its slices.

THEOREM 9.6.7 *Let Ω_0 be the standard basis and $f : \mathcal{B}^n \mapsto \mathcal{B}$. Then there exists $0 \leq k \leq n$ such that*

$$\frac{C_{\Omega_0}(f)}{n} - O(1) \leq C_{\Omega_0}\left(f^{[k]}\right) \leq C_{\Omega_0}(f) + O(n)$$

Proof The first inequality follows from Lemma 9.6.9, the following inequality and the observation that at least one term in an average is greater than or equal to the average.

$$C_{\Omega_0}\left(f^{[0]}, f^{[1]}, \ldots, f^{[n]}\right) \leq \sum_i C_{\Omega_0}(f^{[i]})$$

The second inequality uses the fact that the kth slice of a function can be expressed as

$$f^{[k]}(\boldsymbol{x}) = \tau_k^{(n)}(\boldsymbol{x}) f(\boldsymbol{x}) + \tau_{k+1}^{(n)}(\boldsymbol{x})$$

Since $\tau_j^{(n)}(\boldsymbol{x})$ can be realized by a circuit of size linear in n (see Theorem 2.11.1), the second inequality follows. ■

In Theorem 9.6.9 we show that the monotone circuit size of slice functions provides a lower bound on their non-monotone circuit size up to a polynomial additive term. Before establishing this result we introduce the concept of pseudo-negation. A **pseudo-negation** for variable x_i in a monotone Boolean function $f : \mathcal{B}^n \mapsto \mathcal{B}$ is a function h_i such that replacing each instance of $\overline{x}_i$ in a circuit for f by h_i does not change the value computed by the circuit. Thus, the pseudo-negation h_i acts like the real negation $\overline{x}_i$.

In Theorem 9.6.9 we also show that for $1 \leq i \leq n$ the punctured threshold function $\tau_{k,\neg i}^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}$, which depends on all the variables except x_i, is a pseudo-negation for a kth slice of every monotone function. Since for a given k each of these threshold functions can be realized by a monotone circuit of size $O(n \log n)$ (see Theorem 6.8.2), they can all be realized by a monotone circuit of size $O(n^2 \log n)$. Although this result can be used in Theorem 9.6.9, the following stronger result is used instead.

We now describe a circuit that computes all of the above pseudo-negations efficiently. This circuit uses the **complementary number system**, a system that associates with each integer i in the set $\mathbb{N}(n) = \{0, 1, 2, \ldots, n-1\}$ the complementary set $\mathbb{N}(n) - \{i\}$. It makes use of results on sorting networks found in Chapter 6.

THEOREM 9.6.8 *The set $\{\tau_{k,\neg i}^{(n)} \mid 1 \leq i \leq n\}$ of pseudo-negations can be realized by a monotone circuit of size $O(n \log^2 n)$.*

Proof We assume that $n = 2^s$. If not, add variables with value 0 to increase the number to the next power of 2. This does not change the value of the function on the first n variables.

For this proof let the pseudo-negations $\tau_{k,\neg i}^{(n)}$ be defined for $0 \leq i \leq n-1$ and on the variables whose indices are in $\mathbb{N}(n)$. (We subtract 1 from each index.) Let $D_i = \mathbb{N}(n) - \{i\}$ denote the indices of the variables on which $\tau_{k,\neg i}^{(n)}$ depends. An efficient monotone circuit to compute all the pseudo-negations $\{\tau_{k,\neg i}^{(n)} \mid i \in \mathbb{N}(n)\}$ is based on an efficient decomposition of the sets $\{D_i \mid i \in \mathbb{N}(n)\}$.

For $a, b \geq 0$, let $U_{a,b}$ be defined by

$$U_{a,b} = \{a2^b + c \mid 0 \leq c \leq 2^b - 1\}$$

For example, $U_{3,3} = \{24, 25, 26, 27, 29, 30, 31\}$, $U_{1,2} = \{4, 5, 6, 7\}$, and $U_{2,1} = \{4, 5\}$. The set $U_{a,b}$ has size 2^b.

For $n = 2^s$, every set $D_i = \mathbb{N}(n) - \{i\}$ can be represented as the disjoint union of the sets $U_{a,b}$ below, where $0 \leq a_{i,j} \leq 2^{s-j} - 1$. (This is the complementary number system; see Fig. 9.14.)

$$D_i = U_{a_{i,s-1},s-1} \cup U_{a_{i,s-2},s-2} \cup \cdots \cup U_{a_{i,0},0}$$

To see this, note that if i is in the first (second) half of $\mathbb{N}(n)$, $U_{a_{i,s-1},s-1}$ denotes the second (first) half; that is, $a_{i,s-1} = 1$ ($a_{i,s-1} = 0$). The next set, $U_{a_{i,s-2},s-2}$, is the half of the remaining set $D_i - U_{a_{i,s-1},s-1}$ that does not contain i, etc. Thus, D_i is decomposed as the disjoint union of sets of size $2^{s-1}, 2^{s-2}, \ldots, 2^0$ For example, when $n = 16$, $D_3 = U_{1,3} \cup U_{1,2} \cup U_{0,1} \cup U_{2,0}$. Figure 9.14 shows the values of $a_{i,s-1}, a_{i,s-2}, \ldots, a_{i,0}$ for each $i \in \mathbb{N}(n)$ for $n = 8$.

As suggested in Fig. 9.14, the sets $\{D_i \mid i \in \mathbb{N}(n)\}$ have either $U_{0,s-1}$ or $U_{1,s-1}$ in common. Similarly, they also have either $U_{1,s-1} \cup U_{1,s-2}$, $U_{0,s-1} \cup U_{1,s-2}$, $U_{3,s-1} \cup U_{0,s-2}$, or $U_{2,s-1} \cup U_{0,s-2}$ in common. Continuing in this fashion, we construct the sets $\{D_i \mid i \in \mathbb{N}(n)\}$ by successively forming the disjoint union of 2^j sets, $1 \leq j \leq s$. Assembling the sets in this fashion is much more economical than assembling them individually.

The value of $\tau^{(n)}_{k,\neg i}$, $i \in \mathbb{N}(n)$, is the kth largest variable whose index is in D_i. From now on we equate the variables with their indices. Sorting the sets into which D_i is decomposed simplifies the computation. But these sets are exactly the sets that are sorted by Batcher's sorting network based on Batcher's merging algorithm. (See Theorem 6.8.3.) Since on Boolean data a comparator consists of one AND for the max operation and one OR for the min operation, a monotone circuit of size $O(n \log^2 n)$ exists to sort the sets $\{U_{i,j} \mid 0 \leq i \leq 2^{s-j} - 1, 0 \leq j \leq s - 1\}$.

The functions $\tau^{(n)}_{k,\neg i}$, $0 \leq i \leq n-1$, can be obtained by sorting the sets $\{U_{i,j} \mid 0 \leq i \leq 2^{s-j} - 1, 0 \leq j \leq s - 1\}$, merging them in groups to form D_i for $i \in \mathbb{N}(n)$, as suggested above, and then taking the kth largest element. A faster way merges the sorted versions of the sets $U_{a_{i,s-1},s-1}$, $U_{a_{i,s-2},s-2}, \ldots, U_{a_{i,0},0}$ in the order in which D_i is assembled above. For each of these sets the sorting network presents its elements in sorted order.

i	$a_{i,0}$	$a_{i,1}$	$a_{i,2}$
0	1	1	1
1	0	1	1
2	3	0	1
3	2	0	1
4	5	3	0
5	4	3	0
6	7	2	0
7	6	2	0

Figure 9.14 The coefficients $a_{i,j}$ of $D_i = \mathbb{N}(n) - \{i\}$ in the expansion $U_{a_{i,s-1},s-1} \cup U_{a_{i,s-2},s-2} \cup \cdots \cup U_{a_{i,0},0}$ for $n = 2^s = 8$ and $s = 3$.

Since only the kth element of D_i is needed, it is not necessary to merge all the elements in each set when two sets are merged. To see which elements need to be merged, let $\Delta_i(j) = U_{a_{i,s-1},s-1} \cup U_{a_{i,s-2},s-2} \cup \cdots \cup U_{a_{i,j},j}$. Then $D_i - \Delta_i(j)$ is a set of size $2^j - 1$. Observe that the kth element of D_i can be obtained by merging elements of rank k and $k-1$ of $\Delta_i(1)$ with the element of $U_{a(i,0),0}$. (They all have value 0 or 1.) The middle element is the kth element in D_i. To obtain elements of rank k and $k-1$ of $\Delta_i(1)$, the elements of rank k, $k-1$, $k-2$ and $k-3$ of $\Delta_i(2)$ are merged with the two elements of $U_{a_{i,1},1}$ and the middle two taken. In general, to obtain the elements of rank $k, \ldots, k-2^j+1$ of $\Delta_i(j)$, the elements of rank $k, \ldots, k-2^{j+1}+1$ of $\Delta_i(j+1)$ are merged with the 2^j elements of $U_{a_{i,j},j}$ and the middle 2^j taken.

We now count the number of extra AND and OR gates needed to perform the merges. There are 2^{s-j} sets $\Delta_i(j)$. The 2^j elements needed from these sets are obtained by merging 2^{j+1} elements of $\Delta_i(j+1)$ with the 2^j elements of $U_{a_{i,j},j}$. Since these sets can be merged in a comparator network with $O(j2^j)$ comparators (see Theorem 6.8.2), it follows that all the sets $\Delta_i(j)$, $0 \le i \le n-1$, can be formed with $O(jn)$ gates for $0 \le j \le s-1$. Summing over j, $0 \le j \le (\log_2 n) - 1$ shows that a total of $O(n \log^2 n)$ extra gates suffice. Since $O(n \log^2 n)$ gates are used to sort the sets $\{U_{i,j} \mid 0 \le i \le 2^{s-j}-1, 0 \le j \le s-1\}$, the desired conclusion follows. ■

We can now show that a large lower bound on the monotone circuit size of a slice function implies a large lower bound on its non-monotone circuit size. The importance of this statement is emphasized by the existence of **NP**-complete slice functions. If such a problem can be shown to have a super-polynomial slice function, then $\mathbf{P} \neq \mathbf{NP}$.

THEOREM 9.6.9 *Let $f : \mathcal{B}^n \mapsto \mathcal{B}$ be a slice function. Then*

$$C_{\Omega_0}(f) \le C_{\Omega_{\text{mon}}}(f) \le 2 \cdot C_{\Omega_0}(f) + O(n \log^2 n)$$

Proof The first inequality holds because the standard basis Ω_0 contains the monotone basis. To establish the second inequality, we convert a circuit over Ω_0 by moving all negations to the input variables. This can be done by at most doubling the number of gates. (See Problems 9.11 and 2.12.)

We now show that for slice functions the negation of an input variable x_i can be replaced by the pseudo-negation function $\tau^{(n)}_{k,\neg i}$. To see this, observe that when $|\boldsymbol{x}| > k$, at least $|\boldsymbol{x}| - 1 = k$ of the variables of $\tau^{(n)}_{k,\neg i}$ are 1 and $\tau^{(n)}_{k,\neg i}$ has value 1. On the other hand, when $|\boldsymbol{x}| < k$, then not enough variables can be 1 for $\tau^{(n)}_{k,\neg i}$ to have value 1. Finally, when $|\boldsymbol{x}| = k$, $\tau^{(n)}_{k,\neg i} = 0$ if $x_i = 1$ because not enough of the remaining variables are 1, and $\tau^{(n)}_{k,\neg i} = 1$ when $x_i = 0$ by a similar reasoning. Now replace $\overline{x}_i$ with $\tau^{(n)}_{k,\neg i}$. Since f is a k-slice, $f = 0$ when $|\boldsymbol{x}| < k$, as is $\tau^{(n)}_{k,\neg i}$. If $\overline{x}_i = 1$ when $|\boldsymbol{x}| < k$, replacing $\overline{x}_i$ by its pseudo-negation means replacing $\overline{x}_i$ by 0, which can only decrease the circuit output since it is monotone. Thus, f is computed correctly in this case. The same is true if $|\boldsymbol{x}| > k$, again by monotonicity. Since $\tau^{(n)}_{k,\neg i} = \overline{x}_i$ when $|\boldsymbol{x}| = k$, the circuit correctly computes f for all inputs when $\overline{x}_i$ is replaced by the ith pseudo-negation. ■

AN NP-COMPLETE SLICE FUNCTION We now exhibit the language HALF-CLIQUE CENTRAL SLICE and show it is **NP**-complete. The characteristic functions of this language are slice functions. It follows from Theorem 9.6.9 that if these slice functions have exponential circuit size,

then **P** $\neq$ **NP**. We show that HALF-CLIQUE CENTRAL SLICE is **NP**-complete by reducing HALF-CLIQUE (see Problem 8.25) to it.

DEFINITION 9.6.7 *A* **central slice** *of a function* $f : \mathcal{B}^n \mapsto \mathcal{B}$ *on* n *variables,* $f^{[\lceil n/2 \rceil]}$*, is the* $\lceil n/2 \rceil$*th slice.*

A central slice of a function f on n variables is the function that has value 0 if the weight of the input tuple is less than $\lceil n/2 \rceil$, value 1 if the weight exceeds this value, and is equal to the value of f otherwise.

Given the function $f : \mathcal{B}^* \mapsto \mathcal{B}$, $f^{(n)}$ denotes the function restricted to strings of length n. The family of central slice functions $\{(f^{(n)})^{[\lceil n/2 \rceil]} \mid n \geq 2\}$ identifies the language $L_{\text{central}}(f) = \{\boldsymbol{x} \in \mathcal{B}^n \mid (f^{(n)})^{[\lceil n/2 \rceil]}(\boldsymbol{x}) = 1, n \geq 2\}$.

The central clique function $f^{(n)}_{\text{clique},\lceil n/2 \rceil}$ has value 1 if the input graph contains a clique on $\lceil n/2 \rceil$ vertices. The central slice of the central clique function $f^{(n)}_{\text{clique},\lceil n/2 \rceil}$ is called the **half-clique central slice function** and denoted $f^{(n)}_{\text{clique slice}}$. It has value 1 if the input graph either contains a clique on $\lceil n/2 \rceil$ vertices or contains more edges than are in a clique of this size.

The language HALF-CLIQUE is defined in Problem 8.25 as strings describing a graph and an integer k such that a graph on n vertices contains an $n/2$-clique or has more than k edges. The language HALF-CLIQUE CENTRAL SLICE associated with the central slice of a central clique function is defined below. It simplifies the following discussion to define $e(k)$ as the number of edges between a set of k vertices. Clearly, $e(k) = \binom{k}{2}$.

HALF-CLIQUE CENTRAL SLICE
Instance: The description of an undirected graph $G = (V, E)$ in which $|V|$ is even.
Answer: "Yes" if G contains a clique on $|V|/2$ vertices or at least $e(|V|/2)/2$ edges.

THEOREM 9.6.10 *The language* HALF-CLIQUE CENTRAL SLICE *is* **NP***-complete. Furthermore, for all* $2 \leq k \leq n$

$$C_{\Omega_{\text{mon}}}\left(\left(f^{(n)}_{\text{clique},\lceil n/2 \rceil}\right)^{[k]}\right) \leq C_{\Omega_{\text{mon}}}\left(f^{(n)}_{\text{clique slice}}\right)$$

For $k < e(n/2)$, $\left(f^{(n)}_{\text{clique},\lceil n/2 \rceil}\right)^{[k]} = \tau^{e(n)}_{k+1}$.

Proof We show that HALF-CLIQUE CENTRAL SLICE is **NP**-complete by reducing HALF-CLIQUE to it. Given a graph $G = (V, E)$ in HALF-CLIQUE that has n vertices, n even, we construct a graph $G' = (V', E')$ on $5n$ vertices such that G either contains an $n/2$-clique or has more than k edges if and only if G' contains a (central) clique on $5n/2$ vertices or has at least $\lceil e(5n/2)/2 \rceil$ edges. The construction, which can be done in polynomial time, transforms a graph on n vertices to one on $5n$ vertices such that the former is an instance of HALF-CLIQUE if and only if the latter is an instance of HALF-CLIQUE CENTRAL SLICE.

Let $V = \{v_1, v_2, \ldots, v_n\}$. Construct G' from G by adding the $4n$ vertices $R = \{r_1, r_2, \ldots, r_{2n}\}$ and $S = \{s_1, s_2, \ldots, s_{2n}\}$. Represent edges in E' of G' with the edge variables $\{y_{i,j} \mid 1 \leq i < j \leq 5n\}$. Each edge between vertices of G is an edge between vertices V of G'. Let every edge between vertices in R be in G' as well as all edges between vertices in V and R. Set the edge variables so that the edges between r_i and s_i, $1 \leq i \leq 2n$,

are absent. The unassigned variables are between vertices in S, between vertices in R and S, and between vertices in V and S, of which there are $8n^2 - 3n$. Fix these unassigned edges so that the number of edges between vertices in $V \cup R \cup S$ is $\lceil e(5n/2)/2 \rceil - k$, $1 \leq k \leq n$. There are sufficiently many unassigned edges to do this.

We now show that G contains an $n/2$-clique or has more than k edges if and only if G' contains an $5n/2$-clique or has more than $\lceil e(5n/2)/2 \rceil$ edges. If G has a $n/2$-clique, the edges between V and R combined with the edges between vertices in R and those in G constitute a $5n/2$ clique since $5n/2$ vertices in $V \cup R$ are completely connected. If V has more than k edges, since there are exactly $\lceil e(5n/2)/2 \rceil - k$ edges between vertices in $V \cup R \cup S$, G' has at least $\lceil e(5n/2)/2 \rceil$ edges. On the other hand, if G' has a $(5n/2)$-clique, because there is at least one absent edge between each pair of vertices (r_i, s_i), $1 \leq i \leq 2n$, the largest clique on vertices in $R \cup S$ has size $2n$. Thus, there must be a $(n/2)$-clique on vertices in V; that is, G contains a $(n/2)$-clique. Similarly, since the number of edges between vertices in V and those in $R \cup S$ is exactly $\lceil e(5n/2)/2 \rceil - k$, if G' contains at least $\lceil e(5n/2)/2 \rceil$ edges, G must contain at least k edges.

The membership of graph G in HALF-CLIQUE is determined by specializing the graph G' by mapping its edge variables to the constants 0 and 1 or to variables of G. Thus, the function testing G's membership is obtained through a subfunction reduction of the function testing G''s membership. (See Definition 2.4.2.) Thus, at no increase in circuit size, for any k a circuit for $\left(f^{(n)}_{\text{clique},\lceil n/2 \rceil}\right)^{[k]}$ can be obtained from a circuit for $f^{(n)}_{\text{clique slice}}$. Thus, the circuit size for the latter is at least as large for the former, which gives the second result of the theorem.

The statement that for $k < e(n/2)$, $\left(f^{(n)}_{\text{clique},\lceil n/2 \rceil}\right)^{[k]} = \tau^{e(n)}_{k+1}$ follows from the observation that for these values of k the value of the clique function on inputs of weight $e(n/2) - 1$ or less is 0. ■

As this theorem indicates, the search for a proof that **P** $\neq$ **NP** can be limited to the study of the monotone circuit size of the central slice of certain monotone functions. Other central slices of **NP**-complete problems have been shown to be **NP**-complete also. (See the Chapter Notes.)

9.7 Circuit Depth

Circuit depth and formula size are exponentially related, as shown in Section 9.2.3. In this section we examine the depth of circuits whose operations have either bounded or unbounded fan-in. As seen in Chapter 3, circuits of bounded fan-in are useful in classifying problems by their complexity and in developing relationships between time and space and circuit size and depth.

Circuits of unbounded fan-in are constructed of AND and OR gates with potentially unbounded fan-in whose inputs are the outputs of other such gates or literals, namely, variables and their negations. Every Boolean function can be realized by a circuit of unbounded fan-in and bounded depth, as is seen by considering the DNF of a Boolean function: it corresponds to a depth-2, unbounded fan-in circuit. Knowledge of the complexity of bounded-depth circuits may shed light on the complexity of bounded-fan-in circuits.

In this section we first show that the depth of a function f is equal to the *communication complexity* of a related problem in a two-player game. Communication complexity is a measure of the amount of information that must be exchanged between two players to perform a computation. We establish such a connection for all Boolean functions over the standard basis Ω_0 and monotone functions over the monotone basis Ω_{mon}. These connections are used to derive lower bounds on circuit depth for monotone and non-monotone functions. After establishing these results we examine bounded-depth circuits and demonstrate that some problems require exponential size when realized by such circuits.

9.7.1 Communication Complexity

We define a *communication game* between two players who have unlimited computing power and communicate via an error-free channel. This game has sufficient generality to derive interesting lower bounds on circuit depth.

DEFINITION 9.7.1 *A* **communication game** (U, V) *is defined by sets* $U, V \subseteq B^n$*, where* $U \cap V = \emptyset$*. An instance of the game is defined by* $\boldsymbol{u} \in U$ *and* $\boldsymbol{v} \in V$*.* $\boldsymbol{u}$ *is assigned to Player I and* $\boldsymbol{v}$ *is assigned to Player II. Players alternate sending binary messages to each other. We assume that the binary messages form a prefix code (no message is a prefix for another) so that one player can determine when the other has finished transmitting a message.*

Although each player has unlimited computing power, each message it sends is a function of just its own n*-tuple and the messages it has received previously from the other player. The two functions used by the players to determine the contents of their messages constitute the* **protocol** Π *under which the communication game is played. The protocol also determines the first player to send a message and termination of the game. The* **goal** *of the game is to find an index* i*,* $1 \leq i \leq n$*, such that* $u_i \neq v_i$*.*

Let $\Pi(\boldsymbol{u}, \boldsymbol{v})$ *denote the number of bits exchanged under* Π *on the instance* $(\boldsymbol{u}, \boldsymbol{v})$ *of the game* (U, V)*. The* **communication complexity** $C(U, V)$ *of the communication game* (U, V) *is the minimum over all protocols* Π *of the maximum number of bits exchanged under* Π *on any instance of* (U, V)*; that is,*

$$C(U, V) = \min_{\Pi} \max_{\boldsymbol{u} \in U, \boldsymbol{v} \in V} \Pi(\boldsymbol{u}, \boldsymbol{v})$$

Note that there is always a position i, $1 \leq i \leq n$, such that $u_i \neq v_i$ since $U \cap V = \emptyset$.

The communication game models a search problem; given disjoint sets of n-tuples, U and V, the two players search for an input variable on which the two n-tuples differ. A related communication game measures the exchange of information to obtain the value of a function $f : X \times Y \mapsto Z$ on two variables in which one player has a value in X and the other has a value in Y. The players must acquire enough information about each other's variable to compute the function.

Every communication problem (U, V), where $U, V \subseteq B^n$, can be solved with communication complexity $C(U, V) \leq n + \lceil \log_2 n \rceil$ by the following protocol:

- Player I sends $\boldsymbol{u}$ to Player II.
- Player II determines a position in which $\boldsymbol{u} \neq \boldsymbol{v}$ and sends it to Player I using $\lceil \log_2 n \rceil$ bits.

This bound can be improved to $C(U,V) \leq n + \log_2^* n$, where $\log_2^* n$ is the number of times that $\lceil \log_2 \rceil$ must be taken to reduce n to zero. (See Problem 9.39.) The **log-star function** $\log_2^* n$ grows very slowly. For example, $\log_2^* 10^{10^{10}}$ is 8; by contrast, $\left\lceil \log_2 10^{10^{10}} \right\rceil =$ 33,219,280,949.

These concepts are illustrated by the **parity communication problem** (U,V), defined below, where $n = 2^k$:

$$U = \{\boldsymbol{u} \mid \boldsymbol{u} \text{ has an even number of 1s}\}$$
$$V = \{\boldsymbol{v} \mid \boldsymbol{v} \text{ has an odd number of 1s}\}$$

The following protocol achieves a communication complexity bound of $C(U,V) \leq 2\log_2 n$ for this problem. Later we show it is best possible.

1. If $n = 1$, the players know where their tuples differ and no communication is necessary.
2. If $n > 1$, go to the next step.
3. Player I sends the parity of the first $n/2$ bits of $\boldsymbol{u}$ to Player II.
4. Since $\boldsymbol{u} \neq \boldsymbol{v}$, with one bit Player II tells Player I of half of the variables on which $\boldsymbol{u}$ and $\boldsymbol{v}$ are known to differ. Play is resumed at the first step with the half of the variables on which they are known to differ.

Let $\kappa(n)$ denote the number of bits exchanged with this protocol. Then $\kappa(1) = 0$ and $\kappa(n) \leq \kappa(n/2) + 2$, whose solution is $\kappa(n) = 2\log_2 n$. Thus, $C(U,V) = \kappa(n) \leq 2\log_2 n$.

9.7.2 General Depth and Communication Complexity

We now establish a relationship between the depth $D_{\Omega_0}(f)$ of a Boolean function $f : \mathcal{B}^n \mapsto \mathcal{B}$ over the standard basis Ω_0 and the communication complexity of a communication game in which $U = f^{-1}(0)$ and $V = f^{-1}(1)$, where $f^{-1}(a)$ is the set of n-tuples for which f has value a. Theorem 9.7.1 asserts that $D_{\Omega_0}(f)$ and $C(f^{-1}(0), f^{-1}(1))$ have exactly the same value. Later we establish a similar result for monotone functions realized over the monotone basis. We divide this result into two lemmas that are proved separately.

THEOREM 9.7.1 *For every Boolean function $f : \mathcal{B}^n \mapsto \mathcal{B}$,*

$$D_{\Omega_0}(f) = C(f^{-1}(0), f^{-1}(1))$$

The communication game allows the two players to have unlimited computing power at their disposal. Thus, the protocol they employ can be an arbitrarily complex function. This power reflects the non-uniformity in the circuit model.

LEMMA 9.7.1 *For all Boolean functions $f : \mathcal{B}^n \mapsto \mathcal{B}$ and all $U, V \subseteq \mathcal{B}^n$ such that $U \subseteq f^{-1}(0)$ and $V \subseteq f^{-1}(1)$, the following bound holds:*

$$C(U,V) \leq D_{\Omega_0}(f)$$

Proof In this lemma we demonstrate that a protocol for the communication game $(f^{-1}(0), f^{-1}(1))$ can be constructed from a circuit of minimal depth for the Boolean function f. We

assume that such a circuit has negations only on input variables. By Problem 9.11 there is such a circuit.

Given an instance defined by $\boldsymbol{u} \in f^{-1}(0)$ and $\boldsymbol{v} \in f^{-1}(1)$, the players follow a path from the circuit output to an input at which $\boldsymbol{u}$ and $\boldsymbol{v}$ differ. The invariant that applies at each step is that Player I (which holds $\boldsymbol{u}$) simulates an AND gate whose value on $\boldsymbol{u}$ is 0 whereas Player II (which holds $\boldsymbol{v}$) simulates an OR gate whose value on $\boldsymbol{v}$ is 1. The bits transmitted by one player to the other specify which input to the current gate to follow on the way from the output vertex to an input vertex of the circuit for f.

The proof is by induction. The base case applies to those Boolean functions f for which $D_{\Omega_0}(f) = 0$. In this case f is either x_i or $\overline{x}_i$ for some i where x_i is an input variable of f. Thus, for each instance of the problem, both players know in advance a variable (namely, x_i) on which $\boldsymbol{u}$ and $\boldsymbol{v}$ differ. Hence, $C(U, V) = 0$ and the base case is established.

For the induction step, either $f = f_1 \wedge f_2$ or $f = f_1 \vee f_2$. Consider the first case; the second is treated in a similar fashion. Obviously $D_{\Omega_0}(f) = \max(D_{\Omega_0}(f_1), D_{\Omega_0}(f_2)) + 1$. (We are considering circuits of minimal depth.) Let $U_j = U \cap f_j^{-1}(0)$ for $j = 1, 2$. Since (U_j, V) is a communication game associated with f_j (f_j must have value 1 on V) and $D_{\Omega_0}(f_j) < D_{\Omega_0}(f)$, by induction $C(U_j, V) \leq D_{\Omega_0}(f_j)$.

Since the output gate is AND (the other case is treated similarly), both f_1 and f_2 have value 1 on V, but at least one of them has value 0 on U. We use the following protocol for (U, V): Player I sends 0 if $\boldsymbol{u} \in U_1$ (associated with the input f_1 to this AND gate) and 1 if $\boldsymbol{u} \in U_2$ (associated with the input f_2). (If the output gate is OR, we observe that at least one of f_1 and f_2 has value 1 on V and define $V_1 = V \cap f_1^{-1}(1)$ and $V_2 = V \cap f_2^{-1}(1)$. Player II sends a bit to specify the set containing $\boldsymbol{v}$.) After the first move the players follow the protocol for the f_j defined by the bit sent by Player I. Thus, when the output gate is AND the following bound holds:

$$C(U, V) \leq 1 + \max_{j=1,2}(C(U_j, B)) \leq 1 + \max(D_{\Omega_0}(f_1), D_{\Omega_0}(f_2)) = D_{\Omega_0}(f)$$

The same bound holds when the output gate is OR. ■

We now prove the second half of Theorem 9.7.1.

LEMMA 9.7.2 *Let $U, V \subseteq \mathcal{B}^n$ be such that $U \cap V = \emptyset$. Then there exists a Boolean function $f : \mathcal{B}^n \mapsto \mathcal{B}$ with $U \subseteq f^{-1}(0)$ and $V \subseteq f^{-1}(1)$ such that the following bound holds:*

$$D_{\Omega_0}(f) \leq C(U, V)$$

Proof In this proof we show how to define a Boolean function and a circuit for it from a protocol for (U, V). From the protocol a tree is constructed. The root is associated with the player who sends the first bit. As in the proof of Lemma 9.7.1, Player I is associated with AND gates and Player II with OR gates. Thus, if the protocol specifies that Player I makes the first move, the root is labeled AND. The two possible descendants are labeled with the player who makes the next transmission or by a variable or its negation (the answer) if this is the last transmission under the protocol. The function associated with the protocol is the function computed by the circuit so constructed.

We establish the result by induction. The base case applies to sets U and V for which $C(U, V) = 0$. In this case, there is an index i known in advance to both players on which $\boldsymbol{u} \in U$ and $\boldsymbol{v} \in V$ differ. Since either $u_i = 1$ or $u_i = 0$ for all $\boldsymbol{u} \in U$ (v_i has the

complementary value for all $\boldsymbol{v} \in V$), let $f = \overline{x}_i$ in the first case and $f = x_i$ in the second. Thus, in the first case (the second case is treated similarly) $U \subseteq f^{-1}(0)$, $V \subseteq f^{-1}(1)$ and $D_{\Omega_0}(f) = 0$. This establishes the base case.

For the induction step, without loss of generality, let Player I send the first bit. (The other case is treated similarly.) For some partition of $U = U_0 \cup U_1$, $U_0 \cap U_1 = \emptyset$, Player I sends a 0 if $\boldsymbol{u} \in U_0$ and a 1 if $\boldsymbol{u} \in U_1$, after which the players play with the best protocol for each subcase. It follows that

$$C(U, V) = 1 + \max_{j=1,2}(C(U_j, V))$$

Since $C(U_j, V) < C(U, V)$ for $j = 1, 2$, by induction there exist Boolean functions f_1 and f_2 such that $U_j \subseteq f_j^{-1}(0)$ and $V \subseteq f_j^{-1}(1)$ and $D_{\Omega_0}(f_j) \leq C(U_j, V)$ for $j = 1, 2$. Since the output vertex is assumed to be AND, $f = f_1 \wedge f_2$, f has value 1 only when both f_1 and f_2 have value 1 and has value 0 when either f_1 or f_2 have value 0. Thus, we have

$$\begin{aligned} V &\subseteq f_1^{-1}(1) \cap f_2^{-1}(1) = f^{-1}(1) \\ U = U_1 \cup U_2 &\subseteq f_1^{-1}(0) \cup f_2^{-1}(0) = f^{-1}(0) \end{aligned}$$

from which we conclude that

$$D_{\Omega_0}(f) \leq 1 + \max(D_{\Omega_0}(f_1), D_{\Omega_0}(f_2)) \leq 1 + \max_{j=1,2}(C(U_j, V)) = C(U, V)$$

which is the desired result. ■

This establishes the connection between the depth of a Boolean function f over the standard basis Ω_0 and the communication complexity associated with the sets $f^{-1}(0)$ and $f^{-1}(1)$.

We now draw some conclusions from Theorem 9.7.1. From the observation made above that $C(U, V) \leq n + \log_2^* n$ for an arbitrary communication problem (U, V) when $U, V \in \mathcal{B}^n$, we have that $D_{\Omega_0}(f) \leq n + \log_2^* n$ for all $f : \mathcal{B}^n \mapsto \mathcal{B}$. A better upper bound of $D_{\Omega_0}(f) \leq n+1$ is given in Theorem 2.13.1. The best upper bound of $n - \log_2 \log_2 n + O(1)$ has been derived by Gaskov [110], matching the lower bound of $n - \Theta(\log \log n)$ derived in Theorem 2.12.2.

The parity communication problem described above is defined in terms of the two sets that are the inverse images of the parity function $f_{\oplus}^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}$. As stated in Problem 9.28, this function has a formula size of at least n^2. Since $D_{\Omega}(f) \geq \log_2 L_{\Omega_0}(f)$ (Theorem 9.2.2), it follows that $D_{\Omega}(f_{\oplus}^{(n)}) \geq 2 \log_2 n$, which matches the upper bound on the communication complexity of the parity communication problem. Thus the protocol given earlier for this problem is optimal.

We now introduce the monotone communication game and develop a relationship between its complexity and the depth of monotone functions over a monotone basis.

9.7.3 Monotone Depth and Communication Complexity

We specialize Theorem 9.7.1 to monotone functions by using the fact that if $f : \mathcal{B}^n \mapsto \mathcal{B}$ is monotone and there are two n-tuples $\boldsymbol{u}$ and $\boldsymbol{v}$ such that $f(\boldsymbol{u}) = 0$ and $f(\boldsymbol{v}) = 1$, then there exists an index i, $1 \leq i \leq n$, such that $u_i < v_i$, that is, $u_i = 0$ and $v_i = 1$.

The binary n-tuple $\boldsymbol{x}$ can be defined by the set $\{i \mid x_i = 1\}$ of indices on which variables have value 1. This is a subset of $[n] = \{1, 2, \ldots, n\}$. Let $2^{[n]}$ be the power set of $[n]$, that

is, the set of all subsets of $[n]$. A **monotone minterm** (**monotone maxterm**) is a minimal set of indices of variables that if set to 1 (0) cause f to assume value 1 (0). (The variables of a monotone minterm are variables in a monotone prime implicant of f.) Let $min(f)$ and $max(f)$ be the set of monotone minterms and monotone maxterms of f, respectively. Observe that $min(f) \cap max(f) \neq \emptyset$ because if they have no elements in common, f can be made to assume values 0 and 1 simultaneously for some assignment to the variables of f, a contradiction.

DEFINITION 9.7.2 *A* **monotone communication game** (A, B) *is defined by sets* $A, B \subseteq 2^{[n]}$. *An instance of the game is a pair* (a, b) *where* $a \in A$ *and* $b \in B$. a *is assigned to Player I and* b *is assigned to Player II. Players alternate sending messages as in the communication game, using a predetermined protocol. The* **goal** *of the problem is to find an integer* $i \in a \cap b$. *The* **communication complexity**, $C_{\text{mon}}(A, B)$, *is defined as the minimum over all protocols* Π *of the maximum number of bits exchanged under* Π *on any instance of* (A, B)*:*

$$C_{\text{mon}}(A, B) = \min_{\Pi} \max_{a \in A, b \in B} \Pi(a, b)$$

We now establish a relationship between this complexity measure and the circuit depth of a Boolean function.

THEOREM 9.7.2 *For every monotone Boolean function* $f : \mathcal{B}^n \mapsto \mathcal{B}$,

$$D_{\Omega_{\text{mon}}}(f) = C(f^{-1}(0), f^{-1}(1)) = C_{\text{mon}}(min(f), max(f))$$

Proof We show that $D_{\Omega_{\text{mon}}}(f) = C(f^{-1}(0), f^{-1}(1))$ by specializing Lemmas 9.7.1 and 9.7.2 to monotone functions. In the base case of Lemma 9.7.1 since the circuit is monotone we always discover a coordinate such that $u_i = 0$ and $v_i = 1$ and negations are not needed. Thus, $C(f^{-1}(0), f^{-1}(1)) \leq D_{\Omega_{\text{mon}}}(f)$. In Lemma 9.7.2, since the protocol provides a coordinate i such that $u_i = 0$ and $v_i = 1$, the circuit defined by it is monotone and $D_{\Omega_{\text{mon}}}(f) \leq C(f^{-1}(0), f^{-1}(1))$.

We show that $C(f^{-1}(0), f^{-1}(1)) = C_{\text{mon}}(min(f), max(f))$ in two stages. First we show that $C_{\text{mon}}(min(f), max(f)) \leq C(f^{-1}(0), f^{-1}(1))$. This follows because, given any $a \in min(f)$ and $b \in max(f)$, we extend a and b to binary n-tuples $\boldsymbol{u}$ and $\boldsymbol{v}$ for which $u_r = 0$ for $r \in a$ and $v_s = 1$ for $s \in b$ and use the protocol for the monotone communication game to find an index i such that $u_i = 0$ and $v_i = 1$, that is, for which $i \in a \cap b$. Thus, the monotone communication game exchanges no more bits than the standard game.

To show that $C(f^{-1}(0), f^{-1}(1)) \leq C_{\text{mon}}(min(f), max(f))$, consider an instance $(\boldsymbol{u}, \boldsymbol{v})$ of (U, V) where $U = f^{-1}(0)$ and $V = f^{-1}(1)$. To solve the communication problem (U, V), let $a(\boldsymbol{u}) \in [n]$ be defined by $r \in a(\boldsymbol{u})$ if and only if $a_r = 0$ and let $b(\boldsymbol{v}) \in [n]$ be defined by $s \in b(\boldsymbol{v})$ if and only if $v_s = 1$. The goal of the standard communication game is to find an index i such that $u_i \neq v_i$. It follows from the definition of minterms and maxterms that there exist $p \in min(f)$ and $q \in max(f)$ such that $p \subseteq a$ and $q \subseteq b$. Since each player has unlimited computing resources available, computation of p and q can be done with no communication cost. Now invoke the protocol on the instance (p, q) of the monotone communication game $(min(f), max(f))$. This protocol returns an index $i \in p \cap q$ that is also an index on which $\boldsymbol{u}$ and $\boldsymbol{v}$ differ. But this is a solution to the instance of $(\boldsymbol{u}, \boldsymbol{v})$ of $(f^{-1}(0), f^{-1}(1))$. Thus, no more bits are communicated to solve

the standard communication game than are exchanged with the monotone communication game when the sets U and V are the inverse images of a monotone Boolean function. ■

In the next section we use the above result to derive a large lower bound on the monotone depth of the clique function.

9.7.4 The Monotone Depth of the Clique Function

In this section we illustrate the use of the monotone communication game by showing that in this game at least $\Omega(\sqrt{k})$ bits must be exchanged between two players to compute the clique function $f^{(n)}_{\text{clique},k} : \mathcal{B}^{n(n-1)/2} \mapsto \mathcal{B}$ defined in Section 9.6 when $k \leq (n/2)^{2/3}$. The inputs to $f^{(n)}_{\text{clique},k}$ are variables associated with the edges of a graph on n vertices. If an edge variable $e_{i,j} = 1$, the edge between vertices i and j is present. Otherwise, it is absent. By Theorem 9.7.2, a lower bound of $\Omega(\sqrt{k})$ on the number of bits that must be exchanged between the two players to compute $f^{(n)}_{\text{clique},k}$ implies that $f^{(n)}_{\text{clique},k}$ has depth $\Omega(\sqrt{k})$.

THE RULES OF THE GAME Fix n and k. The players in this communication game are each given sets of edges of graphs on n vertices. Player I is given a set of edges that contains a k-clique (an input on which $f^{(n)}_{\text{clique},k}$ has value 1, a **positive instance**) whereas Player II is given a set of edges that does not contain a k-clique (an input on which it has value 0, a **negative instance**). The goal of the game is to exchange the minimum number of bits for the worst-case instances to permit the players to identify an edge variable that is 1 on a positive instance and 0 on a negative one. This number of bits is the communication complexity of the game.

To derive the lower bound on communication complexity, we restrict the graphs under consideration by choosing them so that every protocol must exchange a lot of data (this cannot make the worst cases any worse). In particular, we give Player I only k-cliques, the set of graphs, CLQ, whose only edges are those between an arbitrary set of k vertices. We call Player I the **clique player**. Also, we give Player II a $(k-1)$-coloring drawn from the set COL of all possible assignments of $k-1$ colors to the n vertices of a graph G. The interpretation of a $(k-1)$-coloring is that two vertices can have the same color only if there is no edge between them. Thus, any graph that has a $(k-1)$-coloring cannot contain a k-clique because the k vertices in such a subgraph must have different colors. We call Player II the **color player**. The goal now becomes for the two players to find a monochromatic edge (both endpoints have the same color) owned by the clique player.

In the standard communication game players alternate exchanging binary messages. We simplify our discussion by assuming that each player transmits one bit simultaneously on each round. We then find a lower bound on the number of rounds and use this as a lower bound on the number of bits exchanged between the two players.

AN ADVERSARIAL STRATEGY We describe an adversarial strategy for the selection of cliques and colorings that insures that many rounds are needed for the two players to arrive at a decision. To present the strategy, we need some notation.

Let CLQ_0 denote the set of graphs $G = (V, E)$ on n vertices that contain only those edges in a k-clique. It follows that CLQ_0 contains $\binom{n}{k}$ graphs. Let COL_0 denote the set of $(k-1)$-colorings of graphs on n vertices, that is, $\text{COL}_0 = \{c \mid c : V \mapsto [k-1]\}$, where $[k-1]$ denotes the set $\{1, 2, \ldots, k-1\}$. It follows that COL_0 contains $(k-1)^n$ $(k-1)$-colorings.

We execute a series of rounds. During each round each player provides one bit of information to the other. This information has the effect of reducing the uncertainty of the color player about the possible k-cliques held by the clique player and of reducing the uncertainty of the clique player about the possible $(k-1)$-colorings held by the color player. The adversary makes the uncertainty large after each round so that the number of rounds needed will be large and a structure of the sets of cliques and colorings that can be analyzed will be maintained. The game ends when both players have found a monochromatic edge that is in a clique.

Let $P_t \subseteq V$ and $M_t \subseteq V$ denote the vertices that after the tth round are present in every k-clique and missing from every k-clique, respectively. (Let $p_t = |P_t|$ and $m_t = |M_t|$.) Since vertices in M_t are not in any cliques after the tth round, as we shall see, each such vertex can be assigned the same color as a "friend" after all vertices not in M_t have been colored. Also, after the tth round the vertices in a k-clique consist of vertices in $V - M_t$ of which those in P_t are the same for all such cliques.

Let $\mathrm{CLQ}(V, P_t, M_t)$ denote the set of k-cliques containing P_t but no vertex in M_t. Let $\mathrm{COL}(V, M_t)$ denote the $(k-1)$-colorings of vertices not in M_t after the tth round. Then $|\mathrm{CLQ}(V, P_t, M_t)| = \binom{n-p_t-m_t}{k-p_t-m_t}$ and $|\mathrm{COL}(V, M_t)| = (n-m_t)^{k-1}$ are the maximum numbers of k-cliques and $(k-1)$-colorings that are possible after the tth round. Let CLQ_t and COL_t denote the actual number of cliques and colorings that are consistent with the information exchanged between players after the tth round.

Given two sets A and B, $A \subseteq B$, we introduce a measure $\mu_B(A) = |A|/|B|$ used in deriving our lower bound. For an element $x \in A$, $\mu_B(A)$ is a rough measure of the amount of information that can be deduced about x. The smaller the value of $\mu_B(A)$, the more information we have about x. This measure is specialized to cliques and colorings after the tth round:

$$\mu_{\mathrm{CLQ}(V,P_t,M_t)}(\mathrm{CLQ}_t) = |\mathrm{CLQ}_t|/|\mathrm{CLQ}(V, P_t, M_t)|$$
$$\mu_{\mathrm{COL}(V,M_t)}(\mathrm{COL}_t) = |\mathrm{COL}_t|/|\mathrm{COL}(V, M_t)|$$

Since the color player does not know the identity of vertices in P_t until after the tth round, its information about the clique held by the other player is measured by p_t and $\mu_{\mathrm{CLQ}(V,P_t,M_t)}(\mathrm{CLQ}_t)$. Since the clique player only knows the color of vertices M_t that are missing in all cliques after the tth round, its information about a $(k-1)$-coloring by the color player is measured by m_t and $\mu_{\mathrm{COL}(V,M_t)}(\mathrm{COL}_t)$.

The number of rounds, T, is large if for $t = T$ no edge present in all remaining cliques CLQ_t that is monochromatic in all remaining colorings COL_t. We show that an adversary can choose the sets CLQ_t and COL_t at each round so that many rounds are needed.

SELECTION OF THE SETS CLQ_T AND COL_T BY THE ADVERSARY: Let the value of the bits sent by the clique and color players be b_{CLQ} and b_{COL}, respectively. At the tth round the following algorithm is used to choose CLQ_t and COL_t:

1) Let $P = P_{t-1}$, $p = p_t$, $M = M_{t-1}$ and $m = m_{t-1}$. Let CLQ^1 be the larger of the two subsets of CLQ_{t-1} consistent with the values $b_{\mathrm{CLQ}} = 0$ and $b_{\mathrm{CLQ}} = 1$. Thus, $\mu_{\mathrm{CLQ}(V,P,M)}(\mathrm{CLQ}^1) \geq \mu_{\mathrm{CLQ}(V,P,M)}(\mathrm{CLQ}_{t-1})/2$.

2) Let CLQ be a collection of k-cliques. Then the set of cliques q in CLQ containing the vertex v is denoted $\mathrm{CLQ}(v) = \{q \in \mathrm{CLQ} \mid v \in q\}$.

Let $\mathrm{CLQ} = \mathrm{CLQ}^1$. As long as there exists $v \in V - P - M$ such that the following is true:

$$\mu_{\mathrm{CLQ}(V,P,M)}(\mathrm{CLQ}(v)) \geq \frac{2(k-p-m)}{(n-p-m)}\mu_{\mathrm{CLQ}(V,P,M)}(\mathrm{CLQ}) \tag{9.2}$$

replace P by $P^* = P \cup \{v\}$, p by $p^* = p + 1$, and CLQ by $\mathrm{CLQ}^* = \mathrm{CLQ}(v)$. Here $(k-p-m)\mu_{\mathrm{CLQ}(V,P,M)}(\mathrm{CLQ})/(n-p-m)$ is the average of $\mu_{\mathrm{CLQ}(V,P,M)}(\mathrm{CLQ}(v))$ over all $v \in V - P - M$. Thus, $\mathrm{CLQ}(v)$ has measure at least twice the average.

Since $|\mathrm{CLQ}(V, P^*, M)| = (k-p-m)|\mathrm{CLQ}(V, P, M)|/(n-p-m)$ after each iteration of this loop, the following bound holds:

$$\mu_{\mathrm{CLQ}(V,P^*,M)}(\mathrm{CLQ}^*) \geq 2\mu_{\mathrm{CLQ}(V,P,M)}(\mathrm{CLQ})$$

That is, the renormalized measure of the set of cliques after one iteration of the loop is at least double that of the measure before the iteration.

After exiting from this loop let $\mathrm{CLQ}_t^* = \mathrm{CLQ}^*$ and let $P_t = P$. Since P_t contains $p_t - p_{t-1}$ more items than P_{t-1}, the following inequality holds:

$$\begin{aligned}\mu_{\mathrm{CLQ}(V,P_t,M_t)}(\mathrm{CLQ}_t^*) &\geq 2^{p_t-p_{t-1}}\mu_{\mathrm{CLQ}(V,P_{t-1},M_{t-1})}(\mathrm{CLQ}^1) \\ &\geq 2^{p_t-p_{t-1}}\mu_{\mathrm{CLQ}(V,P_{t-1},M_{t-1})}(\mathrm{CLQ}_{t-1})/2 \end{aligned} \tag{9.3}$$

Furthermore, for any vertex v remaining in $V - P$ the condition expressed in (9.2) is violated, so that the following holds for $v \in V - P$, where $\alpha = 2(k - p_t - m_{t-1})/(n - p_t - m_{t-1})$:

$$\mu_{\mathrm{CLQ}(V,P_t,M_{t-1})}(\{q \in \mathrm{CLQ}_t^* \mid v \in q\}) < \alpha\left(\mu_{\mathrm{CLQ}(V,P_t,M_{t-1})}(\mathrm{CLQ}_t^*)\right) \tag{9.4}$$

3) Let $\mathrm{COL}_t^* = \{c \in \mathrm{COL}_{t-1} \mid c \text{ is 1-1 on } P_t\}$. That is, COL_t^* is the set of $(k-1)$-colorings in COL_t that assigns unique colors to vertices in P_t. By restricting the $(k-1)$-colorings we do not increase the number of rounds. In Lemma 9.7.3 we develop a lower bound on $\mu_{\mathrm{COL}(V,M_{t-1})}(\mathrm{COL}_t^*)$ in terms of $\mu_{\mathrm{COL}(V,M_{t-1})}(\mathrm{COL}_{t-1})$.

4) Let $M = M_{t-1}$ and $m = m_{t-1}$. Let COL^0 and COL^1 denote the subsets of COL_t^* consistent with the values $b_{\mathrm{COL}} = 0$ and $b_{\mathrm{COL}} = 1$, respectively. Let COL be the larger of these two sets. Then $\mu_{\mathrm{COL}(V,M)}(\mathrm{COL}) \geq \mu_{\mathrm{COL}(V,M)}(\mathrm{COL}_t^*)/2$.

5) The set $\mathrm{COL}_t(u, v) = \{c \in \mathrm{COL} \mid c(u) = c(v)\}$ contains those $(k-1)$-colorings in COL for which vertices u and v have the same color.

As long as there exist $u, v \in V - M$ such that the following is true:

$$\mu_{\mathrm{COL}(V,M)}(\mathrm{COL}_t(u, v)) \geq 2\mu_{\mathrm{COL}(V,M)}(\mathrm{COL})/(k-1)$$

let w be one of u and v that is not in P (they cannot both be in P and have the same color because each coloring is 1-1 on P); replace M by $M^* = M \cup \{w\}$, m by $m^* = m + 1$, and COL by $\mathrm{COL}^* = \mathrm{COL}_t(u, v)$.

The term $\mu_{\mathrm{COL}(V,M)}(\mathrm{COL})/(k-1)$ is the average of $\mu_{\mathrm{COL}(V,M)}(\mathrm{COL}_t(u, v))$ over all u and v in $V - M$. Thus, COL^* contains $(k-1)$-colorings whose measure is at least twice the average.

Since $|\mathrm{COL}(V, M^*)| = |\mathrm{COL}(V, M)|/(k-1)$ after each iteration of this loop, the following holds:

$$\mu_{\mathrm{COL}(V,M^*)}(\mathrm{COL}^*) \geq 2\mu_{\mathrm{COL}(V,M)}(\mathrm{COL})$$

That is, the renormalized measure of the set of $(k-1)$-colorings after each loop iteration is at least double that of the measure before the iteration.

After exiting from this loop, let $M_t = M$. Since M_t contains $m_t - m_{t-1}$ more items than M_{t-1}, the following inequality holds:

$$\begin{aligned}\mu_{\mathrm{COL}(V,M_{t-1})}(\mathrm{COL}^*) &\geq 2^{m_t - m_{t-1}}\mu_{\mathrm{COL}(V,M_{t-1})}(\mathrm{COL}) \\ &\geq 2^{m_t - m_{t-1}}\mu_{\mathrm{COL}(V,M_{t-1})}(\mathrm{COL}_t^*)/2 \qquad (9.5)\end{aligned}$$

6) Let $\mathrm{COL}_t = \mathrm{COL}^*$, $M_t = M$, and $\mathrm{CLQ}_t = \{q \in \mathrm{CLQ}_t^* \mid M_t \cap q = \emptyset\}$. Thus, CLQ_t does not contain any cliques with vertices in M_t. In Lemma 9.7.4 we develop a lower bound on $\mu_{\mathrm{CLQ}(V,P_t,M_{t-1})}(\mathrm{CLQ}_t)$ in terms of $\mu_{\mathrm{CLQ}(V,P_t,M_{t-1})}(\mathrm{CLQ}_t^*)$.

PERFORMANCE OF THE ADVERSARIAL STRATEGY We establish three lemmas and then derive the lower bound on the number of rounds of the communication game.

LEMMA 9.7.3 *After step 3 of the adversarial selection the following inequality holds:*

$$\mu_{\mathrm{COL}(V,M_{t-1})}(\mathrm{COL}_t^*) \geq \left(1 - \frac{(p_t+1)^2}{k-1}\right)\mu_{\mathrm{COL}(V,M_{t-1})}(\mathrm{COL}_{t-1})$$

Proof Recall the definition of $\mathrm{COL}_t(u,v) = \{c \in \mathrm{COL} \mid c(u) = c(v)\}$. Consider the results of step 3 of the tth round in the adversary selection process. Because of the choices made in step 5 in the $(t-1)$st round and the choice of COL_0, the following inequality holds for all $t > 0$ and $u, v \in V - M_{t-1}$ when $u \neq v$:

$$\mu_{\mathrm{COL}(V,M_{t-1})}(\mathrm{COL}_t(u,v)) < 2\mu_{\mathrm{COL}(V,M_{t-1})}(\mathrm{COL}_{t-1})/(k-1)$$

Because $M_t = M_{t-1}$ at step 3 of the tth round and $P_t \subseteq V - M_t$, the same bound applies for u and v in P_t.

The set COL_{t-1} is reduced to $\mathrm{COL}_t^* = \{c \in \mathrm{COL}_{t-1} \mid c \text{ is 1 to 1 on } P_t\}$ by discarding $(k-1)$-colorings for which u and v are in P_t and have the same color. From the above facts the following inequalities hold (here instances of the measure μ carry the subscript $\mathrm{COL}(V, M_{t-1})$):

$$\begin{aligned}\mu(\mathrm{COL}_t^*) &= \mu(\{c \in \mathrm{COL}_{t-1} \mid c \text{ is 1 to 1 on } P_t\}) \\ &= \mu(\mathrm{COL}_{t-1}) - \mu\left(\bigcup_{u,v\in P_t,\ u\neq v} \mathrm{COL}_t(u,v)\right) \\ &\geq \mu(\mathrm{COL}_{t-1}) - \sum_{u,v\in P_t,\ u\neq v} \mathrm{COL}_t(u,v) \\ &> \left(1 - \binom{p_t}{2}\frac{2}{k-1}\right)\mu(\mathrm{COL}_{t-1}) \\ &> \left(1 - \frac{(p_t+1)^2}{k-1}\right)\mu(\mathrm{COL}_{t-1})\end{aligned}$$

From this the conclusion follows. ■

LEMMA 9.7.4 *After step 6 of the adversarial selection the following inequality holds:*

$$\mu_{\mathrm{CLQ}(V,P_t,M_{t-1})}(\mathrm{CLQ}_t) \geq \left(1 - \frac{2km_t}{n}\right) \mu_{\mathrm{CLQ}(V,P_t,M_{t-1})}(\mathrm{CLQ}_t^*)$$

Proof As stated in (9.4), after step 2 of the tth round of the adversary selection process we have for all $v \in V - P_t - M_{t-1}$ the following inequality:

$$\mu_{\mathrm{CLQ}(V,P_t,M_{t-1})}(\{q \in \mathrm{CLQ}_t^* \mid v \in q\}) < \frac{2(k - p_t - m_{t-1})}{(n - p_t - m_{t-1})} \mu_{\mathrm{CLQ}(V,P_t,M_{t-1})}(\mathrm{CLQ}_t^*)$$

Since $M_t \subseteq V - P_t$, this bound applies to $v \in M_t$. In the rest of this proof all instances of μ carry the subscript $\mathrm{CLQ}(V, P_t, M_{t-1})$.

Since $\mathrm{CLQ}_t = \{q \in \mathrm{CLQ}_t^* \mid M_t \cap q = \emptyset\}$, after step 6 the following inequalities hold:

$$\begin{aligned}
\mu(\mathrm{CLQ}_t) &= \mu(\{c \in \mathrm{CLQ}_t \mid M_t \cap q = \emptyset\}) \\
&= \mu(\mathrm{CLQ}_t^*) - \mu\left(\bigcup_{v \in M_t} \{c \in \mathrm{CLQ}_t^* \mid v \in q\}\right) \\
&\geq \left(1 - \frac{2(k - p_t - m_{t-1})m_t}{(n - p_t - m_{t-1})}\right) \mu(\mathrm{CLQ}_t^*) \\
&\geq \left(1 - \frac{2km_t}{n}\right) \mu(\mathrm{CLQ}_t^*)
\end{aligned}$$

From this the conclusion follows. ■

The third lemma sets the stage for the principal result of this section.

LEMMA 9.7.5 *Let $k \geq 2$ and $t \leq \sqrt{k}/4$ and $t \leq n/(8k)$. Then the following inequalities hold:*

$$\mu_{\mathrm{CLQ}(V,P_t,M_{t-1})}(\mathrm{CLQ}_t) \geq 2^{p_t - 2t}$$
$$\mu_{\mathrm{COL}(V,M_t)}(\mathrm{COL}_t) \geq 2^{m_t - 2t}$$

Proof The inequalities hold for $t = 0$ because $\mu_{\mathrm{CLQ}(V,P_0)}(\mathrm{CLQ}_0) = \mu_{\mathrm{COL}(V,M_0)}(\mathrm{COL}_0) = 1$. We assume as inductive hypothesis that the inequalities hold for the first $t - 1$ rounds and show they hold for the tth round as well.

Using the inductive hypothesis and (9.3), we have

$$\mu_{\mathrm{CLQ}(V,P_t,M_{t-1})}(\mathrm{CLQ}_t^*) \geq 2^{p_t - p_{t-1}} \mu_{\mathrm{CLQ}(V,P_{t-1},M_{t-1})}(\mathrm{CLQ}_{t-1})/2 \geq 2^{p_t - 2t + 1} \quad (9.6)$$

Since $\mu_{\mathrm{CLQ}(V,P_t)}(\mathrm{CLQ}_t^*) \leq 1$, we conclude that $p_t \leq 2t - 1$. Using this result, the assumption that $t \leq \sqrt{k}/4$, Lemma 9.7.3, and the inductive hypothesis, we have

$$\begin{aligned}
\mu_{\mathrm{COL}(V,M_{t-1})}(\mathrm{COL}_t^*) &\geq \left(1 - \frac{4t^2}{k-1}\right) \mu_{\mathrm{COL}(V,M_{t-1})}(\mathrm{COL}_{t-1}) \\
&\geq \left(1 - \frac{k}{4(k-1)}\right) \mu_{\mathrm{COL}(V,M_{t-1})}(\mathrm{COL}_{t-1}) \\
&\geq \frac{1}{2} \mu_{\mathrm{COL}(V,M_{t-1})}(\mathrm{COL}_{t-1}) \\
&\geq 2^{m_{t-1} - 2t + 1}
\end{aligned}$$

Combining this and (9.5) (note that in step 6 we let $\mathrm{COL}_t = \mathrm{COL}^*$), we have the first of the two desired conclusions, namely $\mu_{\mathrm{COL}(V,M_t)}(\mathrm{COL}_t) \geq 2^{m_t-2t}$. This implies that $m_t \leq 2t$. Applying this to the inequality in Lemma 9.7.4 and using the condition $t \leq n/(8k)$, we get the following inequality:

$$\mu_{\mathrm{CLQ}(V,P_t,M_{t-1})}(\mathrm{CLQ}_t) \geq \mu_{\mathrm{CLQ}(V,P_t,M_{t-1})}(\mathrm{CLQ}_t^*)/2$$

Combining this with the lower bound given in (9.6), we have the second of the two desired conclusions, namely, $\mu_{\mathrm{CLQ}(V,P_t,M_{t-1})}(\mathrm{CLQ}_t) \geq 2^{p_2-2t}$. ■

We now state the principal conclusion of this section.

THEOREM 9.7.3 *Let $2 \leq k \leq (n/2)^{2/3}$. Then the monotone communication complexity of the k-clique function $f^{(n)}_{\mathrm{clique},k}$ is $\Omega(\sqrt{k})$.*

Proof Run the adversarial selection process for $T = \sqrt{k}/4$ steps to produce sets CLQ_T, COL_T, P_T, and M_T. Below we show that CLQ_T and COL_T are not empty. Give the clique player a k-clique $q \in \mathrm{CLQ}_T$ and the color player a $(k-1)$-coloring $c \in \mathrm{COL}_T$. To show that the two players cannot agree in T or fewer rounds on an edge in a clique in CLQ_T that is monochromatic in all $c \in \mathrm{COL}_T$, assume they can, and let $(u, v) \in q$ be that edge. If follows that both u and v are in M_T. But this cannot happen because, by construction, $q \cap M_T = \emptyset$.

To show that CLQ_T and COL_T are not empty, observe that $k \leq (n/2)^{2/3}$ and $t \leq \sqrt{k}/4$ imply that $t \leq n/(8k)$. Thus, Lemma 9.7.5 can be invoked, which implies that $p_t, m_t \leq 2t \leq \sqrt{k}/2 \leq k/2 < n$. Invoking the definitions, the following inequalities also hold.

$$\mathrm{CLQ}_t \geq 2^{p_t-2t}\mathrm{CLQ}(V, P_t, M_{t-1}) > 0$$
$$\mathrm{COL}_t \geq 2^{m_t-2t}\mathrm{COL}(V, M_t) > 0$$

Since the right-hand sides are non-zero, we have the desired conclusion. ■

9.7.5 Bounded-Depth Circuits

As explained earlier, bounded-depth circuits are studied to help us understand the depth of bounded fan-in circuits. Bounded-depth circuits for arbitrary Boolean functions require that the fan-in of some gates be unbounded because otherwise only a bounded number of inputs can influence the output(s).

In Section 2.3 we encountered the DNF, CNF, SOPE, POSE, and RSE normal forms. Each of these corresponds to a circuit of bounded depth. The DNF and SOPE normal forms represent Boolean functions as the OR of the AND of literals. The OR and each of the ANDs is a function of a potentially unbounded number of literals. The same statement applies to the CNF and POSE normal forms when AND and OR are exchanged. The RSE normal form represents Boolean functions as the EXCLUSIVE OR of the AND of variables, that is, without the use of negation. Again, the fan-in of the two types of operation is potentially unbounded. As stated in Problems 2.8 and 2.9, the SOPE and POSE of the parity function $f^{(n)}_{\oplus}$ have exponential size, as does the RSE of the OR function $f^{(n)}_{\vee}$. In Problem 2.10 it is stated that the function $f^{(n)}_{\mathrm{mod}\ 3}$ has exponential size in the DNF, CNF, and RSE normal forms.

In this section we show that every bounded-depth circuit for the parity function $f_{\oplus}^{(n)}$ over the basis containing the NOT gate on one input and the AND and OR gates on an arbitrary number of inputs has exponential size. Thus, the depth-2 result extends to arbitrary depth.

BOUNDED-DEPTH PARITY CIRCUITS HAVE EXPONENTIAL SIZE We use an approximation method to derive a lower bound on the size of a bounded-depth circuit for $f_{\oplus}^{(n)}$. This method parallels almost exactly the method of Section 9.6.3. Starting with gates most distant from the output and progressing toward it, replace each gate of a given circuit by an approximating circuit. We show that as each replacement is made, the number of new errors it introduces is small. However, we also show that after all gates are approximated, the number of errors between the approximating circuit and $f_{\oplus}^{(n)}$ is large. This implies that the number of gates replaced is large.

The approximation method used here replaces each gate in a circuit by a polynomial over $GF(3)$, the three-element field containing $\{-1, 0, 1\}$, with the property that if the variables of such a polynomial assume values in $\mathcal{B} = \{0, 1\}$, the value of the polynomial is in $\mathcal{B}$. For example, the polynomial $x_1(1 - x_2)x_3$ has value 1 over $\mathcal{B}$ only when $x_1 = x_3 = 1$ and $x_2 = 0$ and has value 0 otherwise. Thus, it corresponds exactly to the minterm $x_1\overline{x}_2x_3$. Since every minterm can be represented as a polynomial of this kind, every Boolean function f can realized by a polynomial over $GF(3)$ by forming the sum of one such polynomial for each of its minterms. A ***b*-approximator** is polynomial of degree b that approximates a Boolean function.

Although we establish the lower bound for the basis containing NOT and the unbounded fan-in AND and OR gates, the result continues to hold if the unbounded fan-in MOD_3 function is added to the basis. (See Problem 9.41.) We begin by showing that the function computed by a circuit C containing $size(C)$ gates cannot differ from its b-approximator on too many input tuples.

LEMMA 9.7.6 *Let $f : \mathcal{B}^n \mapsto \mathcal{B}$ be computed by a circuit C of depth d. There is a $(2k)^d$-approximator circuit $\widehat{C}$ computing $\widehat{f} : \mathcal{B}^n \mapsto \mathcal{B}$ such that f and $\widehat{f}$ differ on at most $size(C)2^{n-k}$ input n-tuples, where n is the number of inputs on which C depends and $size(C)$ is the number of gates that it contains.*

Proof We construct a b-approximator for C, $b = (2k)^d$, by approximating inputs (x_i and $\overline{x}_i$ are approximated exactly on $\mathcal{B}$ by x_i and $(1 - x_i)$), after which we approximate gates all of whose inputs have been approximated until the output gate has been approximated. We establish the result of the lemma by induction.

We treat the statement of the lemma as our inductive hypothesis and show that if it holds for $d = D - 1$, it holds for $d = D$. The hypothesis holds on inputs, namely, when $d = 0$. Suppose the hypothesis holds for $d = D - 1$. Since C has depth d, each of the inputs to the output gate has depth at most $D - 1$ and satisfies the hypothesis. The output gate is AND, OR, or NOT. Suppose it is NOT. Let g be the function associated with its input. We replace the NOT gate with the function $(1 - g)$, which introduces no new errors. Since g and $1 - g$ have the same degree, the inductive hypothesis holds in this case.

If the output gate is the AND of $g_1, g_2, \ldots, g_m$, it can be represented exactly by the function $g_1g_2 \cdots g_m$. However, this polynomial has degree $m(2k)^{d-1}$ if each of its inputs has degree at most $(2k)^{d-1}$; this violates the inductive hypothesis if $m > 2k$, which may happen because the fan-in of the gate is potentially unbounded. Thus we must introduce some error in order to reduce the degree of the approximating polynomial. Since the OR of

$g_1, g_2, \ldots, g_m$ can be represented by $1-(1-g_1)(1-g_2)\cdots(1-g_m)$ using DeMorgan's Rules, both AND and OR of $g_1, g_2, \ldots, g_m$ have the same degree. We find an approximating polynomial for both AND and OR by approximating the OR gate.

We approximate the OR of $g_1, g_2, \ldots, g_m$ by creating subsets $S_1, S_2, \ldots, S_k$ of $\{g_1, g_2, \ldots, g_m\}$, computing $f_i = (\sum_{j\in S_i} g_j)^2$, and combining these results in

$$\text{OR}(f_1, f_2, \ldots, f_k) = 1-(1-f_1)(1-f_2)\cdots(1-f_k)$$

The degree of this approximation is $2k$ times the maximal degree of any polynomial in the set $\{g_1, g_2, \ldots, g_m\}$ or at most $(2k)^d$, the desired result.

There is no error in this approximation if the original OR has value 0. We now show that there exist subsets $S_1, S_2, \ldots, S_k$ such that the error is at most 2^{n-k} when the original OR has value 1. Let's fix on a particular input n-tuple $\boldsymbol{x}$ to the circuit. Suppose each subset is formed by deciding for each function in $\{g_1, g_2, \ldots, g_m\}$ with probability $1/2$ whether or not to include it in the set. If one or more of $\{g_1, g_2, \ldots, g_m\}$ is 1 on $\boldsymbol{x}$, the probability of choosing a function for set whose value is 1 is at least $1/2$. Thus, the probability that $\text{OR}(f_1, f_2, \ldots, f_k)$ has value 0 when the original OR has value 1 is the probability that each of $f_1, f_2, \ldots, f_k$ has value 0, which is at most 2^{-k}. Since the sets $\{S_1, S_2, \ldots, S_k\}$ result in an error on input $\boldsymbol{x}$ with probability at most 2^{-k}, the average number of errors on input $\boldsymbol{x}$, averaged over all choices for the k sets, is at most 2^{-k} and the average number of errors on the set of 2^n inputs is at most 2^{n-k}. It follows that some set $\{S_1, S_2, \ldots, S_k\}$ (and a corresponding approximating function) has an incorrect value on at most 2^{n-k} inputs. Since by the inductive hypothesis at most $(size(C)-1)2^{n-k}$ errors occur on all but the output gate, at most $size(C)2^{n-k}$ errors occur on the entire circuit. ■

The next result demonstrates that a $\sqrt{n}$-approximator (obtained by letting $k = n^{1/2d}/2$) and the parity function must differ on many inputs. This is used to show that the circuit being approximated must have many gates.

LEMMA 9.7.7 *Let $\widehat{f}: \mathcal{B}^n \mapsto \mathcal{B}$ be a $\sqrt{n}$-approximator for $f_\oplus^{(n)}$. Then, $\widehat{f}$ and $f_\oplus^{(n)}$ differ on at least $2^n/50$ input n-tuples.*

Proof Let $U \subseteq \mathcal{B}^n$ be the n-tuples on which the functions agree. We derive an upper bound on $|U|$ of $\beta = (49)2^n/50$ that implies the lower bound of the lemma. We derive this bound indirectly. Since there are $3^{|U|}$ functions $g: U \mapsto \{-1, 0, 1\}$, assign each one a different polynomial and show that the number of such polynomials is at most 3^β, which implies that $|U| \leq \beta$.

Transform the polynomial in the variables $x_1, x_2, \ldots, x_n$ representing $f_\oplus^{(n)}$ by mapping x_i to $y_i = 2x_i - 1$. This mapping sends 1 to 1 and 0 to -1. (Observe that $y_i^2 = 1$.) It does not change the degree of a polynomial. In these new variables $f_\oplus^{(n)}$ can be represented exactly by the polynomial $y_1 y_2 \cdots y_n$.

Given a function $g: U \mapsto \{-1, 0, 1\}$, extend it arbitrarily to a function $\widetilde{g}: \mathcal{B}^n \mapsto \{-1, 0, 1\}$. Let p be a polynomial in $Y = \{y_1, y_2, \ldots, y_n\}$ that represents $\widetilde{g}$ on U exactly. Let $cy_{i_1}y_{i_2}\cdots y_{i_t}$ be a term in p for some constant $c \in \{-1, 1\}$. We show that if t is larger than $n/2$ we can replace this term with a smaller-degree term.

Let $T = \{y_{i_1}, y_{i_2}, \ldots, y_{i_t}\}$ and $\overline{T} = Y - T$. The term $cy_{i_1}y_{i_2}\cdots y_{i_t}$ can be written as $c\Pi T$, where by ΠT we mean the product of all terms in T. With $y_i^2 = 1$, this may be rewritten as $c\Pi Y \Pi \overline{T}$. Since $f_\oplus^{(n)} = \Pi Y$, on the set U this is equivalent to $c\widehat{f}\Pi\overline{T}$,

which has degree $\sqrt{n} + n - |\overline{T}|$. Thus, a term $cy_{i_1}y_{i_2}\cdots y_{i_t}$ of degree $t \geq n/2$ can be replaced by a term of degree $\sqrt{n} + n - t$. It follows that the number of polynomials (and functions) representing functions whose values coincide with $f_{\oplus}^{(n)}$ on U is the number of polynomials of degree at most $\sqrt{n} + n/2$. Since there are $\binom{n}{j}$ ways to choose a term containing j variables of Y, there are at most N ways to choose polynomials representing functions $g : U \mapsto \{-1, 0, 1\}$, where N satisfies the following bound:

$$N \leq \sum_{j=0}^{\sqrt{n}+(n/2)} \binom{n}{j}$$

For sufficiently large n, the bound to N is approximately $0.9772 \cdot 2^n < (49/50)2^n$. (See Problem 9.7.) Since each of the N terms can be included in a polynomial with coefficient -1, 0, or 1, there are at most 3^N distinct polynomials and corresponding functions $g : U \mapsto \{-1, 0, 1\}$, which is the desired conclusion. ■

We summarize these two results in Theorem 9.7.4.

THEOREM 9.7.4 *Every circuit of depth d for the parity function $f_{\oplus}^{(n)}$ has a size exceeding $2^{n^{1/2d}}/2/50$ for sufficiently large n.*

Proof Let U be the set of n-tuples on which $f_{\oplus}^{(n)}$ and its approximation $\widehat{f}$ differ. From Lemma 9.7.6, $|U|$ is at most size$(C)2^{n-k}$. Now let $k = n^{1/2d}/2$. From Lemma 9.7.7 these two functions must differ on at least $\frac{1}{50}2^n$ input n-tuples. Thus, size$(C)2^{n-k} \geq \frac{1}{50}2^n$ from which the conclusion follows. ■

Problems

MATHEMATICAL PRELIMINARIES

9.1 Show that the following identity holds for integers r and L:

$$\left\lceil \frac{L}{r+1} \right\rceil + \left\lfloor \frac{rL}{r+1} \right\rfloor = L$$

9.2 Show that a rooted tree of maximal fan-in r containing k internal vertices has at most $k(r-1)+1$ leaves and that a rooted tree with l leaves and fan-in r has at most $l-1$ vertices with fan-in 2 or more and at most $2(l-1)$ edges.

9.3 For positive integers n_1, n_2, a_1, and a_2, show that the following identity holds:

$$\frac{n_1^2}{a_1} + \frac{n_2^2}{a_2} \geq \frac{(n_1+n_2)^2}{(a_1+a_2)}$$

9.4 The **external path length** $e(T, L)$ of a binary tree T with L leaves is the sum of the lengths of the paths from the root to the leaves. Show that $e(T, L) \geq L\lceil \log_2 L \rceil - 2^{\lceil \log_2 L \rceil} + L$.

Hint: Argue that the external path length is minimal for a nearly balanced binary tree. Use this fact and a proof by induction to obtain the external path length of a binary tree with $L = 2^k$ for some integer k. Use this result to establish the above statement.

9.5 For positive integers r and s, show that $\lceil s/r \rceil (s \bmod r) + \lfloor s/r \rfloor (r - s \bmod r) = s$.
Hint: Use the fact that for any real number a, $\lceil a \rceil - \lfloor a \rfloor = 1$ if a is not an integer and 0 otherwise. Also use the fact that $s \bmod r = s - \lfloor s/r \rfloor \cdot r$.

9.6 (**Binomial Theorem**) Show that the coefficient of the term $x^i y^{n-i}$ in the expansion of the polynomial $(x + y)^n$ is the binomial coefficient $\binom{n}{i}$. That is,

$$(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^i y^{n-i}$$

9.7 Show that the following sum is closely approximated by $0.4772 \cdot 2^n$ for large n:

$$\sum_{i=(n/2)}^{(n/2)+\sqrt{n}} \binom{n}{i}$$

Hint: Use the fact that $n!$ can be very closely approximated by $\sqrt{2\pi n}\, n^n e^{-n}$ to approximate $\binom{n}{i}$. Then approximate a sum by an integral (see Problem 2.23) and consult tables of values for the **error function** $\operatorname{erf}(x) = \int_0^x e^{-t^2} dt$.

9.8 Let $0 \leq x \leq y$. Show that $x + \sqrt{y - x} \geq \sqrt{y}$.

CIRCUIT MODELS AND MEASURES

9.9 Provide an algorithm that produces a formula for each circuit of fan-out 1 over a basis that has fan-in of at most 2.

9.10 Show that any monotone Boolean function $f^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}$ can be expanded on its first variable as

$$f(x_1, x_2, \ldots, x_n) = f(0, x_2, \ldots, x_n) \vee (x_1 \wedge f(1, x_2, \ldots, x_n))$$

9.11 Show that a circuit for a Boolean function (one output vertex) over the standard basis can be transformed into one that uses negation only on inputs by at most doubling the number of AND, OR, and NOT gates and without changing its depth by more than a constant factor.
Hint: Find the two-input gate closest to the output gate that is connected to a NOT gate. Change the circuit to move the NOT gate closer to the inputs.

RELATIONSHIPS AMONG COMPLEXITY MEASURES

9.12 Using the construction employed in Theorem 9.2.1, show that the depth of a function $f : \mathcal{B}^n \mapsto \mathcal{B}^m$ in a circuit of fan-out s over a complete basis Ω of fan-in r satisfies the inequality

$$D_{s,\Omega}(f) \leq D_\Omega(f)\,(1 + l(\Omega) + l(\Omega) \log_s (rC_{s,\Omega}(f)/D))$$

9.13 Show that there are ten functions f with $L_\Omega(f) = 2$ that are dependent on two variables and that each can be realized from a circuit for $f^{(1)}_{\text{mux}}$ plus at most one instance of NOT on an input to $f^{(1)}_{\text{mux}}$ and on its output.

9.14 Extend the upper bound on depth versus formula size of Theorem 9.2.2 to monotone functions.

LOWER-BOUND METHODS FOR GENERAL CIRCUITS

9.15 Show that the function $f(x_1, x_2, \ldots, x_n) = x_1 \wedge x_2 \wedge \cdots \wedge x_n$ has circuit size $\lceil (n-1)/(r-1) \rceil$ and depth $\lceil \log_r n \rceil$ over the basis containing the r-input AND gate.

9.16 The parity function $f^{(n)}_{\oplus} : \mathcal{B}^n \mapsto \mathcal{B}$ has value 1 when an odd number of its variables have value 1 and 0 otherwise. Derive matching upper and lower bounds on the size and depth of the smallest and shallowest circuit(s) for $f^{(n)}_{\oplus}$ over the basis B_2.

9.17 Show that the function $f^{(n)}_{\text{mod } 4}$ defined to have value 1 if the sum of the n inputs modulo 4 is 1 can be realized by a circuit over the basis B_2 whose size is $2.5n + O(1)$.
Hint: Show that the function is symmetric and devise a circuit to compute the sum of three bits as the sum of two bits.

9.18 Over the basis B_2 derive good upper and lower bounds on the circuit size of the functions $f^{(n)}_4 : \mathcal{B}^n \mapsto \mathcal{B}$ and $f^{(n)}_5 : \mathcal{B}^n \mapsto \mathcal{B}$ defined as

$$f^{(n)}_4 = ((y+2) \bmod 4) \bmod 2$$
$$f^{(n)}_5 = ((y+2) \bmod 5) \bmod 2$$

Here $y = \sum_{i=1}^{n} x_i$ and $\sum$ and $+$ denote integer addition.

9.19 Show that the set of Boolean functions on two variables that depend on both variables contains only AND-type and parity-type functions. Here an AND-**type function** computes $(x^a \wedge y^b)^c$ for Boolean constants a, b, c whereas a **parity-type function** computes $x \oplus y \oplus c$ for some Boolean constant c.

9.20 The threshold function $\tau^{(n)}_t : \mathcal{B}^n \mapsto \mathcal{B}$ on n inputs has value is 1 if t or more inputs are 1 and 0 otherwise. Show that over the basis B_2 that $C_{B_2}(\tau^{(n)}_2) \geq 2n - 4$.

9.21 A formula for the parity function $f^{(n)}_{\oplus,c} : \mathcal{B}^n \mapsto \mathcal{B}$ on n inputs is given below. Show that it has circuit size exactly $3(n-1)$ over the standard basis when NOT gates are not counted:

$$f^{(n)}_{\oplus,c} = x_1 \oplus x_2 \oplus \cdots \oplus x_n \oplus c$$

9.22 Show that $f^{(n)}_{\oplus,c}$ has circuit size exactly $4(n-1)$ over the standard basis when NOT gates are counted.

9.23 Show that $f^{(n)}_{\oplus,c}$ has circuit size exactly $7(n-1)$ over the basis $\{\wedge, \neg\}$.

LOWER BOUNDS TO FORMULA SIZE

9.24 Show that the multiplexer function $f_{\text{mux}}^{(p)}$ can be realized by a formula of size $32^p - 2$ in which the total number of address variables is $2(2^p - 1)$.

Hint: Expand the function $f_{\text{mux}}^{(p)}$ as suggested below, where $\boldsymbol{a}^{(k)}$ denotes the k components of $\boldsymbol{a}$ with smallest index and $P = 2^p$:

$$f_{\text{mux}}^{(p)}(\boldsymbol{a}^{(p)}, y_{P-1}, \ldots, y_0) = f_{\text{mux}}^{(1)}(a_{p-1}, f_{\text{mux}}^{(p-1)}(\boldsymbol{a}^{(p-1)}, y_{P-1}, \ldots, y_{P/2}), f_{\text{mux}}^{(p-1)}(\boldsymbol{a}^{(p-1)}, y_{P/2-1}, \ldots, y_0))$$

Also, represent $f_{\text{mux}}^{(1)}$ as shown below.

$$f_{\text{mux}}^{(1)}(a, y_1, y_0) = (\overline{a} \wedge y_0) \vee (a \wedge y_1)$$

9.25 Show that Nečiporuk's method cannot provide a lower bound larger than $O(n^2/\log n)$ for a function on n variables.

9.26 Derive a quadratic upper bound on the formula size of the parity function $f_{\oplus}^{(n)}$ over the standard basis.

9.27 Nečiporuk's function is defined in terms of an $\lceil n/m \rceil \times m$ matrix of Boolean variables, $\boldsymbol{X} = \{x_{i,j}\}$, $m = \lceil \log_2 n \rceil + 2$, and a matrix $\boldsymbol{\Sigma} = \{\sigma_{i,j}\}$ of the same dimensions in which each entry $\sigma_{i,j}$ is a distinct m-tuple over $\mathcal{B}$ containing at least two 1s. Nečiporuk's function, $N(\boldsymbol{X})$, is defined as

$$N(\boldsymbol{X}) = \bigoplus_{i,j} x_{i,j} \bigwedge \bigoplus_{\substack{k=1 \\ (k \neq i)}} \prod_{\substack{l \text{ such that} \\ \sigma_{i,j}(l)=1}} x_{k,l}$$

Here $\bigoplus$ denotes the **exclusive or** operation. Show that this function has formula size $\Omega(n^2/\log n)$ over the basis B_2.

9.28 Use Krapchenko's method to derive a lower bound of n^2 on the formula size of the parity function $f_{\oplus}^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}$.

9.29 Use Krapchenko's method to derive a lower bound of $\Omega(t(n-t+1))$ on the formula size over the standard basis of the threshold function $\tau_t^{(n)}$, $1 \leq t \leq n-1$.

9.30 Generalize Krapchenko's lower-bound method as follows. Let $f : \mathcal{B}^n \mapsto \mathcal{B}$ and let $A \subseteq f^{-1}(0)$ and $B \subseteq f^{-1}(1)$. Let $Q = [q_{i,j}]$ be defined by $q_{i,j} = 1$ if $\boldsymbol{x}_i \in A$ and $\boldsymbol{x}_j \in B$ are neighbors and $q_{i,j} = 0$ otherwise. Let $P = QQ^T$ and $\overline{P} = Q^TQ$. Then $p_{r,s}$ is the number of common neighbors to $\boldsymbol{x}_r$ and $\boldsymbol{x}_s$ in B. The matrices P and $\overline{P}$ are symmetric and their largest eigenvalues, $\lambda(P)$ and $\lambda(\overline{P})$, are both non-negative and $\lambda(P) = \lambda(\overline{P})$. Show that

$$L_\Omega(f) \geq \lambda(P)$$

9.31 Under the conditions of Problem 9.30, let

$$D(f) = \frac{1}{|B|} \sum_{r,s} p_{r,s}, \quad \overline{D}(f) = \frac{1}{|B|} \sum_{r,s} \overline{p}_{r,s}, \quad K(f) = \frac{|\mathcal{N}(A,B)|^2}{|A||B|}$$

where $K(f)$ is the lower bound given in Theorem 9.4.2. Show that

$$K(f) \leq D(f) \leq \lambda(P)$$
$$K(f) \leq \overline{D}(f) \leq \lambda(\overline{P})$$

Hint: Use the fact that the largest eigenvalue of a matrix P satisfies

$$\lambda(P) = \max_{x \neq 0} \frac{\boldsymbol{x}^T P \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}$$

Also, let s_i be the sum of the elements in the ith column of the matrix Q. Show that $\sum_i s_i^2 = \sum_{r,s} p_{r,s}$.

LOWER-BOUND METHODS FOR MONOTONE CIRCUITS

9.32 Consider a monotone circuit on n inputs that computes a monotone Boolean function $f : \mathcal{B}^n \mapsto \mathcal{B}$. Let the circuit have k two-input AND gates, one of them the output gate, and let these gates compute the Boolean functions $g_1, g_2, \ldots, g_k = f$, where the AND gates are inverse-ordered by their distance from the output gate computing f. Since the function g_j is computed using the values of $x_1, x_2, \ldots, x_n, g_1, \ldots, g_{j-1}$, show that g_j can be computed using at most $n + j - 2$ two-input OR gates and one AND gate. Show that this implies the following upper bound on the monotone circuit size of f:

$$C_{\Omega_{\text{mon}}}(f) \leq kn + \binom{k-1}{2} - 1$$

Let $C_\wedge(f)$ denote the minimum number of AND gates used to realize f over the monotone basis. This result implies the following relationship:

$$C_{\Omega_{\text{mon}}}(f) = O\left((C_\wedge(f))^2\right)$$

How does this result change if the gate associated with f is an OR gate?

9.33 Show that the prime implicants of a monotone function are monotone prime implicants.

9.34 Find the monotone implicants of the Boolean threshold function $\tau_t^{(n)} : \mathcal{B}^n \mapsto \mathcal{B}$, $1 \leq t \leq n$.

9.35 Using the gate-elimination method, show that $C_{\Omega_{\text{mon}}}(\tau_2^{(n)}) \geq 2n - 3$.

9.36 Show that an expansion of the form of equation (9.1) on page 420 holds for every monotone function.

9.37 Show that the $f_{\text{clique,k}}^{(n)} : \mathcal{B}^{n(n-1)/2} \mapsto \mathcal{B}$ can be realized by a monotone circuit of size $O(n^n)$.

9.38 Show that the largest value assumed by $\min(\sqrt{k-1}/2, n/(2k))$ under variation of k is $\Omega(n^{1/3})$.

CIRCUIT DEPTH

9.39 Show that the communication complexity of a problem (U, V), $U, V \subseteq B^n$, satisfies $C(U, V) \leq n + \log_2^* n$, where $\log_2^* n$ is the number of times that $\lceil \log_2 \rceil$ must be taken to reduce n to zero.

Hint: Complete the definition of a protocol in which Player I sends Player II $n - \lceil \log_2 n \rceil$ bits on the first round and Player II responds with a message specifying whether or not its n-tuple agrees with that of Player I and if not, where they differ.

9.40 Consider the communication problem defined by the following sets:

$$U = \{\boldsymbol{u} \mid 3 \text{ divides the number of 1s in } \boldsymbol{u}\}$$
$$V = \{\boldsymbol{v} \mid 3 \text{ does not divide the number of 1s in } \boldsymbol{u}\}$$

Show that a protocol exists that solves this problem with communication complexity $3\lceil \log_2 n \rceil$.

9.41 Show that Theorem 9.7.4 continues to hold when the MOD_3 function is added to the basis where MOD_3 is the Boolean function that has value 1 when the number of 1s among its inputs is not divisible by 3.

Chapter Notes

The dependence of circuit size on fan-out stated in Theorem 9.2.1 is due to Johnson et al. [150]. The depth bound implied by this result is proportional to the product of the depth and the logarithm of the size of the original circuit. Hoover et al. [138] have improved the depth bound so that it is proportional to $(\log_r s) D_\Omega(f)$ without sacrificing the size bound of [150].

The relationship between formula size and depth in Theorem 9.2.2 is due to Spira [314], whose depth bound has a coefficient of proportionality of 2.465 over the basis of all Boolean functions on two variables. Over the basis of all Boolean functions except for parity and its complement, Preparata and Muller [259] obtain a coefficient of 1.81. Brent, in a paper on the parallelization of arithmetic formulas [58], has effectively extended the relationship between depth and formula size to monotone functions. (See also [359].)

An interesting relationship between complexity measures that is omitted from Section 9.2, due to Paterson and Valiant [240], shows that circuit size and depth satisfy the inequality

$$D_\Omega(f) \geq \frac{1}{4} C_\Omega(f) \log C_\Omega(f) - O(C_\Omega(f))$$

The lower bounds of Theorem 9.3.2 on functions in $Q_{2,3}^{(n)}$ are due to Schnorr [300], whereas that of Theorem 9.3.3 on the multiplexer function is due to Paul [244]. Blum [48], building on the work of Schnorr [302], has obtained a lower bound of $3(n-1)$ for a particular function of n variables over the basis B_2. This is the best circuit-size lower bound for this basis. Zwick [374] has obtained a lower bound of $4n$ for certain symmetric functions over the basis U_2. Red'kin [274] has obtained lower bounds with coefficients as high as 7 for certain functions over the bases $\{\wedge, \neg\}$ and $\{\vee, \neg\}$. (See Problem 9.23.) Red'kin [276] has used the gate-elimination method to show that the size of the ripple-adder circuit of Section 2.7 cannot be improved.

The coefficient of Nečiporuk's lower-bound method [230] in Theorem 9.4.1 has been improved upon by Paterson (unpublished) and Zwick [373]. Paul [244] has applied Nečiporuk's method to show that the indirect storage access function has formula size $\Omega(n^2/\log n)$ over the basis B_2. Nečiporuk's method has also been applied to many other problems, including the determinant [169], the marriage problem [126], recognition of context-free languages [241], and the clique function [304].

The proof of Krapchenko's lower bound [174] given in Theorem 9.4.2 is due to Paterson, as described by Bopanna and Sipser [50]. Koutsoupias [172] has obtained the results of Problems 9.30 and 9.31, improving upon the Krapchenko lower bounds for the kth threshold function by a factor of at least 2. Andreev [24], building on the work of Subbotovskaya [320], has improved upon Krapchenko's method and exhibits a lower bound of $\Omega(n^{2.5-\epsilon})$ on a function of n variables for every fixed $\epsilon > 0$ when n is sufficiently large. Krichevskii [176] has shown that over the standard basis, $\tau_t^{(n)}$ requires formula size $\Omega(n \log n)$, which beats Krapchenko's lower bound for small and large values of t.

Symmetric functions are examined in Section 2.11 and upper bounds are given on the circuit size of such functions over the basis $\{\wedge, \vee, \oplus\}$. Polynomial-size formulas for symmetric functions are implicit in the work of Ofman [234] and Wallace [356], who also independently demonstrated how to add two binary numbers in logarithmic depth. Krapchenko [175] demonstrated that all symmetric Boolean functions have formula size $O(n^{4.93})$ over the standard basis. Peterson [247], improving upon the results of Pippenger [248] and Paterson [241], showed that all symmetric functions have formula size $O(n^{3.27})$ over the basis B_2. Paterson, Pippenger, and Zwick [242,243] have recently improved these results, showing that over B_2 and U_2 formulas exist of size $O(n^{3.13})$ and $O(n^{4.57})$, respectively, for many symmetric Boolean functions including the majority function, and of size $O(n^{3.30})$ and $O(n^{4.85})$, respectively, for all symmetric Boolean functions.

Markov demonstrated that the minimal number of negations needed to realize an arbitrary binary function on n variables with an arbitrary number of output variables, maximized over all such functions, is at most $\lceil \log_2(n+1) \rceil$. For Boolean functions (they have one output variable) it is at most $\lfloor \log_2(n+1) \rfloor$. Fischer [100] has described a circuit whose size is at most twice that of an optimal circuit plus the size of a circuit that computes $f_{\mathrm{NEG}}(x_1, \ldots, x_n) = (\overline{x}_1, \ldots, \overline{x}_n)$ and whose depth is at most that of the optimal circuit plus the depth of a circuit for f_{NEG}. He exhibits a circuit for f_{NEG} of size $O(n^2 \log n)$ and depth $O(\log n)$. This is the result given in Theorem 9.5.1. Tanaka and Nishino [323] have improved the size bound on f_{NEG} to $O(n \log^2 n)$ at the expense of increasing the depth bound to $O(\log^2 n)$. Beals, Nishino, and Tanaka [32] have further improved these results, deriving simultaneous size and depth bounds of $O(n \log n)$ and $O(\log n)$, respectively.

Using non-constructive methods, a series of upper bounds have been developed on the monotone formula size of the threshold functions $\tau_t^{(n)}$ by Valiant [346] and Bopanna [49], culminating in bounds by Khasin [166] and Friedman [106] of $O(t^{4.3} n \log n)$ over the monotone basis. With constructive methods, Ajtai, Komlós, and Szemerédi [14] obtained polynomial bounds on the formula size $\tau_t^{(n)}$ over the monotone basis. Using their construction, Friedman [106] has obtained a bound on formula size over the monotone basis of $O(t^c n \log n)$ for c a large constant.

Over the basis B_2, Fischer, Meyer, and Paterson [101] have shown that the majority function $\tau_t^{(n)}$, $t = \lceil n/2 \rceil$, and other symmetric functions require formula size $\Omega(n \log n)$. Pudlák [264], building on the work of Hodes and Specker [136], has shown that all but 16 symmetric

Boolean functions on n variables require formula size $\Omega(n \log\log n)$ over the same basis. The 16 exceptional functions have linear formula size.

Using counting arguments such as those given in Section 2.12, Gilbert [114] has shown that most monotone Boolean functions on n variables have a circuit size that is $\Omega(2^n/n^{3/2})$. Red'kin [275] has shown that the lower bound can be achieved to within a constant multiplicative factor by every monotone Boolean function.

Tiekenherinrich [330] gave a $4n$ lower bound to the monotone circuit size of a simple function. Dunne [87] derived a $3.5n$ lower bound on the monotone circuit size for the majority function.

The lower bound on the monotone circuit size of binary sorting (Theorem 9.6.1) is due to Lamagna and Savage [188] using an argument patterned after that of Van Voorhis [351] for comparator-based sorting networks. Muller and Preparata [225,226] demonstrate that binary sorting over the standard basis has circuit size $O(n)$. (See Theorem 2.11.1.) Pippenger and Valiant [253] and Lamagna [187] demonstrate an $\Omega(n \log n)$ lower bound on the monotone circuit size of merging. These results are established in Section 9.6.1. The sorting network designed by Ajtai, Komlós, and Szemerédi [14] when specialized to Boolean data yields a monotone circuit of size $O(n \log n)$ for binary sorting.

The first proof that the monotone circuit size of $n \times n$ Boolean matrix multiplication (see Section 9.6.2) is $\Omega(n^3)$ was obtained by Pratt [256]. Later Paterson [238] and Mehlhorn and Galil [218] demonstrated that it is exactly $n^2(2n-1)$. Weiss [361] discovered a simple application of the function-replacement method to both Boolean convolution and Boolean matrix multiplication, as summarized in Corollary 9.6.1 and Theorem 9.6.5. (Wegener [360, p. 170] extended Weiss's result to include the number of ORs.) Wegener [357] has exhibited an n-input, n-output Boolean function (Boolean direct product) whose monotone circuit size is $\Omega(n^2)$. Earlier several authors examined the class of multi-output functions known as **Boolean sums** in which each output is the OR of a subset of inputs. Nečiporuk [231] gave an explicit set of Boolean sums and demonstrated that its monotone circuit size is $\Omega(n^{3/2})$. This lower bound for such functions was independently improved to $\Omega(n^{5/3})$ by Mehlhorn [216] and Pippenger [250]. More recently, Andreev [23] has constructed a family of Boolean sums with monotone circuit size that is $\Omega(n^{2-\epsilon})$ for every fixed $\epsilon > 0$.

The first super-polynomial lower bound on the monotone circuit size of the clique function was established by Razborov [270]. Shortly afterward, Andreev [22], using similar methods, gave an exponential lower bound on the monotone circuit size of a problem in **NP**. Because the clique function is complete with respect to monotone projections [310,344], this established an exponential lower bound for the clique function. Alon and Bopanna [17], by strengthening Razborov's method, gave a direct proof of this fact, giving a lower bound exponential in $\Omega\left((n/\log n)^{1/3}\right)$. The stronger lower bound given in Theorem 9.6.6, which is exponential in $\Omega(n^{1/3})$, is due to Amano and Maruoka [20]. They apply *bottleneck counting*, an idea of Haken [125], to establish this result. Amano and Maruoka [20] have also extended the approximation method to circuits that have negations only on their inputs and for which the number of inputs carrying negations is small. They show that, even with a small number of negations, an exponential lower bound on the circuit size of the clique function can be obtained.

Having shown that monotone circuit complexity can lead to exponential lower bounds, Razborov [271] then cast doubt on the likelihood that this approach would lead to exponential non-monotone circuit size bounds by proving that the matching problem on bipartite graphs, a problem in **P**, has a super-polynomial monotone circuit size. Tardos [324] strengthened

Razborov's lower bound, deriving an exponential one. Later Razborov [273] demonstrated that the obvious generalization of the approximation method cannot yield better lower bounds than $\Omega(n^2)$ for Boolean functions on n inputs realized by circuits over complete bases.

Berkowitz [37] introduced the concept of pseudo-inverse and established Theorem 9.6.9. Valiant [347], Wegener [358], and Paterson (unpublished — see [92,360]) independently improved upon the size of the monotone circuit realizing all pseudo-negations from $O(n^2 \log n)$ to $O(n \log^2 n)$ to produce Theorem 9.6.8. Lemma 9.6.9 is due to Dunne [90].

In his Ph.D. thesis Dunne [88] has given the most general definition of pseudo-negation. He shows that a Boolean function h is a pseudo-negation on variable x_i of a Boolean function f on the n variables $x_1, \ldots, x_n$ if and only if h satisfies

$$f(\boldsymbol{x})|_{x_i=0} \leq h(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \leq f(\boldsymbol{x})|_{x_i=1}$$

Here $f(\boldsymbol{x})|_{x_i=a}$ denotes the function obtained from f by fixing x_i at a.

Dunne [89] demonstrated that HALF-CLIQUE CENTRAL SLICE is **NP**-complete (Theorem 9.6.10) and showed that the central slices of the HAMILTONIAN CIRCUIT (there is a closed path containing each vertex once) and SATISFIABILITY are **NP**-complete. As mentioned by Dunne [91], not all **NP**-complete problems have **NP**-complete central slices.

The concept of communication complexity arose in the context of the VLSI model of computation discussed in Chapter 12. In this case it measures the amount of information that must be transmitted from the inputs to the outputs of a function. The communication game described in Section 9.7.1 is different: it characterizes a search problem because its goal is to find an input variable on which two n-tuples in disjoint sets disagree.

Yao [366] developed a method to derive lower bounds on the communication complexity of functions $f : X \times Y \mapsto Z$. He considered the matrix of values of f where the rows and columns are indexed by the values of X and Y. He defined monochromatic rectangles as submatrices in which all entries are the same. He then established that the logarithm of the minimal number of disjoint rectangles in this matrix is a lower bound on the number of bits that must be exchanged to compute f. (This result shows, for example, that the identity function $f : \mathcal{B}^{2n} \mapsto \mathcal{B}$ defined for $f(\boldsymbol{x}, \boldsymbol{y}) = 1$ if and only if $x_i = y_i$ for all $1 \leq i \leq n$ requires the exchange of at least $n + 1$ bits.) Savage [288] adapted the crossing sequence argument from one-tape Turing machines (an application of the pigeonhole principle) to derive lower bounds on predicates. Mehlhorn and Schmidt [220] show that functions $f : X \times Y \mapsto Z$ for which Z is a subset of a field have a communication complexity that is at most the rank of the two-dimensional matrix of values of f.

The development of the relationship between the circuit depth of a function and its communication complexity follows that given by Karchmer and Wigderson [157]. Karchmer [156] cites Yannakakis for independently discovering the connection $D_{\Omega_0}(f) = C(f^{-1}(0), f^{-1}(1))$ of Theorem 9.7.1 for non-monotone functions. Karchmer and Wigderson [157] have examined st-connectivity in this framework. This is the problem of determining from the adjacency matrix of an undirected graph G with n vertices and two distinguished vertices, s and t, whether there is a path from s to t. When characterized as a Boolean function on the edge variables, this is a monotone function. Karchmer and Wigderson [157] have shown that the circuit depth of this function is $\Omega((\log n)^2 / \log \log n)$, a result later improved to $\Omega((\log n)^2)$ independently by Håstad and Boppana in unpublished work. Raz and Wigderson [269] have shown via a complex proof that the clique problem on n-vertex graphs studied in Section 9.7.4 has monotone communication complexity and depth $\Omega(n)$. The simpler but weaker lower bound for this problem developed in Section 9.7.4 is due to Goldmann and Håstad [116].

Furst, Saxe, and Sipser [107] and, independently, Ajtai [13] obtained the first strong lower bounds on the size of bounded-depth circuits. They demonstrated that every bounded-depth circuit for the parity function $f_{\oplus}^{(n)}$ has superpolynomial size. Using a deeper analysis, Yao [368] demonstrated that bounded-depth circuits for $f_{\oplus}^{(n)}$ have exponential size. Håstad [124] strengthened the results and simplified the argument, giving a lower bound on circuit size of $2^{\Omega(n^{1/d}/10)}$ for circuits of depth d.

Razborov [272] examined a more powerful class of bounded-depth circuits, namely, circuits that use unbounded fan-in AND, OR, and parity functions. He demonstrated that the majority function $\tau_{n/2}^{(n)}$ has exponential size over this larger basis. Smolensky [313] simplified and strengthened Razborov's result, obtaining an exponential lower bound on the size of a bounded-depth circuit for the MOD_p function over the basis AND, OR, and MOD_q when p and q are distinct powers of primes. We use a simplified version of his result in Section 9.7.5.