9.3 The Monotone Switching Lemma

In Razborov's method of approximations one only uses DNFs to approximate gates. In this way, OR gates can be easily approximated: an OR of DNFs is a DNF, and we only need to keep its small enough. The case of AND gates is, however, more complicated. So, a natural idea to try to approximate by *both* DNFs and CNFs. When appropriately realized, this idea leads to a general, and relatively simple lower-bounds criterion for monotone circuits. Due to the symmetry between DNFs and CNFs, this criterion is often much easier to apply and yields exponential lower bounds for many functions, including the clique function.

> Still, there are functions—like the perfect matching function—for which the criterion seems to fail. This is why we will discuss Razborov's method later in Sect. 9.10 in full detail: unlike the general criterion, which we are going to present now, Razborov's method is much more subtle, tailor made for the *specific* function one deals with and can be applied in situations where the general criterion fails to produce strong lower bounds. Yet another reason to include Razborov's proof for the perfect matching function is that this function belongs to P, and the proof was never treated in a book.

Our goal is to show that, if a monotone boolean function can be computed by a small monotone circuit, then it can be approximated by small monotone CNFs and DNFs. Thus, in order to prove that a function requires large circuits it is enough to show that it does not have a small CNF/DNF approximation. The proof of this will be based on the "monotone switching lemma" allowing us to switch between CNFs and DNFs, and vice versa.

By a monotone k-CNF (conjunctive normal form) we will mean an And of an arbitrary number of monotone clauses, each being an Or of at most k variables. Dually, a monotone k-DNF is an Or of an arbitrary number of monomials, each being an And of at most k variables. In an *exact* k-CNF all clauses must have *exactly* k distinct variables; *exact* k-DNFs are defined similarly. For two boolean functions f and g of the same set of variables, we write $f \leq g$ if $f(x) \leq g(x)$ for all input vectors x. For a CNF/DNF C we will denote by $|C|$ the number of clauses/monomials in it.

The following lemma was first proved in Jukna (1999) in terms of so-called "finite limits", a notion suggested by Sipser (1985); we will also use this notion later (in Sect. 11.3) to prove lower bounds for depth-3 circuits. In terms of DNFs and CNFs the lemma was then proved by Berg and Ulfberg (1999). Later, a similar lemma was used by Harnik and Raz (2000) to improve the numerically strongest known lower bound $2^{\Omega(n^{1/3}/\log n)}$ of Andreev (1987b) to $2^{\Omega((n/\log n)^{1/3})}$. The idea of the lemma itself was also implicit in the work of Haken (1995).

Lemma 9.15. (Monotone Switching Lemma) *For every* $(s-1)$*-CNF* $f_{\rm cnf}$ *there is an* $(r-1)$*-DNF* $f_{\rm dnf}$ *and an exact* r*-DNF* D *such that*

$$f_{\rm dnf} \leq f_{\rm cnf} \leq f_{\rm dnf} \vee D \quad \text{and} \quad |D| \leq (s-1)^r \,. \tag{9.2}$$

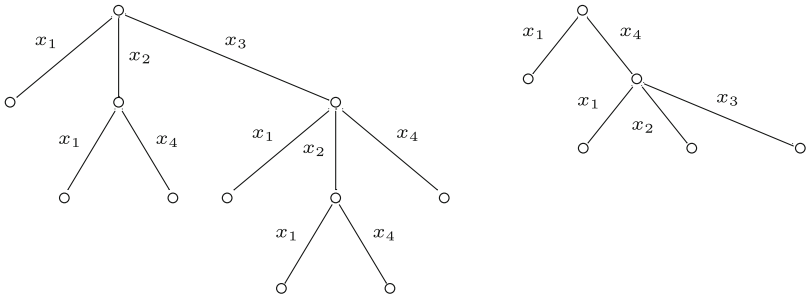

Fig. 9.2 Two DNF-trees of the same 3-CNF $f_{\text{cnf}} = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_4)$. The second tree is obtained by parsing the clauses of f_{cnf} in the reverse order

Dually, for every $(r-1)$*-DNF* f_{dnf} *there is an* $(s-1)$*-CNF* f_{cnf} *and an exact* s*-CNF* C *such that*

$$f_{\text{cnf}} \wedge C \leq f_{\text{dnf}} \leq f_{\text{cnf}} \quad \textit{and} \quad |C| \leq (r-1)^s \,. \tag{9.3}$$

Proof. We prove the first claim (the second is dual). Let $f_{\text{cnf}} = q_1 \wedge \cdots \wedge q_l$ be an $(s-1)$-CNF; hence, each clause q_i has $|q_i| \leq s-1$ variables. It will be convenient to identify clauses and monomials with the *sets* of indices of their variables. We say that a monomial p *pierces* a clause q_i if $p \cap q_i \neq \emptyset$.

We associate with f_{cnf} the following "transversal" tree T of fan-out at most $s-1$ (see Fig. 9.2).

The first node of T corresponds to the first clause q_1, and the outgoing $|q_1|$ edges are labeled by the variables from q_1. Suppose we have reached a node v, and let p be the monomial consisting of the labels of edges from the root to v. If p pierces all the clauses of f_{cnf}, then v is a leaf. Otherwise, let q_i be the *first* clause such that $p \cap q_i = \emptyset$. Then the node v has $|q_i|$ outgoing edges labeled by the variables in q_i.

Note that the resulting tree T depends on what ordering of clauses of f_{cnf} we fix, that is, in which order we parse the clauses (see Fig. 9.2). Still, for any such tree we have that, for every assignment $x \in \{0,1\}^n$, $f_{\text{cnf}}(x) = 1$ if and only if x is consistent with at least one path from the root to a leaf of T. This holds because $f_{\text{cnf}}(x) = 1$ iff the set $S_x = \{i \mid x_i = 1\}$ intersects all clauses $q_1, \ldots, q_l$.

Some paths in T may be longer than $r-1$. So, we now cut off these long paths. Namely, let f_{dnf} be the OR of all paths of length at most $r-1$ ending in leafs, and D be the OR of all paths of length exactly r. Observe that for every assignment $x \in \{0,1\}^n$:

- $f_{\text{dnf}}(x) = 1$ implies $f_{\text{cnf}}(x) = 1$, and
- $f_{\text{cnf}}(x) = 1$ implies $f_{\text{dnf}}(x) = 1$ or $D(x) = 1$.

Thus, $f_{\text{dnf}} \leq f_{\text{cnf}} \leq f_{\text{dnf}} \vee D$. Finally, we also have that $|D| \leq (s-1)^r$, because every node of T has fan-out at most $s-1$. □

Most important in the Switching Lemma is that the exact DNFs and CNFs correcting possible errors contain only $(s-1)^r$ monomials instead of all $\binom{n}{r}$ possible monomials, and only $(r-1)^s$ clauses instead of all $\binom{n}{s}$ possible clauses.

9.4 The Lower-Bounds Criterion

We now give a general lower-bounds criterion for monotone circuits.

Definition 9.16. Let f be a monotone boolean function of n variables. We say that f is *t-simple* if for every pair of integers $2 \le r, s \le n$ there exists an exact s-CNF C, an exact r-DNF D, and a subset $I \subseteq [n]$ of size $|I| \le s-1$ such that

(a) $|C| \le t \cdot (r-1)^s$ and $|D| \le t \cdot (s-1)^r$, and
(b) Either $C \le f$ or $f \le D \vee \bigvee_{i \in I} x_i$ (or both) hold.

Theorem 9.17. *If a monotone boolean function can be computed by a monotone circuit of size t, then f is t-simple.*

Proof. Let $F(x_1, \ldots, x_n)$ be a monotone boolean function, and suppose that F can be computed by a monotone circuit of size t. Our goal is to show that the function F is t-simple. To do this, fix an arbitrary pair of integer parameters $2 \le s, r \le n$.

Let $f = g * h$ be a gate in our circuit. That is, f is a function computed at some node of the circuit, and g and h are functions computed at its inputs. By an *approximator* of this gate we will mean a pair $(f_{\text{cnf}}, f_{\text{dnf}})$, where f_{cnf} is an $(s-1)$-CNF (a *left* approximator of f) and f_{dnf} is an $(r-1)$-DNF (a *right* approximator of f) such that $f_{\text{dnf}} \le f_{\text{cnf}}$.

We say that such an approximator $f_{\text{cnf}}, f_{\text{dnf}}$ of f introduces a new error on input $x \in \{0,1\}^n$ if the approximators of g and of h did not make an error on x, but the approximator of f does. That is, $g_{\text{cnf}}(x) = g_{\text{dnf}}(x) = g(x)$ and $h_{\text{cnf}}(x) = h_{\text{dnf}}(x) = h(x)$, but either $f_{\text{cnf}}(x) \ne f(x)$ or $f_{\text{dnf}}(x) \ne f(x)$.

We define approximators inductively as follows.

Case 1: f is an input variable, say, $f = x_i$. In this case we take $f_{\text{cnf}} = f_{\text{dnf}} := x_i$. It is clear that this approximator introduces no errors.

Case 2: f is an And gate, $f = g \wedge h$. In this case we take $f_{\text{cnf}} := g_{\text{cnf}} \wedge h_{\text{cnf}}$ as the left approximator of f; hence, f_{cnf} introduces no new errors. To define the right approximator of f we use Lemma 9.15 to convert f_{cnf} into an $(r-1)$-DNF f_{dnf}; hence, $f_{\text{dnf}} \le f_{\text{cnf}}$. Let E_f be the set of inputs on which f_{dnf} introduces a new error, that is,

$$E_f := \{x \mid f(x) = f_{\text{cnf}}(x) = 1 \text{ but } f_{\text{dnf}}(x) = 0\}.$$

By Lemma 9.15, all these errors can be "corrected" by adding a relatively small exact r-DNF: there is an exact r-DNF D such that $|D| \le (s-1)^r$ and $D(x) = 1$ for all $x \in E_f$.

Case 3: f is an Or gate, $f = g \vee h$. This case is dual to Case 2. We take $f_{\text{dnf}} := g_{\text{dnf}} \vee h_{\text{dnf}}$ as the right approximator of f; hence, f_{dnf} introduces no new errors. To define the left approximator of f we use Lemma 9.15 to convert f_{dnf} into an $(s-1)$-CNF f_{cnf}; hence, $f_{\text{dnf}} \le f_{\text{cnf}}$. Let E_f be the set of inputs on which f_{cnf} introduces a new error, that is,

$$E_f := \{x \mid f(x) = f_{\text{dnf}}(x) = 0 \text{ but } f_{\text{cnf}}(x) = 1\}.$$

By Lemma 9.15, all these errors can be "corrected" by adding a relatively small exact s-CNF: there is an exact s-CNF C such that $|C| \le (r-1)^s$ and $C(x) = 0$ for all $x \in E_f$.

Proceeding in this way we will reach the last gate of our circuit computing the given function F. Let $(F_{\rm cnf}, F_{\rm dnf})$ be its approximator, and let E be the set of all inputs $x \in \{0,1\}^n$ on which F differs from at least one of the functions $F_{\rm cnf}$ or $F_{\rm dnf}$. Since at input gates (= variables) no error was made, for every such input $x \in E$, the corresponding error must be introduced at some intermediate gate. That is, for every $x \in E$ there is a gate f such that $x \in E_f$ (approximator of f introduces an error on x for the first time). But we have shown that, for each gate, all these errors can be corrected by adding an exact s-CNF of size at most $(r-1)^s$ or an exact r-DNF of size at most $(s-1)^r$. Since we have only t gates, all such errors $x \in E$ can be corrected by adding an exact s-CNF C of size at most $t \cdot (r-1)^s$ and an exact r-DNF D of size at most $t \cdot (s-1)^r$, that is, for all inputs $x \in \{0,1\}^n$, we have

$$C(x) \wedge F_{\rm cnf}(x) \le F(x) \le F_{\rm dnf}(x) \vee D(x)\,,$$

where $F_{\rm dnf} \le F_{\rm cnf}$. This already implies that the function F is t-simple. Indeed, if the CNF $F_{\rm cnf}$ is empty (that is, if $F_{\rm cnf} \equiv 1$) then $C \le F$, and we are done. Otherwise, $F_{\rm cnf}$ must contain some clause q of length at most $s-1$, say, $q = \bigvee_{i\in I} x_i$ for some $I \subseteq [n]$ of size $|I| \le s-1$. Since clearly $F_{\rm cnf} \le q$, the condition $F_{\rm dnf} \le F_{\rm cnf}$ implies $F \le F_{\rm dnf} \vee D \le F_{\rm cnf} \vee D \le q \vee D$, as desired. This completes the proof of Theorem 9.17. □

In applications, boolean functions f are usually defined as set-theoretic predicates. In this case we say that f accepts a set $S \subseteq \{1,\ldots,n\}$ and write $f(S) = 1$ if and only if f accepts its incidence vector. Let $\overline{S} = \{1,\ldots,n\} \setminus S$ denote the complement of S. We say that a set S is a

- *Positive input* for f if $f(S) = 1$;
- *Negative input* for f if $f(\overline{S}) = 0$.

Put differently, a positive (negative) input is a set of variables which, if assigned the value 1 (0), forces the function to take the value 1 (0) regardless of the values assigned to the remaining variables. The minimal (under set inclusion) positive inputs for f are called *minterms* of f. Similarly, the maximal negative inputs for f are called *maxterms* of f.

Note that one and the same set S can be both a positive and a negative input! For example, if $f(x_1,x_2,x_3)$ outputs 1 iff $x_1 + x_2 + x_3 \ge 2$, then $S = \{1,2\}$ is both positive and negative input for f, because $f(1,1,x_3) = 1$ and $f(0,0,x_3) = 0$.

To re-formulate the definition of t-simplicity (Definition 9.16) in terms of positive/negative inputs, note that if C is a CNF, then $C \le f$ means that every negative input of f must contain at least one clause of C (looked at as set of indices of its variables). Similarly, $f \le D \vee \bigvee_{i\in I} x_i$ means that every positive input must either intersect the set I or contain at least one monomial of D. Thus, if $\mathcal{F}_1$ ($\mathcal{F}_0$) is a family of positive (negative) inputs of f, and $\#_k(\mathcal{F})$ denotes the maximum number