Chapter 10

Read-once functions

Martin C. Golumbic and Vladimir Gurvich

10.1 Introduction

In this chapter, we present the theory and applications of read-once Boolean functions, one of the most interesting special families of Boolean functions. A function f is called *read-once* if it can be represented by a Boolean expression using the operations of conjunction, disjunction and negation, in which every variable appears exactly once. We call such an expression a *read-once expression* for f. For example, the function

$$f_0(a, b, c, w, x, y, z) = ay \vee cxy \vee bw \vee bz$$

is a read-once function since it can be factored into the expression

$$f_0 = y(a \vee cx) \vee b(w \vee z)$$

which is a read-once expression.

Observe, from the definition, that read-once functions must be monotone (or unate), since every variable appears either in its positive or negative form in the read-once expression (see Exercise 1). However, we will make the stronger assumption that a read-once function is positive, simply by renaming any negative variable $\overline{x}_i$ as a new positive variable x'_i. Thus, every variable will be positive, and we may freely rely on the results presented earlier (in particular, in Chapters 1 and 4) on positive Boolean functions.

Let us look at two simple functions,

$$f_1 = ab \vee bc \vee cd$$

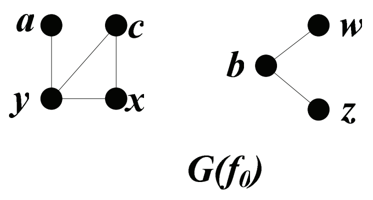

Figure 10.1: The co-occurrence graph of $f_0 = ay \vee cxy \vee bw \vee bz$.

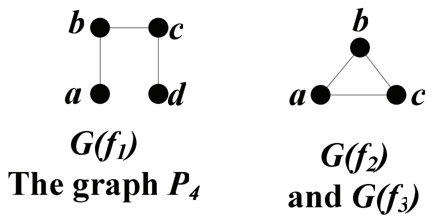

Figure 10.2: The co-occurrence graphs of f_1, f_2, f_3.

and

$$f_2 = ab \vee bc \vee ac.$$

Neither of these is a read-once function; indeed, it is impossible to express them so that each variable appears only once. (Try to do it.) The functions f_1 and f_2 illustrate the two types of forbidden functions which characterize read-once functions, as we will see. We begin by defining the co-occurrence graph of a positive Boolean function.

Let f be a positive Boolean function over the variable set $V = \{x_1, x_2, \ldots, x_n\}$. The *co-occurrence graph* of f, denoted $G(f) = (V, E)$, has vertex set V (the same as the set of variables), and there is an edge (x_i, x_j) in E if x_i and x_j occur together (at least once) in some prime implicant of f. In this chapter, we often regard a prime implicant as the set of its literals. Formally, let $\mathcal{P}$ denote the collection of prime implicants of f. Then,

$$(x_i, x_j) \in E \iff x_i, x_j \in P \text{ for some } P \in \mathcal{P}.$$

Figures 10.1 and 10.2 show the co-occurrence graphs of f_0, f_1, f_2.

We denote by P_4 the graph consisting of a chordless path on 4 vertices and 3 edges, which is the graph $G(f_1)$ in Figure 10.2 (see also Appendix A). A graph is called P_4-*free* if it contains no induced subgraph isomorphic to P_4. The P_4-free graphs are also known as *cographs* (for "complement reducible graphs"); we will have more to say about them in Section 10.4.

Since we have observed that f_1 is not read-once, and since its co-occurrence graph is P_4, it would be reasonable to conjecture that *the co-occurrence graph of a read-once function must be P_4-free.* In fact, we will prove this statement in Section 10.3. But this is not enough; in order to characterize read-once functions in terms of graphs, we will need a second property called *normality.*[1]

To see this, note that the function

$$f_3 = abc$$

has the same co-occurrence graph as f_2, namely, the triangle $G(f_2) = G(f_3)$ in Figure 10.2, yet f_3 is clearly read-once and f_2 is not read-once. This example illustrates the motivation for the following definition.

A Boolean function f is called *normal* if every clique of its co-occurrence graph is contained in a prime implicant of f.

In our example, f_2 fails to be normal, since the triangle $\{a, b, c\}$ is not contained in any prime implicant of f_2. This leads to our second necessary property of read-once functions, namely, that *a read-once function must be normal*, which we will also prove in Section 10.3. Moreover, a classical theorem of Gurvich [393, 397] shows that combining these two properties characterizes read-once functions.

Theorem 10.1. *A positive Boolean function f is read-once if and only if its co-occurrence graph $G(f)$ is P_4-free and f is normal.*

A new proof of this theorem will be given in Section 10.3 as part of Theorem 10.6.

Read-once functions first appeared explicitly in the literature in the papers of Chein [173] and Hayes [445] which gave exponential time recognition algorithms for the family (see the historical notes at the end of this chapter). Gurvich [393, 396, 397] gave the first characterization theorems for read-once functions; they will be presented in Section 10.3. Several authors have subsequently discovered and rediscovered these and a number of other characterizations. Theorem 10.1 also provides the justification for the polynomial time recognition algorithm of read-once functions by Golumbic, Mintz and Rotics [374, 375] to be presented in Section 10.5. In particular, we will show how to factor read-once functions using the properties of P_4-free graphs.

[1]The property of normality is sometimes called *clique-maximality* in the literature. It also appears in the definition of *conformal hypergraphs* in Berge [64] and is used in the theory of acyclic hypergraphs.

Read-once functions have been studied in computational learning theory where they have been shown to constitute a class that can be learned in polynomial time. Section 10.6 will survey some of these results. Additional applications of read-once functions will be presented in Section 10.7.

Before turning our full attention to read-once functions, however, we review a few properties of the dual of a Boolean function and prove an important result on positive Boolean functions that will be useful in subsequent sections.

10.2 The dual subimplicant theorem for positive Boolean functions

In this section, we first recall some of the relationships between the prime implicants of a function f and the prime implicants of its dual function f^d in the case of positive Boolean functions. All of these properties have been presented in Chapter 1 and Chapter 4. We then present a characterization of the subimplicants of the dual of a positive Boolean function, due to Boros, Gurvich and Hammer [110]. This result will be used later in the proof of one of the characterizations of read-once functions.

10.2.1 Implicants and dual implicants

The dual of a Boolean function f is the function f^d defined by

$$f^d(X) = \overline{f(\overline{X})},$$

and an expression for f^d can be obtained from any expression for f by simply interchanging the operators $\wedge$ and $\vee$ as well as the constants 0 and 1. In particular, given a DNF expression for f, this exchange yields a CNF expression for f^d. This shows that the dual of a read-once function is also read-once.

The process of transforming a DNF expression of f into a DNF expression of f^d is called *DNF dualization*; its complexity for positive Boolean functions is still unknown, the current best algorithm being quasi-polynomial [323], see Chapter 4.

Let $\mathcal{P}$ be the collection of prime implicants of a positive Boolean function f over the variables $x_1, x_2, \ldots, x_n$, and let $\mathcal{D}$ be the collection of prime implicants of the dual function f^d. We assume throughout that all of the variables for f (and hence for f^d) are essential. We use the term "dual (prime) implicant" of f to mean a (prime) implicant of f^d. For positive functions, the prime implicants of f correspond precisely to the set of minimal true points $minT(f)$, and the dual prime implicants of f correspond precisely to the set of maximal false points $maxF(f)$, see Sections 1.10.3 and 4.2.1.

Theorem 4.7 states that the implicants and dual implicants of a Boolean function f, viewed as sets of literals, have pairwise non-empty intersections. In particular, this holds for the prime implicants and the dual prime implicants. Moreover, the prime implicants and the dual prime implicants are minimal with this property, that is, for every proper subset S of a dual prime implicant of f, there is a prime implicant P such that $P \cap S = \emptyset$.

In terms of hypergraph theory, the prime implicants $\mathcal{P}$ form a clutter (i.e., a collection of sets, or hyperedges, such that no set contains another set), as does the collection of dual prime implicants $\mathcal{D}$.

Finally, we recall the following properties of duality to be used in this chapter and which can be derived from Theorems 4.1 and 4.19.

Theorem 10.2. *Let f and g be positive Boolean functions over $\{x_1, x_2, \ldots, x_n\}$, and let $\mathcal{P}$ and $\mathcal{D}$ be the collections of prime implicants of f and g, respectively. Then the following statements are equivalent:*

(i) *$g = f^d$;*

(ii) *for every partition of $\{x_1, x_2, \ldots, x_n\}$ into sets A and $\overline{A}$, there is either a member of $\mathcal{P}$ contained in A or a member of $\mathcal{D}$ contained in $\overline{A}$, but not both;*

(iii) *$\mathcal{D}$ is exactly the family of minimal transversals of $\mathcal{P}$;*

(iv) *$\mathcal{P}$ is exactly the family of minimal transversals of $\mathcal{D}$;*

(v) (a) *for all $P \in \mathcal{P}$ and $D \in \mathcal{D}$, we have $P \cap D \neq \emptyset$ and*

(b) *for every subset $B \subseteq \{x_1, x_2, \ldots, x_n\}$, there exists $D \in \mathcal{D}$ such that $D \subseteq B$ if and only if $P \cap B \neq \emptyset$ for every $P \in \mathcal{P}$.*

We obtain from Theorem 10.2(v) the following characterization of dual implicants.

Theorem 10.3. *A set of variables B is a dual implicant of the function f if and only if $P \cap B \neq \emptyset$ for all prime implicants P of f.*

10.2.2 The dual subimplicant theorem

We are now ready to present a characterization of the subimplicants of the dual of a positive function, due to Boros, Gurvich and Hammer [110]. This characterization is interesting on its own and also provides a useful tool for proving other results.

Let f be a positive Boolean function over the variables $V = \{x_1, x_2, \ldots, x_n\}$, and let f^d be its dual. As before, $\mathcal{P}$ and $\mathcal{D}$ denote the prime implicants of f and f^d, respectively. We assume throughout that all of the variables of f (and f^d) are essential.

A subset T of the variables is called a *dual subimplicant* of f if T is a subset of a dual prime implicant of f, i.e., if there exists a prime implicant D of f^d such that $T \subseteq D$. A *proper dual subimplicant* is a non-empty proper subset of a dual prime implicant.

Example 10.1. *Let $f = x_1x_2 \vee x_2x_3x_4 \vee x_4x_5$. Its dual is $f^d = x_1x_3x_5 \vee x_1x_4 \vee x_2x_4 \vee x_2x_5$. The proper dual subimplicants of f are the pairs $\{x_1, x_3\}, \{x_3, x_5\}, \{x_1, x_5\}$ and the five singletons $\{x_i\}$, $i = 1, \ldots, 5$.* □

We will make use below of the following consequence of Theorem 10.3.

Remark 10.1. Let T be a subset of the variables $\{x_1, x_2, \ldots, x_n\}$. If T is a proper dual subimplicant of f, then there exists a prime implicant $P \in \mathcal{P}$ such that $P \cap T = \emptyset$. □

Let T be a subset of the variables. Our goal will be to determine whether T is contained in some $D \in \mathcal{D}$, i.e., whether T is a dual subimplicant. We define the following sets of prime implicants of f, with respect to the set T:

$$\mathcal{P}_0(T) = \{P \in \mathcal{P} | P \cap T = \emptyset\},$$

and, for all $x \in T$,

$$\mathcal{P}_x(T) = \{P \in \mathcal{P} | P \cap T = \{x\}\}.$$

Note that by Theorem 10.3, $\mathcal{P}_0(T)$ is empty if and only if T is a dual implicant, and by Remark 10.1, $\mathcal{P}_0(T)$ is nonempty when T is a proper dual subimplicant. The remaining prime implicants in $\mathcal{P}$, which contain two or more variables of T, will not be relevant for our analysis. (We may omit the parameter T from our notation when it is clear which subset is meant.)

A *selection* $\mathcal{S}(T)$, with respect to T, consists of one prime implicant $P_x \in \mathcal{P}_x(T)$ for every $x \in T$. A selection is called *covering* if there is a prime implicant $P_0 \in \mathcal{P}_0(T)$ such that $P_0 \subseteq \bigcup_{x \in T} P_x$. Otherwise, it is called *non-covering.* (See Example 10.2.)

We now present the characterization of the dual subimplicants of a positive Boolean function from [110].

Theorem 10.4. *Let f be a positive Boolean function over the variable set $\{x_1, x_2, \ldots, x_n\}$, and let T be a subset of the variables. Then T is a dual subimplicant of f if and only if there exists a non-covering selection with respect to T.*

Proof. Assume that T is a dual subimplicant of f, and let $D \in \mathcal{D}$ be a prime implicant of f^d for which $T \subseteq D$. For any variable $x \in T$ the subset $D \setminus \{x\}$ is a proper subset of D, and therefore, by Remark 10.1 (or trivially, if $D = \{x\}$), there exists a prime implicant $P_x \in \mathcal{P}$ such that $P_x \cap (D \setminus \{x\}) = \emptyset$. Since $P_x \cap D \neq \emptyset$ by Theorem 10.3, we have $\{x\} = P_x \cap D = P_x \cap T$, that is, $P_x \in \mathcal{P}_x(T)$.

If $\mathcal{S} = \{P_x | x \in T\}$ were a covering selection, then there would exist a prime implicant $P_0 \in \mathcal{P}_0(T)$ such that $P_0 \subseteq \bigcup_{x \in T} P_x$. But this would imply

$$P_0 \cap D \subseteq (\bigcup_{x\in T} P_x) \cap D = \bigcup_{x\in T} (P_x \cap D) = T$$

which together with $P_0 \cap T = \emptyset$ would give $P_0 \cap D = \emptyset$, contradicting Theorem 10.3. Thus, the selection $\mathcal{S}$ we have constructed is a non-covering selection with respect to T. (Note that in the special case when $T = D$, we would have $\mathcal{P}_0(T)$ empty, and any selection would be non-covering.)

Conversely, suppose there exists a non-covering selection $\mathcal{S} = \{P_x | x \in T\}$ where $P_x \in \mathcal{P}_x(T)$. Since $\mathcal{S}$ is non-covering, we have for all $P_0 \in \mathcal{P}_0(T)$ that

$$P_0 \not\subseteq \bigcup_{x\in T} P_x.$$

Let B be defined as the complementary set

$$B = \left(\{x_1, x_2, \dots, x_n\} \setminus \bigcup_{x\in T} P_x\right) \cup T.$$

Clearly, for any prime implicant $P_0 \in \mathcal{P}_0(T)$, we have $P_0 \cap B \neq \emptyset$, since $\mathcal{S}$ is non-covering. Moreover, by definition, all other prime implicants $P \in \mathcal{P} \setminus \mathcal{P}_0(T)$ intersect T and, therefore, they too intersect B, since $T \subseteq B$. Thus, we have shown that $P \cap B \neq \emptyset$ for all $P \in \mathcal{P}$, implying that B is a (not necessarily prime) dual implicant.

Let $D \in \mathcal{D}$ be a dual prime implicant such that $D \subseteq B$. From the definition of B, it follows that $P_x \cap B = \{x\}$ for all $x \in T$. But each P_x intersects D since P_x is a prime implicant and D is a dual prime implicant, which together with the fact that $D \subseteq B$ implies that $P_x \cap D = \{x\}$. Hence, $T \subseteq D$, proving that T is a dual subimplicant. □

We will often apply Theorem 10.4 in its contrapositive form or in its dual form, as follows.

Remark 10.2. A subset T is not a dual subimplicant of f if and only if every selection with respect to T is a covering selection. □

Remark 10.3. We may also apply Theorem 10.4 to subimplicants of f and dual selections, where the roles of $\mathcal{P}$ and $\mathcal{D}$ are reversed in the obvious manner. $\Box$

Example 10.2. *Consider the positive Boolean function*

$$f = adg \vee adh \vee bdg \vee bdh \vee eag \vee ebg \vee ecg \vee eh$$

whose co-occurrence graph is shown in Figure 10.3.

(i) *Let $T = \{b, c, h\}$. We have*

$$\mathcal{P}_0(T) = \{adg, eag\}, \quad \mathcal{P}_b(T) = \{bdg, ebg\}, \quad \mathcal{P}_c(T) = \{ecg\}, \quad \mathcal{P}_h(T) = \{adh, eh\}.$$

The selection $\mathcal{S} = \{bdg, ecg, eh\}$ is non-covering since $\{a, d, g\}, \{a, e, g\} \not\subseteq \{b, c, d, e, g, h\}$, hence by Theorem 10.4, T is a dual subimplicant.

(ii) *Now let $T' = \{a, b, g\}$. We have*

$$\mathcal{P}_0(T') = \{eh\}, \quad \mathcal{P}_a(T') = \{adh\}, \quad \mathcal{P}_b(T') = \{bdh\}, \quad \mathcal{P}_g(T') = \{ecg\}.$$

There is only one possible selection $\mathcal{S}' = \{adh, bdh, ecg\}$ and $\mathcal{S}'$ is a covering selection since $\{e, h\} \subseteq \{a, b, c, d, e, g, h\}$. Hence, by Remark 10.2, T' is not a dual subimplicant.

It can be verified that T is contained in the dual prime implicant $abch$, and that in order to extend T' to a dual implicant it would be necessary to add either e or h, however, neither $abeg$ nor $abgh$ are prime (since $abe, bgh \in \mathcal{D}$), see Exercise 5. $\Box$

The problem of recognizing whether a given subset T is a dual subimplicant of a positive function f given by its complete DNF was shown to be NP-complete by Boros, Gurvich and Hammer [110]. However, they point out that Theorem 10.4 can be applied in a straightforward manner to answer this recognition problem in $O(n|f|^{1+\min\{|T|,|\mathcal{P}_0(T)|\}})$ time, where $|f|$ denotes the number of literals in the complete DNF of f. This becomes feasible for very small and very large values of $|T|$, such as 2, 3, $n-2, n-1$. Specifically, by applying this for every pair $T = \{x_i, x_j\}$, $1 \leq i < j \leq n$, we obtain the following.

Theorem 10.5. *The co-occurrence graph $G(f^d)$ of the dual of a positive Boolean function f can be determined in polynomial time, when f is given by its complete DNF. The complexity of determining all the edges of $G(f^d)$ is at most $O(n^3|f|^3)$.*

Proof. Consider a given pair $T = \{x_i, x_j\}$. We observe the following:

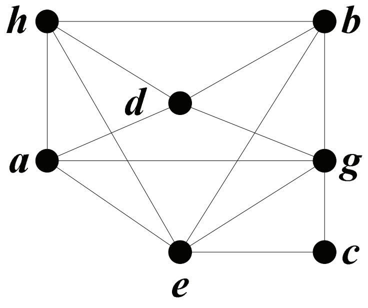

Figure 10.3: The co-occurrence graph for Example 10.2.

(1) If either $\mathcal{P}_{x_i}$ or $\mathcal{P}_{x_j}$ is empty, then there is no possible selection (covering or non-covering). Hence, Theorem 10.4 implies that x_i and x_j are not contained together in a dual prime implicant, and therefore, are not adjacent in $G(f^d)$.

(2) If both $\mathcal{P}_{x_i}$ and $\mathcal{P}_{x_j}$ are non-empty, but $\mathcal{P}_0$ is empty, then there is a selection and every selection will be non-covering. Hence, Theorem 10.4 implies that $\{x_i, x_j\}$ is a dual subimplicant, and so x_i and x_j are adjacent in $G(f^d)$.

(3) If all three sets $\mathcal{P}_0, \mathcal{P}_{x_i}$ and $\mathcal{P}_{x_j}$ are non-empty, then we may have to check all possible $O(|f|^2)$ selections before knowing whether there is a non-covering selection.

We leave a detailed complexity analysis as an exercise for the reader. □

Example 10.3. *Let us calculate $G(f^d)$ for the function $f = abc \vee bde \vee ceg$, as illustrated in Figure 10.4.*

The pair (a, b) is not an edge: indeed, we have in this case $\mathcal{P}_a = \emptyset$, so a and b are not adjacent in $G(f^d)$. Similarly, $(a, c), (b, d), (c, g), (d, e), (e, g)$ are also non-edges.

The pair (b, c) is an edge: in this case, both $\mathcal{P}_b$ and $\mathcal{P}_c$ are non-empty, but $\mathcal{P}_0$ is empty, so b and c are adjacent in $G(f^d)$. Similarly, $(b, e), (c, e)$ are also edges.

The pair (a, e) is an edge: in this case, as in the previous one, both $\mathcal{P}_a \neq \emptyset$ and $\mathcal{P}_e \neq \emptyset$, but $\mathcal{P}_0 = \emptyset$, so a and e are adjacent in $G(f^d)$. Similarly, $(b, g), (c, d)$ are also edges.

The pair (a, d) is an edge: in this case, $\mathcal{P}_a = \{abc\}$, $\mathcal{P}_d = \{bde\}$, $\mathcal{P}_0 = \{ceg\}$. Since $\{c, e, g\} \not\subseteq \{a, b, c, d, e\}$, we conclude that a and d are adjacent in $G(f^d)$. Similarly, $(a, g), (d, g)$ are also edges.

Notice what happens if we add an additional prime implicant bce to the function f in this example. Consider the function $f' = abc \vee bde \vee ceg \vee bce$. Then ad is not a dual

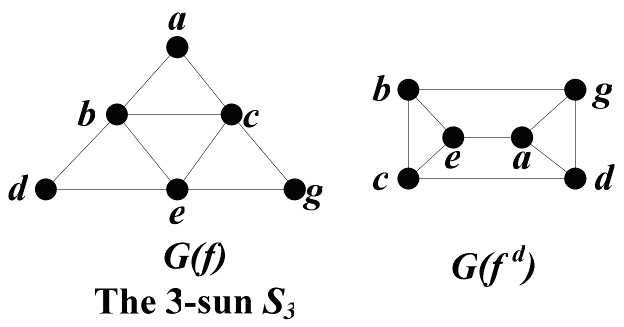

Figure 10.4: The co-occurrence graphs of f and f^d in Example 10.3.

subimplicant of f' although it was of f. Indeed, there is still only one selection $\{abc, bde\}$ but now it is covering, since it contains bce. By symmetry, neither ag nor dg are dual subimplicants of f'. □

10.3 Characterizing read-once functions

In this section, we present the mathematical theory underlying read-once functions due to Gurvich [393, 396, 397] and rediscovered by several other authors, see [271, 272, 514, 648]. The algorithmic aspects of recognizing and factoring read-once functions will be presented in Section 10.5.

Recall from Section 10.1 that a *read-once expression* is a Boolean expression in which every variable appears exactly once. A *read-once Boolean function* is a function that can be transformed (i.e., factored) into a read-once expression over the operations of conjunction and disjunction. We have also assumed read-once functions to be positive.

A positive Boolean expression, over the operations of conjunction and disjunction, may be represented as a (rooted) *parse tree* whose leaves are labeled by the variables $\{x_1, x_2, \ldots, x_n\}$, and whose internal nodes are labeled by the Boolean operations $\wedge$ and $\vee$. The parse tree represents the computation of the associated Boolean function according to the given expression, and each internal node is the root of a subtree corresponding to a part of the expression; see Figure 10.5. (A parse tree is a special type of combinational circuit, as introduced in Section 1.13.2.) If the expression is read-once, then each variable

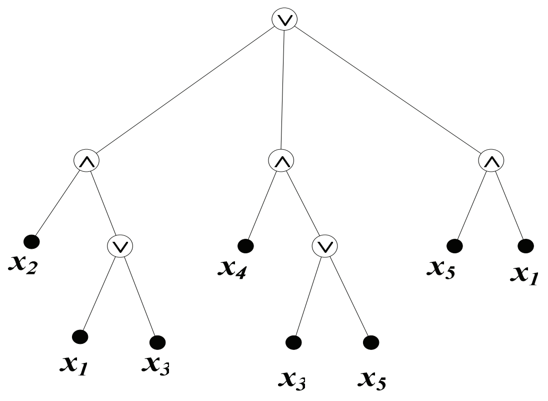

Figure 10.5: The parse tree of the expression $x_2(x_1 \vee x_3) \vee x_4(x_3 \vee x_5) \vee x_5x_1$.

appears on exactly one leaf of the tree, and there is a unique path from the root to the variable.

We begin by presenting a very useful lemma relating a read-once expression to the co-occurrence graph of the function. It also shows that *the read-once expression is unique for a read-once function* (Exercise 9).

Lemma 10.1. *Let T be the parse tree of a read-once expression for a positive Boolean function f over the variables $x_1, x_2, \ldots, x_n$. Then (x_i, x_j) is an edge in $G(f)$ if and only if the lowest common ancestor of x_i and x_j in the tree T is labeled $\wedge$ (conjunction).*

Proof. Since T is a tree, there is a unique path from the leaf labeled x_i to the root. Thus, for a pair $(x_i, x_j$ there is a unique lowest common ancestor v of x_i and x_j.

The lemma is trivial if there is only one variable. Let us assume that the lemma is true for all functions with fewer than n variables, and prove the result by induction. Let $u_1, \ldots, u_r$ be the children of the root of T, and for $k = 1, \ldots, r$, let T_k be the subexpression (subtree) rooted at u_k, denoting its corresponding function by f_k. Note that the variables at the leaves of T_k are disjoint from the leaves of T_l for $k \neq l$, since the expression is read-once.

If the root of T is labeled $\vee$, then $f = f_1 \vee \cdots \vee f_r$ and the graph $G(f)$ will be the disjoint union of the graphs $G(f_k)$ $(k = 1, \ldots, r)$, since multiplying out each of the expressions T_k will yield disjoint prime implicants of f. Thus, x_i and x_j are adjacent in $G(f)$ if and only if they are in the same T_k and adjacent in $G(f_k)$ and, by induction, if and only if their lowest common ancestor (in T_k and hence in T) is labeled $\wedge$ (conjunction).

If the root is labeled $\wedge$, then $f = f_1 \wedge \cdots \wedge f_r$ and the graph $G(f)$ will be the join of the graphs $G(f_k)$, $(k = 1, \ldots, r)$. That is, every vertex of the subgraph $G(f_k)$ is adjacent to every vertex of the subgraph $G(f_l)$ for $k \neq l$, since multiplying out each expression T_k and then expanding the entire expression T will put every pair of variables from different subtrees into some (perhaps many) prime implicants. Therefore, if x_i and x_j are on leaves of different subtrees, then they are connected in $G(f)$ and their lowest common ancestor is the root of T which is labeled $\wedge$. If x_i and x_j are on leaves of the same subtree, then again by induction, (x_i, x_j) is an edge in $G(f_k)$ if and only if the lowest common ancestor of x_i and x_j is labeled $\wedge$ (conjunction). □

We are now ready to present and prove the main characterization theorem of read-once functions. We describe briefly what will be shown in our Theorem 10.6 appearing below.

We have already seen in Theorem 10.2 that for any positive Boolean function f, every prime implicant P of f and every prime implicant D of its dual f^d must have at least one variable in common. This property is strengthened in the case of read-once functions, by condition (iv) in Theorem 10.6, which claims that f is read-once if and only if this common variable is unique. Moreover, this condition immediately implies (by definition) that the co-occurrence graphs $G(f)$ and $G(f^d)$ have no edges in common, for otherwise, a pair of variables adjacent in both graphs would be contained in some prime implicant and in some dual prime implicant. This is condition (iii) of our theorem, and already implies that *recognizing read-once functions has polynomial-time complexity* (by Theorem 10.5).

Condition (ii) is a further strengthening of condition (iii). It says that in addition to being edge-disjoint, the graphs are complementary, that is, every pair of variables either appear together in some prime implicant or together in some dual prime implicant but not both.

The remaining condition (v) characterizing read-once functions is the one mentioned as Theorem 10.1 at the beginning of this chapter, namely, that the co-occurrence graph $G(f)$ is P_4-free and the maximal cliques of $G(f)$ are precisely the prime implicants of f (normality). It is condition (v) that will be used in Section 10.5 to obtain an efficient $O(n|f|)$ recognition algorithm for read-once functions.

Example 10.4. *The function*

$$f_4 = x_1x_2 \vee x_2x_3 \vee x_3x_4 \vee x_4x_5 \vee x_5x_1$$

whose co-occurrence graph $G(f_4)$ is the chordless 5-cycle C_5, is normal but $G(f_4)$ is not P_4-free. Hence, f_4 is not a read-once function. Its dual

$$f_4^d = x_1x_2x_4 \vee x_2x_3x_5 \vee x_3x_4x_1 \vee x_4x_5x_2 \vee x_5x_1x_3,$$

whose co-occurrence graph $G(f_4^d)$ is the clique (complete graph) K_5 which is P_4-free, is not a normal function. □

Theorem 10.6. *Let f be a positive Boolean function over the variable set $\{x_1, x_2, \ldots, x_n\}$. Then the following conditions are equivalent:*
(i) *f is a read-once function;*
(ii) *the co-occurrence graphs $G(f)$ and $G(f^d)$ are complementary, i.e., $G(f^d) = \overline{G(f)}$;*
(iii) *the co-occurrence graphs $G(f)$ and $G(f^d)$ have no edges in common, i.e., $E(G(f)) \cap E(G(f^d)) = \emptyset$;*
(iv) *for all $P \in \mathcal{P}$ and $D \in \mathcal{D}$, we have $|P \cap D| = 1$;*
(v) *the co-occurrence graph $G(f)$ is P_4-free and f is normal.*

Proof. (i) $\Longrightarrow$ (ii): Assume that f is a read-once function, and let T be the parse tree of a read-once expression for f. By interchanging the operations $\vee$ and $\wedge$, we obtain the parse tree T^d of a read-once expression for the dual f^d. By Lemma 10.1, (x_i, x_j) is an edge in $G(f)$ if and only if the lowest common ancestor of x_i and x_j in the tree T is labeled $\wedge$ (conjunction). Similarly, (x_i, x_j) is an edge in $G(f^d)$ if and only if the lowest common ancestor of x_i and x_j in the tree T^d is labeled $\wedge$ (conjunction). It follows from the above construction that $G(f)$ and $G(f^d)$ are complementary.

(ii) $\Longrightarrow$ (iii): Trivial.

(iii) $\Longleftrightarrow$ (iv): As noted in the discussion above, by definition, the co-occurrence graphs $G(f)$ and $G(f^d)$ have no edges in common if and only if $|P \cap D| \leq 1$, for every prime implicant P of f and every prime implicant D of its dual f^d. However, for any positive Boolean function we have $|P \cap D| \geq 1$ by Theorem 10.2(v), which proves the equivalence.

(iv) $\Longrightarrow$ (v): We first prove that the function f is normal (Claim 1), and then that the graph $G(f)$ is P_4-free (Claim 3). We may assume both conditions (iii) and (iv) since we have already shown that they are equivalent.

Claim 1. *The function f is normal, that is, every clique of $G(f)$ is contained in a prime implicant of f.*

The claim is certainly true for any clique of size one, since we assume that all variables are essential, and it is true for any clique of size two, by the definition of the co-occurrence graph $G(f)$. Let us consider the smallest value k ($k \geq 3$) for which the claim fails, that is, there exists a clique $K = \{x_1, \ldots, x_k\}$ of $G(f)$ which is not a subimplicant of f. We denote the subcliques of K of size $k-1$ by $K_i = K - \{x_i\}$, $i = 1, \ldots, k$.

By our assumption of k being smallest possible, each set K_i is a subimplicant of f, so each is contained, respectively, in a prime implicant $P_i \in \mathcal{P}$ which we can express in the form

$$P_i = K_i \cup A_i$$

where $K \cap A_i = \emptyset$, since K is not a subimplicant.

In addition, each variable $x_i \in K$ is contained in a dual prime implicant $D_i \in \mathcal{D}$, which we can express in the form

$$D_i = \{x_i\} \cup B_i$$

where $K \cap B_i = \emptyset$, by our assumption (iv). Applying (iv) further, we note that

$$|P_i \cap D_j| = |(K_i \cup A_i) \cap (\{x_j\} \cup B_j)| = 1$$

for all i, j. In the case of $i \neq j$, since $x_j \in K_i$, this implies

$$A_i \cap B_j = \emptyset \quad (\forall i \neq j). \tag{10.1}$$

In the case of $i = j$, we obtain

$$|A_i \cap B_i| = 1$$

since the common variable cannot belong to K. This enables us to define

$$y_i = A_i \cap B_i \quad (i = 1, \ldots k). \tag{10.2}$$

Moreover, $y_i \neq y_j$ for $i \neq j$ by (10.1).

We now apply Theorem 10.4 (the dual subimplicant theorem). Consider a pair $T = \{x_i, x_j\}$ $(1 \leq i < j \leq k)$. Since (x_i, x_j) is an edge of $G(f)$, by assumption (iii), it is not an edge of $G(f^d)$ and hence not a dual subimplicant. By Theorem 10.4, this implies that every selection $\mathcal{S}$ with respect to T must be a covering selection.

Now, $\mathcal{S} = \{P_i, P_j\}$ is a selection for $T = \{x_i, x_j\}$ since $P_i \cap \{x_i, x_j\} = \{x_j\}$ and $P_j \cap \{x_i, x_j\} = \{x_i\}$. Therefore, there exists a prime implicant P_0 such that $P_0 \cap \{x_i, x_j\} = \emptyset$ and $P_0 \subseteq P_i \cup P_j$. Thus, $P_0 \subseteq (K \setminus \{x_i, x_j\}) \cup A_i \cup A_j$.

Since, $1 = |P_0 \cap D_i| = |P_0 \cap B_i|$, it follows from (10.2) that $y_i \in P_0$. Similarly, $1 = |P_0 \cap D_j| = |P_0 \cap B_j|$, so $y_j \in P_0$. Thus, (y_i, y_j) is an edge in $G(f)$. In fact, since i and j were chosen arbitrarily, the set $Y = \{y_1, \ldots, y_k\}$ is a clique in $G(f)$.

Now, we apply Theorem 10.4 to the dual function f^d, as suggested in Remark 10.3. Since the clique K is not a subimplicant of f, every dual selection $\mathcal{S}'$ with respect to K must be a covering dual selection. In particular, $\mathcal{S}' = \{D_1, \ldots, D_k\}$ is such a selection since $D_i = \{x_i\} \cup B_i$ intersects K only in x_i. Therefore, there exists a dual prime implicant D_0 satisfying $D_0 \cap K = \emptyset$ and $D_0 \subseteq \bigcup_{x_i \in K}(\{x_i\} \cup B_i)$, or

$$D_0 \subseteq \bigcup\nolimits_{x_i \in K} B_i. \tag{10.3}$$

For each i, we have $1 = |D_0 \cap P_i| = |D_0 \cap (K_i \cup A_i)|$. It therefore follows from (10.1), (10.2) and (10.3) that $D_0 \cap P_i = \{y_i\}$. Moreover, since i was chosen arbitrarily, $Y = \{y_1, \ldots, y_k\} \subseteq D_0$, implying that Y is a clique in $G(f^d)$. This is a contradition to (iii), since Y cannot be both a clique in $G(f)$ and a clique in $G(f^d)$. *This proves Claim 1.*

Claim 2. *If $(x_1, x_2), (x_2, x_3) \in E(G(f))$ and $(x_1, x_3) \notin E(G(f))$, then $(x_1, x_3) \in E(G(f^d))$.*

Suppose that (x_1, x_3) is not an edge of $G(f^d)$. Choose prime implicants

$$\{x_1, x_2\} \cup A_{12}, \{x_2, x_3\} \cup A_{23} \in \mathcal{P}$$

and dual prime implicants

$$\{x_1\} \cup B_1, \{x_3\} \cup B_3 \in \mathcal{D}.$$

By our assumptions,

$$\{x_1, x_2, x_3\} \cap (A_{12} \cup A_{23} \cup B_1 \cup B_3) = \emptyset.$$

By condition (iv), we have

$$|(\{x_1, x_2\} \cup A_{12}) \cap (\{x_1\} \cup B_1)| = 1 \Longrightarrow |A_{12} \cap B_1| = 0 \tag{10.4}$$

$$|(\{x_2, x_3\} \cup A_{23}) \cap (\{x_3\} \cup B_3)| = 1 \Longrightarrow |A_{23} \cap B_3| = 0 \tag{10.5}$$

and

$$|(\{x_1, x_2\} \cup A_{12}) \cap (\{x_3\} \cup B_3)| = 1 \Longrightarrow |A_{12} \cap B_3| = 1 \tag{10.6}$$

$$|(\{x_2, x_3\} \cup A_{23}) \cap (\{x_1\} \cup B_1)| = 1 \Longrightarrow |A_{23} \cap B_1| = 1. \tag{10.7}$$

From (10.6) and (10.7), we can define

$$y_1 = A_{12} \cap B_3$$

$$y_3 = A_{23} \cap B_1.$$

and from (10.4) and (10.5),

$$y_1 \neq y_3.$$

On the one hand, since we have assumed that $\{x_1, x_3\}$ is not a subimplicant of the dual f^d, by Theorem 10.4 we claim that every selection with respect to $\{x_1, x_3\}$ is covering. Now

$$\mathcal{S} = \{\{x_1, x_2\} \cup A_{12}, \{x_2, x_3\} \cup A_{23}\}$$

is such a selection, so there exists a prime implicant

$$P_0 \subseteq \{x_2\} \cup A_{12} \cup A_{23}.$$

By condition (iv), (10.4) and (10.5), we have

$$|P_0 \cap (\{x_1\} \cup B_1)| = 1 \Longrightarrow P_0 \cap (\{x_1\} \cup B_1) = y_3$$

and

$$|P_0 \cap (\{x_3\} \cup B_3)| = 1 \Longrightarrow P_0 \cap (\{x_3\} \cup B_3) = y_1.$$

Hence, $\{y_1, y_3\} \subseteq P_0$ and (y_1, y_3) is an edge of $G(f)$, that is,

$$(y_1, y_3) \in E(G(f)). \tag{10.8}$$

On the other hand, since we have also assumed that $\{x_1, x_3\}$ is not a subimplicant of the original function f, we again apply Theorem 10.4, this time in its dual form, by claiming that every dual selection with respect to $\{x_1, x_3\}$ is covering. Now,

$$\mathcal{S}' = \{\{x_1\} \cup B_1, \{x_3\} \cup B_3\}$$

is such a dual selection, so there exists a dual prime implicant

$$D_0 \subseteq B_1 \cup B_3.$$

By condition (iv), we have

$$|D_0 \cap (\{x_2, x_3\} \cup A_{23})| = 1 \Longrightarrow D_0 \cap (\{x_2, x_3\} \cup A_{23}) = y_3$$

and

$$|D_0 \cap (\{x_1, x_2\} \cup A_{12})| = 1 \Longrightarrow D_0 \cap (\{x_1, x_2\} \cup A_{12}) = y_1.$$

Hence, $\{y_1, y_3\} \subseteq D_0$ and (y_1, y_3) is an edge of $G(f^d)$, that is,

$$(y_1, y_3) \in E(G(f^d)). \tag{10.9}$$

Finally, combining the conclusions of (10.8) and (10.9), we have a contradiction, since $G(f)$ and $G(f^d)$ cannot share a common edge. *This proves Claim 2.*

Claim 3. *The graph $G(f)$ is P_4-free.*

Suppose $G(f)$ has a copy of P_4 with edges $(x_1, x_2), (x_2, x_3), (x_3, x_4)$ and nonedges $(x_2, x_4), (x_4, x_1), (x_1, x_3)$. By Claim 2, we have $(x_1, x_3), (x_2, x_4)$ are edges in $G(f^d)$. Choose prime implicants

$$\{x_1, x_2\} \cup A_{12},\ \{x_3, x_4\} \cup A_{34} \in \mathcal{P}$$

and dual prime implicants

$$\{x_1, x_3\} \cup B_{13},\ \{x_2, x_4\} \cup B_{24} \in \mathcal{D}.$$

By repeatedly using condition (iv), it is simple to verify that the sets

$$\{x_1, x_2, x_3, x_4\}, \quad A_{12} \cup A_{34}, \quad B_{13} \cup B_{24} \tag{10.10}$$

are pairwise disjoint.

Since $\{x_1, x_4\}$ is not a subimplicant of f, Theorem 10.4 implies that the dual selection

$$\mathcal{S}' = \{\{x_1, x_3\} \cup B_{13}, \{x_2, x_4\} \cup B_{24}\}$$

with respect to $\{x_1, x_4\}$ must be covering. So there exists a dual prime implicant $D_0 \in \mathcal{D}$ satisfying $D_0 \subseteq S'$ where

$$S' = (\{x_1, x_3\} \cup B_{13}) \cup (\{x_2, x_4\} \cup B_{24})$$

and $x_1, x_4 \notin D_0$. By the pairwise disjointness of the sets in (10.10), we have

$$S' \cap (\{x_1, x_2\} \cup A_{12}) = \{x_1, x_2\}$$

so

$$D_0 \cap (\{x_1, x_2\} \cup A_{12}) = \{x_2\}.$$

Hence, $x_2 \in D_0$.

In a similar manner, we can show that

$$D_0 \cap (\{x_3, x_4\} \cup A_{34}) = \{x_3\}.$$

Hence, $x_3 \in D_0$.

Thus, we have shown $x_2, x_3 \in D_0$, implying that (x_2, x_3) is an edge of $G(f^d)$, a contradiction to condition (iii). *This proves Claim 3.*

(v) $\Longrightarrow$ (i): Let us assume that f is normal and that $G = G(f)$ is P_4-free. We will show how to construct a read-once formula for f recursively. In order to prove this implication, we will use the following property of P_4-free graphs (cographs) which we will prove in Section 10.4, Theorem 10.7.

Claim 4. *If a graph G is P_4-free, then its complement $\overline{G}$ is also P_4-free; moreover, if G has more than one vertex, precisely one of G and $\overline{G}$ is connected.*

The function is trivially read-once if there is only one variable. Assume that the implication (v) $\Rightarrow$ (i) is true for all functions with fewer than n variables.

By Claim 4, one of G or $\overline{G}$ is disconnected. Suppose G is disconnected, with connected components $G_1, \ldots, G_r$ partitioning the variables of f into r disjoint sets. Then the prime implicants of f are similarly partitioned into r collections $\mathcal{P}_i$, $(i = 1, \ldots, r)$, defining positive functions $f_1, \ldots, f_r$, respectively, where $G_i = G(f_i)$ and $f = f_1 \vee \cdots \vee f_r$. Clearly, $G(f_i)$ is P_4-free since it is an induced subgraph of $G(f)$, and each f_i is normal for the same reason. Therefore, by induction, there is a read-once expression F_i for each i, and combining these, we obtain a read-once expression for f given by $F = F_1 \vee \cdots \vee F_r$.

Now suppose that $\overline{G}$ is disconnected, and let $H_1, \ldots, H_r$ be the connected components of $\overline{G}$, again partitioning the variables into r disjoint sets. Define $G_i = \overline{H_i}$. We observe that every vertex x_i of G_i is adjacent to every vertex x_j of G_j for $i \neq j$, so each maximal clique of $G(f)$ consists of a union of maximal cliques of $G_1, \ldots, G_r$. Moreover, since f is normal, the maximal cliques are precisely the prime implicants. It now follows that by restricting f to the variables of G_i, we obtain a normal function f_i whose co-occurrence graph $G(f_i) = G_i$ is P_4-free, and $f = f_1 \wedge \cdots \wedge f_r$. Therefore, by induction as before, there is a read-once expression F_i for each i, and combining these, we obtain a read-once expression for f given by $F = F_1 \wedge \cdots \wedge F_r$. □

Example 10.5. *Let us again consider the function*

$$f_0 = ay \vee cxy \vee bw \vee bz$$

whose co-occurrence graph $G(f_0)$ was shown in Figure 10.1. Clearly, f_0 is normal, and $G(f_0)$ is P_4-free and has two connected components $G_1 = G_{\{a,c,x,y\}}$ and $G_2 = G_{\{b,w,z\}}$. Using the arguments presented after Claim 4 above, we can handle these components separately, finding a read-once expression for each, and taking their disjunction.

For G_1, we note that its complement $\overline{G_1}$ is disconnected with two components, namely an isolated vertex $H_1 = \{y\}$ and $H_2 = \overline{G_1}_{\{a,c,x\}}$ having two edges; we can handle the components separately and take their conjunction. The complement $\overline{H_2}$ has an isolate $\{a\}$ and edge (c, x) which we combine with disjunction. Finally, complementing (c, x) gives two isolates which are combined with conjunction. Therefore, the read-once expression representing G_1 will be $y \wedge (a \vee [c \wedge x])$.

For G_2, we observe that its complement $\overline{G_2}$ has an isolate $\{b\}$ and edge (w, z) which we combine with conjunction, giving $b \wedge (w \vee z)$. So the read-once expression for f_0 is

$$f_0 = [y \wedge (a \vee [c \wedge x])] \vee [b \wedge (w \vee z)].$$

□

10.4 The properties of P_4-free graphs and cographs

The recursive construction of a read-once expression that we have just seen illustrated at the end of the last section in Example 10.5, was based on the special properties of P_4-free graphs, and in particular the use of Claim 4. We present these structural and algorithmic properties in this section.

The *complement reducible* graphs, or *cographs*, can be defined recursively as follows:

(1) a single vertex is a cograph,

(2) the union of disjoint cographs is a cograph,

(3) the join of disjoint cographs is a cograph,

where the *join* of disjoint graphs $G_1, \ldots, G_k$ is the graph G with $V(G) = V(G_1) \cup \cdots \cup V(G_k)$ and $E(G) = E(G_1) \cup \cdots \cup E(G_k) \cup \{(x, y) \mid x \in V(G_i), y \in V(G_j), i \neq j\}$. An equivalent definition can be obtained by substituting for (3) the rule

(3′) the complement of a cograph is a cograph,

see Exercise 15.

The building of a cograph G from these rules can be represented by a rooted tree T which records its construction, where

(a) the leaves of T are labeled by the vertices of G,

(b) if G is formed from the disjoint cographs $G_1, \ldots, G_k$ ($k > 1$), then the root r of T has as its children the roots of the trees of $G_1, \ldots, G_k$; moreover,

(c) the root r is labeled 0 if G is formed by the *union* rule (2), and labeled 1 if G is formed by the *join* rule (3).

Among all such constructions, there is a canonical one whose tree T is called the *cotree* and satisfies the additional property that

(d) on every path, the labels of the internal nodes alternate between 0 and 1.

Thus, the root of the cotree is labeled 1 if G is connected and labeled 0 if G is disconnected; an internal node is labeled 0 if its parent is labeled 1, and vice-versa. A subtree T_u rooted at an internal node u represents the subgraph of G induced by the labels of its leaves, and vertices x and y of G are adjacent in G if and only if their least common ancestor in the cotree is labeled 1.

Notice that the recursive application of rules (1)–(3) follows a *bottom-up* viewpoint of the construction of G. But an alternate *top-down* viewpoint can be taken, as a recursive decomposition of G, where we repeatedly partition the vertices according to either the connected components of G (union) or the connected components of its complement (join).

One can recognize whether a graph G is a cograph by repeatedly decomposing it this way, until either the decomposition fails on some component H (both H and $\overline{H}$ are connected) or it succeeds to reach all the vertices. The *cotree* is thus built top-down as the decomposition proceeds.[2]

The next theorem gives several characterizations of cographs.

Theorem 10.7. *The following are equivalent for an undirected graph G.*

(i) *G is a cograph.*

(ii) *G is P_4-free.*

(iii) *For every subset X of vertices ($|X| > 1$), either the induced subgraph G_X is disconnected or its complement $\overline{G}_X$ is disconnected.*

In particular, any graph G for which both G and $\overline{G}$ are connected, must contain an induced P_4. This claim appears in Seinsche [757]; independently, it was one of the problems on the 1971 Russian Mathematics Olympiad and seven students gave correct proofs, see [342]. The full version of the theorem was given independently by Gurvich [393, 394, 396] and by Corneil, Lerchs and Burlingham [195] where further results on the theory of cographs were developed. Note that it is impossible for both a graph G and its complement $\overline{G}$ to be disconnected, see Exercise 7.

It is rather straightforward to recognize cographs and build their cotree in $O(n^3)$ time. The first linear $O(n + e)$ time algorithm for recognizing cographs appears in Corneil, Perl and Stewart [196]. Subsequently, other linear time algorithms have appeared in [140, 402, **?**]; a fully dynamic algorithm is given in [766] and a parallel algorithm was proposed in [233].

Proof of Theorem 10.7. (iii) $\Longrightarrow$ (i): This implication follows immediately from the top-down construction of the cotree, as discussed above.

(i) $\Longrightarrow$ (ii): Let T be the cotree of G, and for vertex $x \in V(G)$, let p_x denote the path in T from the leaf labeled x to the root of the tree.

[2]This latter viewpoint is a particular case of modular decomposition [334] that applies to arbitrary graphs, and any modular decomposition algorithm will produce a cotree when given a cograph, although such general algorithms [401, 593] are more involved than is necessary for cograph recognition.

Suppose that G contains an induced P_4 with edges $(a, b), (b, c), (c, d)$. Since c and d are adjacent in G, their least common ancestor in T is an internal node u labeled 1. Consider the path p_a. Since both p_c and p_d must meet p_a in an internal node labeled by a 0, it follows that (i) they meet p_a in the same internal node, say v, and (ii) v is an ancestor of u.

Let us consider p_b. Now, p_b meets p_a in an internal node z labeled 1. If z is above v, then the least common ancestor of b and d will be z which is labeled 1, contradicting the fact that b and d are non-adjacent in G. Furthermore, $z \neq v$ since they have opposite labels, which implies that z must lie below v on p_a. However, in this case, the least common ancestor of b and c will be v which is labeled 0, contradicting the fact that b and c are adjacent in G. This proves the implication.

(ii) $\Longrightarrow$ (iii): Assume that G is P_4-free, thus $\overline{G}$ is also P_4-free, since P_4 is self-complementary. Suppose that there is an induced subgraph H of G such that both H and its complement $\overline{H}$ are connected. Clearly, they are also P_4-free, and can contain neither an isolated vertex nor a universal vertex (one that is adjacent to all other vertices).

We will construct an ordering $a_1, a_2, \ldots, a_n$ of $V(H)$ such that, for *odd*-indexed vertices a_{2j-1}:

$$(a_i, a_{2j-1}) \in E(H), \text{for all } i < 2j - 1$$

and, for *even*-indexed vertices a_{2j}:

$$(a_i, a_{2j}) \in E(\overline{H}), \text{for all } i < 2j.$$

In this case, a_n will either be an isolated vertex if n is even, or a universal vertex if n is odd, a contradiction.

Choose a_1 arbitrarily. Since a_1 cannot be universal in H, there is a vertex a_2 such that $(a_1, a_2) \in E(\overline{H})$. Since H is connected, there is a path in H from a_1 to a_2. Consider the shortest such path. It consists of exactly two edges of H, say $(a_1, a_3), (a_2, a_3) \in E(H)$, since H is P_4-free.

By a complementary argument, since $\overline{H}$ is connected and P_4-free, there is a shortest path in $\overline{H}$ from a_2 to a_3 consisting of exactly two edges of $\overline{H}$, say $(a_2, a_4), (a_3, a_4) \in E(\overline{H})$. Now we argue that $(a_1, a_4) \in E(\overline{H})$ since otherwise, H would have a P_4.

We continue constructing the ordering in the same manner. Assume we have $a_1, a_2, \ldots, a_{2j}$; we will find the next vertices in the ordering.

(Find a_{2j+1}). There is a shortest path in H from a_{2j-1} to a_{2j} consisting of exactly two edges of H, say $(a_{2j-1}, a_{2j+1}), (a_{2j}, a_{2j+1}) \in E(H)$. Note that a_{2j+1} has not yet been seen in the ordering, since none of the a_i is adjacent to a_{2j}. We argue, for all $i < 2j - 1$ that $(a_i, a_{2j+1}) \in E(H)$ since otherwise, H would have a P_4 on the vertices $\{a_i, a_{2j-1}, a_{2j+1}, a_{2j}\}$. Thus, we have enlarged our ordering by one new vertex.

(Find a_{2j+2}). There is a shortest path in $\overline{H}$ from a_{2j} to a_{2j+1} consisting of exactly two edges of $\overline{H}$, say $(a_{2j}, a_{2j+2}), (a_{2j+1}, a_{2j+2}) \in E(\overline{H})$. Now we argue, for all $i < 2j$ that $(a_i, a_{2j+2}) \in E(\overline{H})$ since otherwise, $\overline{H}$ would have a P_4 on the vertices $\{a_i, a_{2j}, a_{2j+2}, a_{2j+1}\}$. Thus, we have enlarged our ordering by another new vertex.

Eventually, this process orders all vertices, and the last one a_n will be either isolated or universal, giving the promised contradiction. □

10.5 Recognizing read-once functions

Given a Boolean function f, can we efficiently determine whether f is a read-once function? This is known as the recognition problem for read-once functions, which we define as follows.

Read-Once Recognition
Input: A representation of a positive Boolean function f by its list of prime implicants, i.e., its complete disjunctive normal form (DNF) expression.
Output: A read-once expression for f, or "failure" if there is none.

Chein [173] and Hayes [445] first introduced read-once functions and provided an exponential-time recognition algorithm for the family. Peer and Pinter [682] also gave an exponential-time factoring algorithm for read-once functions, whose non-polynomial complexity is due to the need for repeated calls to a routine that converts a DNF representation to a CNF representation, or vice-versa. We have already observed in Section 10.3 that combining Theorem 10.5 with condition (iii) of Theorem 10.6 implies that *recognizing read-once functions has polynomial-time complexity*, although without immediately providing the read-once expression.

In this section, we present the polynomial-time recognition algorithm due to Golumbic, Mintz and Rotics [373, 374, 375] and analyze its computational complexity. The algorithm is described in Figure 10.6. It is based on condition (v) of Theorem 10.6, that a function is read-once if and only if its co-occurrence graph is P_4-free (i.e., a cograph) and the function is normal. That is, we first test whether $G(f)$ is P_4-free and construct its cotree T, then we test whether f is normal. Passing both tests assures that f is read-once. Moreover, T will provide us with the read-once expression, see Remark 10.4 below.

Remark 10.4. The reader has no doubt noticed that the cotree of a P_4-free graph is very similar to the parse tree of a read-once expression. In fact, when a function is read-once,

```
Algorithm GMR Read-Once Recognition($f$)
Step 1: Build the co-occurrence graph $G(f)$.
Step 2: Test whether $G(f)$ is $P_4$-free. If so, construct the cotree $T$ for $G(f)$. Otherwise, exit with “failure”.
Step 3: Test whether $f$ is a normal function, and if so, output $T$ as the read-once expression. Otherwise, exit with “failure”.
```

Figure 10.6: Algorithm GMR Read-Once Recognition

its parse tree is identical to the cotree of its co-occurrence graph: just switch the labels $\{0, 1\}$ to $\{\vee, \wedge\}$. On the other hand, by the same token, a cotree always generates a read-once expression which represents “some” Boolean function g. Thus, the question to be asked is:

Given a function f, although $G(f)$ may be P_4-free and thus has a cotree T, will the read-once function g represented by T be equal to f, or not? (In other words, $G(g) = G(f)$ and, by construction, the maximal cliques of $G(g)$ are precisely the prime implicants of g, so will these also be the prime implicants of f?)

The function $f = ab \vee bc \vee ac$ is a negative example; its graph is a triangle and $g = abc$.

The answer to our question lies in testing normality, i.e., comparing the prime implicants of g with those of f, and doing it efficiently. □

The main result of this section is the following.

Theorem 10.8. *[373, 374, 375] Given the complete DNF formula of a positive Boolean function f on n variables, the* GMR *algorithm solves the* Read-Once Recognition *problem in time $O(n|f|)$, where $|f|$ denotes the length of the DNF expression.*

Proof. (Step 1.) The first step of the GMR algorithm is building the graph $G(f)$. If an arbitrary positive function f is given by its DNF expression, that is, as a list of its prime implicants $\mathcal{P} = \{P_1, \ldots, P_m\}$, then the edge set of $G(f)$ can be found in $O(\sum_{i=1}^{m} |P_i|^2)$ time. It is easy to see that this is at most $O(n|f|)$.

(Step 2.) As we have seen in Section 10.4, the complexity of testing whether the graph $G(f)$ is P_4-free and providing a read-once expression (its cotree T) is $O(n + e)$, as first shown in [196]. This is at worst $O(n^2)$ and is bounded by $O(n|f|)$. (A straightforward application of Theorem 10.7 would yield complexity $O(n^3)$).

(Step 3.) Finally, we show that the function f can be tested for normality in $O(n|f|)$ time by a novel method due to [373] and described more fully in [374, 375, 637][3]. As in Remark 10.4, we will denote by g the function represented by the cotree T; we will verify that $g = f$.

Testing normality.

We may assume that $G = G(f)$ has successfully been tested to be P_4-free, and that T is its cotree. We construct the set of maximal cliques of G recursively, by traversing the cotree T from bottom to top, according to Lemma 10.2 below. For a node x of T, we denote by T_x the subtree of T rooted at x, and we denote by g_x the function represented by T_x. We note that T_x is also the cotree representing the subgraph G_X of G induced by the set X of labels of the leaves of T_x.

First we introduce some notation. Let $X_1, X_2, \dots, X_r$ be disjoint sets, and let $\mathcal{C}_i$ be a set of subsets of X_i $(1 \le i \le r)$. We define the *Cartesian sum* $\mathcal{C} = \mathcal{C}_1 \otimes \cdots \otimes \mathcal{C}_r$, to be the set whose elements are unions of individual elements from the sets $\mathcal{C}_i$ (one element from each set). In other words,

$$\mathcal{C} = \mathcal{C}_1 \otimes \cdots \otimes \mathcal{C}_r = \{C_1 \cup \cdots \cup C_r \mid C_i \in \mathcal{C}_i, 1 \le i \le r\}.$$

For a cotree T, let $\mathcal{C}(T)$ denote the set of all maximal cliques in the cograph corresponding to T. From the definitions of cotree and cograph, we obtain:

Lemma 10.2. *Let G be a P_4-free graph and let T be the cotree of G. Let h be an internal node of T and let $h_1, \dots, h_r$ be the children of h in T.*

(1) *If h is labeled with 0, then $\mathcal{C}(T_h) = \mathcal{C}(T_{h_1}) \cup \cdots \cup \mathcal{C}(T_{h_r})$.*

(2) *If h is labeled with 1, then $\mathcal{C}(T_h) = \mathcal{C}(T_{h_1}) \otimes \cdots \otimes \mathcal{C}(T_{h_r})$.*

The following algorithm calculates, for each node x of the cotree, the set $\mathcal{C}(T_x)$ of all the maximal cliques in the cograph defined by T_x. It proceeds bottom up, using Lemma 10.2, and also keeps at each node x:

$s(T_x)$: The number of cliques in $\mathcal{C}(T_x)$. This number is equal to the number of prime implicants in g_x.

[3] In [374] only a complexity bound of $O(n^2k)$ was claimed, where k is the number of prime implicants; however, using an efficient data structure and careful analysis, it has been shown in [375], following [637], that the method can be implemented in $O(n|f|)$. For the general case of a positive Boolean function given in DNF form, it is possible to check normality in $O(n^3k)$ time using the results of [504]; see Exercise 13.

$L(T_x)$: The total length of the list of cliques at T_x, namely, $L(T_x) = \sum\{|C||C \in \mathcal{C}(T_x)\}$, which represents the total length of the list of prime implicants of g_x.

A global variable L maintains the overall size of the clique lists as they are being built. (In other words, L is the sum of all $L(T_x)$ taken over all x on the frontier as we proceed bottom up.)

The steps of the normality checking algorithm are given in Figure 10.7. This algorithm correctly tests normality, since it tests whether the maximal cliques of the cograph are precisely the prime implicants of f.

Complexity analysis.

The purpose of comparing $s(T_h)$ with k at each step is simply a speed-up mechanism to assure that the number of cliques never exceeds the number of prime implicants. Similarly, calculating $L(T_h)$, i.e., $|g_h|$ and comparing L with $|f|$ at each step assures that the overall length of the list of cliques will never exceed the sum of the lengths of the prime implicants. (Note that we pre-compute L, and test against $|f|$ *before* we actually build a new set of cliques.)

For efficiency, we number the variables $\{x_1, x_2, \ldots, x_n\}$, and maintain both the prime implicants and the cliques as lists of their variables. Then, each collection of cliques $\mathcal{C}(T_x)$ is maintained as a list of such lists. In this way, constructing $\mathcal{C}(T_h)$ in Step 3b(1) can be done by concatenating the lists $\mathcal{C}(T_{h_1}), \ldots, \mathcal{C}(T_{h_r})$; and constructing $\mathcal{C}(T_h)$ in Step 3b(2) can be done by creating a new list of cliques by repeatedly taking r (sub)cliques, one from each set $\mathcal{C}(T_{h_1}), \ldots, \mathcal{C}(T_{h_r})$ and concatenating these r (disjoint) lists of variables.

Thus, the overall calculation of $C(T_h)$ takes at most $O(|f|)$ time. Since the number of internal nodes of the cotree is less than n, the complexity of Steps 3a and 3b is $O(n|f|)$.

It remains to compare the list of the prime implicants of f with the list of the maximal cliques $\mathcal{C}(T_y)$, where y is the root of T. This can be accomplished using radix sort in $O(nk)$ time. Initialize two $k \times n$ bit matrices $\mathbf{P}$ and $\mathbf{C}$ filled with zeros. Each prime implicant P_i is traversed (it is a list of variables) and for every $x_j \in P_i$ we assign $\mathbf{P}_{i,j} \leftarrow 1$, thus, converting it into its characteristic vector which will be in row i of $\mathbf{P}$. Similarly, we traverse each maximal clique C_i and convert it into its characteristic vector which will be in row i of $\mathbf{C}$. It is now a straightforward procedure to lexicographically sort the rows of these two matrices and compare them in $O(nk)$ time.

This concludes the proof, since the complexity of each step is bounded by $O(n|f|)$.

$\square$

Of course, as we are used by now, the form in which a function f is given influences the computational complexity of recognizing whether it is read-once. For example, if

Algorithm Checking Normality(f)

Step 3a: Initialize k to be the number of terms (clauses) in the DNF representation of f. For every leaf a of T, set $\mathcal{C}(T_a) = \{a\}$ and set $s(T_a) = 1$, $L(T_a) = 1$, and $L = n$.

Step 3b: Scan T from bottom to top, at each internal node h reached, let $h_1, \dots, h_r$ be the children of h and do:

(1) If h is labeled with 0:

- set $s(T_h) = s(T_{h_1}) + \cdots + s(T_{h_r})$
- if $s(T_h) > k$ stop, and claim that f is not normal; otherwise,
- set $L(T_h) = L(T_{h_1}) + \cdots + L(T_{h_r})$
- L remains unchanged
- set $\mathcal{C}(T_h) = \mathcal{C}(T_{h_1}) \cup \cdots \cup \mathcal{C}(T_{h_r})$

(2) If h is labeled with 1:

- set $s(T_h) = s(T_{h_1}) \times \cdots \times s(T_{h_r})$
- if $s(T_h) > k$ stop, and claim that f is not normal; otherwise,
- set $L(T_h) = \Sigma\{|C_1| + \cdots + |C_r| \mid (C_1, \dots, C_r) \in \mathcal{C}(T_{h_1}) \times \cdots \times \mathcal{C}(T_{h_r})\}$
- set $L \leftarrow L + L(T_h) - [L(T_{h_1}) + \cdots + L(T_{h_r})]$
- if $L > |f|$ stop, and claim that f is not normal; otherwise,
- set $\mathcal{C}(T_h) = \mathcal{C}(T_{h_1}) \otimes \cdots \otimes \mathcal{C}(T_{h_r})$

Step 3c: Let y be the root of T, and let $\mathcal{C}(T_y)$ be the set of maximal cliques of the cograph, obtained by the preceding step.

- If $s(T_y) \neq k$ or if $|L| \neq |f|$ stop, and claim that f is not normal.
- Otherwise, compare the set $\mathcal{C}(T_y)$ with the set of prime implicants (from the DNF) of f, using radix sort as described in the proof. If the sets are equal, claim that f is normal. Otherwise, claim that f is not normal.

Figure 10.7: The algorithm for checking normality.

f is initially represented by an arbitrary Boolean expression, we are required to pay a preprocessing expense to test that f is positive and to transform f into its DNF expression in order to apply the GMR algorithm. The same would be true if f were to be given as a BDD. This preprocessing could be exponential in the size of the original input.

Actually, for a general (non-monotone) DNF expression ψ, Theorem 1.30 (page 61) implies that it is NP-hard to decide whether ψ represents a read-once function. But the question remains open for BDDs and for positive expressions (other than DNFs).

Exercise 25 raises some related open questions regarding the complexity of recognizing a read-once function depending on the representation of the function. For example, we may be fortunate to receive f as a very compact expression, yet not know how to take advantage of this. When may it be possible to efficiently construct the co-occurrence graph of a Boolean function and test normality for forms other than a positive DNF representation?

10.6 Learning read-once functions

I've got a secret. It's a Boolean function f. Can you guess what it is? You can ask me questions, like "What is the value of f at the point X?" Can you figure out my mystery function with just 20 questions?

The answer, of course, is *yes*, 20 questions are enough if the number of variables is at most 4. Otherwise, the answer is *no.* If there are n variables, then there will be 2^n independent points to be queried, before you can be sure to "know" the function.

Suppose I give you a clue: The function f is a positive Boolean function. Now can you learn f with fewer queries?

Again the answer is *yes.* The extra information given by the clue, allows you to ask fewer questions in order to learn the function. For example, in the case $n = 4$, first try (1,1,0,0). If the answer is true, then you immediately know that (1,1,1,0), (1,1,0,1) and (1,1,1,1) are all true. If the answer is false, then (1,0,0,0), (0,1,0,0) and (0,0,0,0) are all false. Either way, you asked one question and got four answers. Not bad. Now if you query (0,0,1,1), you will similarly get two or three more free answers. In the worst case, it could take 10 queries to learn the function (rather than 16 had you queried each point).

Learning a Boolean function in this manner is sometimes called Exact Learning with Queries; see Angluin [19]. It receives as input an oracle for a Boolean function f, that is, a "black box" which can answer a query on the value of f at a given Boolean point in constant time. It then attempts to learn the value of f at all 2^n points and outputs a Boolean expression that is logically equivalent to f.

If we know something extra about the structure of the function f, then it may be possible to reduce the number of queries required to learn the function. We saw this above in our example with the clue (that the mystery function was positive). However, even for positive functions, the number of queries needed to learn the function remains exponential.

The situation is much better for read-once functions. In this case, the number of required queries can be reduced to a polynomial number, and the unique read-once formula can be produced, provided we "know" that the function is read-once. Thus, the read-once functions constitute a very natural class of functions that can be learned efficiently and, for this reason, they have been extensively studied within the computational learning theory community.

For our purposes, we define the problem as follows.

READ-ONCE EXACT LEARNING
Input: A black-box oracle to evaluate f at any given point, where f is known *a priori* to be a positive read-once function.
Output: A read-once factorization for f.

Remark 10.5. There is a subtle but significant difference between the EXACT LEARNING problem and the RECOGNITION problem. With recognition, we have a DNF expression for f and must determine if it represents a read-once function. With exact learning, we have an oracle for f whose correct usage relies upon the a priori assumption that the function to be learned is read-once. So the input assumptions are different, but the output goal in both cases is a correct read-once expression for f. Also, when measuring the complexity of recognition, we count the algorithmic operations; when measuring the complexity of exact learning, we must count both the operations implemented by the algorithm and the number of queries to the oracle. □

As we have already seen in Section 10.5, the GMR recognition algorithm: (1) uses the DNF expression to construct the co-occurrence graph $G(f)$, then (2) tests whether $G(f)$ is P_4-free and builds a cotree T for it, and (3) uses T and the original DNF formula to test whether f is normal; if so, T is the read-once expression.

In contrast to this, Angluin, Hellerstein and Karpinski [20] give the exact learning algorithm in Figure 10.8.

The main difference that concerns us, between AHK exact learning and GMR recognition, will be Step 1, that is, *how to construct $G(f)$ using an oracle?* We outline their

```
Algorithm AHK Read-Once Exact Learning($f$)
Step 0: Check whether $f$ is a constant function, using the oracle: if $f(\mathbf{1}) = 0$ then $f$
is constant 0; if $f(\mathbf{0}) = 1$ then $f$ is constant 1.
Step 1: Use the oracle to construct the co-occurrence graph $G(f)$.
Step 2: Build a cotree $T$ for $G(f)$ ("knowing" a priori that it must be $P_4$-free and
thus will succeed).
Step 3: Immediately output $T$ as the read-once expression ("knowing" a priori that
$f$ is normal).
```

Figure 10.8: Algorithm AHK Read-Once Exact Learning

solution through a series of exercises.

(A) In a greedy manner, we can determine whether a subset $U \subseteq X$ of the variables contains a prime implicant, and find one when the answer is positive. Exercise 16 gives such a routine Find-PI-In(U) which has complexity $O(n)$ plus $|U|$ queries to the oracle. A similar greedy algorithm Find-DualPI-In(U) will find a dual prime implicant contained in U.

(B) An algorithm Find-Essential-Variables is developed in Exercises 17, 18 and 19 that not only finds the set Y of essential variables[4] but in the process, for each variable x_i in Y, generates a prime implicant $P[i]$ and a dual prime implicant $D[i]$ containing x_i. This algorithm uses Find-PI-In and Find-DualPI-In and can be implemented to run in $O(n^2)$ time using $O(n^2)$ queries to the oracle.

(C) Finally, we construct the co-occurrence graph $G(f)$ based on the following Lemma (whose proof is proposed as Exercise 14):

Lemma 10.3. *Let f be a non-constant read-once function over the variables $N = \{x_1, x_2, \ldots, x_n\}$. Suppose that D_i is a dual prime implicant containing x_i but not x_j, and that D_j is a dual prime implicant containing x_j but not x_i. Let $R_{i,j} = (N \setminus (D_i \cup D_j)) \cup \{x_i, x_j\}$. Then (x_i, x_j) is an edge in the co-occurrence graph $G(f)$ if and only if $R_{i,j}$ contains a prime implicant.*

We obtain $G(f)$ using the oracle in the following way: For each pair of essential variables x_i and x_j,

C.1: if $x_i \in D[j]$ or $x_j \in D[i]$ then (x_i, x_j) is *not* an edge of $G(f)$.

[4]We have generally assumed throughout this chapter that all of the variables for a Boolean function f (and hence for f^d) are essential. However, in the exact learning problem, we may wish to drop this assumption and need to find the set of essential variables.

C.2: Otherwise, construct $R_{i,j}$ from $D[i]$ and $D[j]$ and test whether $R_{i,j}$ contains a prime implicant using just one query to the oracle, i.e., is $f(X^{R_{i,j}}) = 1$? If so, then (x_i, x_j) is an edge in $G(f)$, otherwise, it is not an edge.

Complexity.

The computational complexity of the algorithm is determined as follows. Step 0 requires two queries to the oracle. Step 1 constructs the co-occurrence graph $G(f)$ by first calling the algorithm FIND-ESSENTIAL-VARIABLES (Part B) to generate $P[i]$ and $D[i]$ for each variable x_i in $O(n^2)$ time using $O(n^2)$ queries, then it applies Lemma 10.3 (Part C) to determine the edges of the graph. Step C.1 can be done in the same complexity as Step B, however, Step C.2 uses $O(n^3)$ time and $O(n^2)$ queries, since, for each pair i, j, we have $O(n)$ operations and 1 query. Step 2, building the cotree T for $G(f)$ takes $O(n^2)$ time using one of the fast cograph algorithms of [140, 196, 402], and Step 3 takes no time at all.

Summarizing the above, the overall complexity using the method of Angluin, Hellerstein and Karpinski [20] will be $O(n^3)$ time and $O(n^2)$ queries. However, Dahlhaus subsequently reported an alternative to Step C.2 in an unpublished manuscript [232] using only $O(n^2)$ time. (Further generalizations by Raghavan and Schach [720] lead to the same time bound.)

The main result, therefore, is the following.

Theorem 10.9. *The* READ-ONCE EXACT LEARNING *problem can be solved with the AHK algorithm in $O(n^2)$ time, using $O(n^2)$ queries to the oracle.*

Proof. The correctness of the AHK exact learning algorithm follows from Lemma 10.3, Exercises 17-19 and Remark 10.4. □

Remark 10.6. If a lying, deceitful, cunning adversary were to place a *non*-read-once function into our "black box" query oracle, then the exact learning method described here would give an incorrect identification answer, since the "a priori read-once" assumption is vital for the construction of $G(f)$. (See the discussion in Exercise 27 concerning what might happen if such an oracle were to be applied to a non-read-once function.) □

Further topics relating computational learning theory with read-once functions may be found in [13, 20, 147, 450, 369, 370, 448, 696, 720, 776, 819, etc.].

10.7 Related topics and applications of read-once functions

In this section, we briefly mention three topics related to read-once functions and application areas in which they play an interesting role.

10.7.1 The readability of a Boolean function

Suppose a given function f is not a read-once function. In this case, we may still want to obtain an expression which is logically equivalent to f and which has a small number of repetitions of the variables. In [374], Golumbic, Mintz and Rotics introduced the notion of the readability of a Boolean function to capture this notion.

We call a Boolean expression *read-m* if each variable appears at most m times in the expression. A Boolean function f is defined to be a *read-m function* if it has an equivalent read-m expression. Finally, the *readability* of f is the smallest number m such that f is a read-m function.

In general determining the readability of a function may be quite difficult, and to the best of our knowledge, it is not known whether there is a polynomial time algorithm which given a function f in an irredundant DNF or CNF representation decides whether f is a read-m function or not, for a fixed $m \geq 2$, even in the case of $m = 2$.

It was therefore proposed in [374] to investigate restrictions of the general problem to special cases of positive Boolean functions f identified by the structure of the co-occurrence graph $G(f)$. As a first step in this direction, they have shown the following result.

Theorem 10.10. *[374] Let f be a positive Boolean function. If f is a normal function and its co-occurrence graph $G(f)$ is a partial k-tree, then f is a read-2^k function and a read-2^k expression for f can be obtained in polynomial $(O(n^{k+1}))$ time.*

Notice that if $G(f)$ is a tree, then f would immediately be normal. Therefore, in the case of $k = 1$, the Theorem 10.10 reduces to the following.

Corollary 10.1. *Let f be a positive Boolean function. If $G(f)$ is a tree, then f is a read-twice function.*

The definition of readability does not require the function to be positive. Thus, characterizing read-twice Boolean functions and characterizing positive read-twice Boolean functions, appear to be separate open questions.

10.7.2 Factoring general Boolean functions

Factoring is the process of deriving a parenthesized Boolean expression or *factored form* representing a given Boolean function. Since, in general, a function will have many factored forms, the problem of factoring Boolean functions into shorter, more compact logically equivalent expressions is one of the basic operations in the early stages in designing logic circuits. Generating an optimum factored form (a shortest length expression) is an NP-hard problem. Thus, heuristic algorithms have been developed in order to obtain good factored forms.

An exception to this, as we have already seen, are the read-once functions. For a read-once function f, the read-once expression is unique, it can be determined very efficiently, and, moreover, it is the shortest possible expression for f. According to [682], read-once functions account for a significant percentage of functions that arise in real circuit applications. Some smaller or specifically designed circuits may indeed be read-once functions, but most often they will not even be positive functions. Nevertheless, we can use the optimality of factoring read-once functions as part of a heuristic method.

Such an approach for factoring general Boolean functions has been described in [372, 638], and is based on graph partitioning. Their heuristic algorithm is recursive and operates on the function and on its dual, to obtain the better factored expression. As a special class, which appears in the lower levels of the recursive factoring process, are the read-once functions.

The original function f is decomposed into smaller components, for example, $f = f_1 \vee f_2 \vee f_3$, and when a component is recognized to be read-once, a special purpose subroutine (namely, the GMR algorithm of Section 10.5) is called to factor that read-once component efficiently and optimally. Their method has been implemented in the SIS logic synthesis environment, and an empirical evaluation indicates that the factored expressions obtained are usually significantly better than those from previous fast algebraic factoring algorithms, and are quite competitive with previous Boolean factoring methods but with lower computation costs (see [637, 638]).

10.7.3 Positional games

We introduce here the notions of normal, extensive and positional game forms, and then show their relationship with read-once functions.

Definition 10.1. *Given three finite sets* $S^1 = \{s^1_1, s^1_2, ..., s^1_{m_1}\}$, $S^2 = \{s^2_1, s^2_2, ..., s^2_{m_2}\}$, *which are interpreted as the sets of strategies of the players* 1 *and* 2, *and* $X = \{x_1, x_2, ..., x_k\}$, *which is interpreted as the set of outcomes, a* game form *(of two players) is a mapping*

$g : S^1 \times S^2 \to X$, which assigns an outcome $x(s^1, s^2) \in X$ to every pair of strategies $s^1 \in S^1$, $s^2 \in S^2$.

A convenient representation of a game form is a matrix $M = M(g)$ whose rows are labeled by S^1, whose columns are labeled by S^2 and whose elements are labeled by X. For example,

$$M_1 = \begin{bmatrix} x_1 & x_2 \\ x_2 & x_1 \end{bmatrix}.$$

Each outcome $x \in X$ may appear several times in $M(g)$, because g may not be injective. We can interpret $M(g)$ as a game in normal form "but without payoff, which is not given yet."

Definition 10.2. *Two strategies s_1^i and s_2^i of player i, where $i = 1$ or 2, are called* equivalent *if for every strategy s^{3-i} of the opponent, we have $g(s_1^i, s^{3-i}) = g(s_2^i, s^{3-i})$; in other words, if in matrix $M(g)$ the rows $(i = 1)$ or the columns $(i = 2)$ corresponding to the strategies s_1^i and s_2^i, are equal.*

We will restrict ourselves by studying the game forms *without equivalent strategies.*

Definition 10.3. *Given a read-once function f, we can interpret its parse tree (or read-once formula) $T(f)$ as an* extensive game form (or game tree) of two players. *The leaves $X = \{x_1, x_2, ..., x_k\}$ of T are the* final positions or outcomes. *The internal vertices of T are the* internal positions. *The game starts at the root of T and ends in a final position $x \in X$. Each path from the root to a final position (leaf) is called a* play. *If an internal node v is labeled by $\vee$ (respectively, by $\wedge$), then it is the turn of player 1 (respectively, player 2) to move in v. This player can choose any vertex which is a child of v in T.*

A strategy *of a player is a mapping which assigns a move to every position in which this player has to move. In other words, a strategy is a plan of how to play in every possible situation.*

Any pair of strategies s^1 of player 1 and s^2 of player 2 define a play $p(s^1, s^2)$ and an outcome $x(s^1, s^2)$ which would appear if both players implement these strategies.

Two strategies s_1^i and s_2^i of player i, where $i = 1$ or 2, are called equivalent *if for every strategy s^{3-i} of the opponent the outcome is the same, that is, if $x(s_1^i, s^{3-i}) = x(s_2^i, s^{3-i})$. By suppressing all but one (arbitrary) strategy from every class of equivalent strategies, we obtain two reduced sets of strategies denoted by $S^1 = \{s_1^1, s_2^1, ..., s_{m_1}^1\}$ and $S^2 = \{s_1^2, s_2^2, ..., s_{m_2}^2\}$.*

The mapping $g : S^1 \times S^2 \to X$, *which assigns the outcome* $x(s^1, s^2) \in X$ *to every pair of strategies* $s^1 \in S^1$, $s^2 \in S^2$ *defines a game form, which we call the* normal form *of the corresponding extensive game form.*

Note that such a mapping $g = g(T)$ *may be not injective, because different pairs of strategies may generate the same play.*

We call a game form g positional *if it is the normal form of an extensive game form, that is, if* $g = g(T(f))$ *for a read-once function* f.

Example 10.6. *In the extensive game form defined by the read-once formula* $((x_1 \vee x_2)x_3 \vee x_4)x_5$, *each player has three strategies, and the corresponding normal game form is given by the following* (3×3)*-matrix:*

$$M_2 = \begin{bmatrix} x_1 & x_3 & x_5 \\ x_2 & x_3 & x_5 \\ x_4 & x_4 & x_5 \end{bmatrix}.$$

The game form given by the matrix

$$M_3 = \begin{bmatrix} x_1 & x_1 \\ x_2 & x_3 \end{bmatrix}$$

is also generated by a read-once formula, namely, by $x_1 \vee x_2x_3$. □

Our aim is to characterize the positional game forms.

Definition 10.4. *Let us consider a game form* g *and the corresponding matrix* $M = M(g)$. *We associate with* M *two DNFs, representing two Boolean functions* $f_1 = f_1(g) = f_1(M)$ *and* $f_2 = f_2(g) = f_2(M)$, *respectively, by first taking the conjunction of all the variables in each row (respectively, each column) of* M, *and then taking the disjunction of all these conjunctions for all rows (respectively, columns) of* M.

We call a game form g *(as well as its matrix* M*)* tight *if the functions* f_1 *and* f_2 *are mutually dual.*

Example 10.7. *Matrix* M_2 *of Example 10.6 generates the functions* $f_1(M_2) = x_1x_3x_5 \vee x_2x_3x_5 \vee x_4x_5$ *and* $f_2(M_2) = x_1x_2x_4 \vee x_3x_4 \vee x_5$. *These functions are mutually dual, thus the game form is tight. Matrix* M_3 *is also tight, because its functions* $f_1(M_3) = x_1 \vee x_2x_3$ *and* $f_2(M_3) = x_1x_2 \vee x_1x_3$ *are mutually dual. However,* M_1 *is not tight, because its functions* $f_1(M_1) = f_2(M_1) = x_1x_2$ *are not mutually dual.* □

Remark 10.7. It is proven in [392] that a normal game form (of two players) is Nash-solvable (i.e., for an arbitrary payoff the obtained game has at least one Nash equilibrium in pure strategies) if and only if this game form is tight. □

Theorem 10.11. *Let f be a read-once function, $T = T(f)$ the parse tree of f interpreted as an extensive game form, $g = g(T)$ its normal form, $M = M(g)$ the corresponding matrix, and $f_1 = f_1(M)$, $f_2 = f_2(M)$ the functions generated by M. Then $f_1 = f$ and $f_2 = f^d$.*

Proof. By induction. For a trivial function f the claim is obvious. If $f = f' \vee f''$ then $f_1 = f_1' \vee f_1''$ and $f_2 = f_2' \wedge f_2''$. If $f = f' \wedge f''$ then $f_1 = f_1' \wedge f_1''$ and $f_2 = f_2' \vee f_2''$. The theorem follows directly from the definition of strategies. □

Definition 10.5. *We call a game form $g : S^1 \times S^2 \rightarrow X$ (as well as the corresponding matrix M)* rectangular *if every outcome $x \in X$ occupies a rectangular array in M, that is, if the following property holds: $g(s_1^1, s_1^2) = g(s_2^1, s_2^2) = x$ implies $g(s_1^1, s_2^2) = g(s_2^1, s_1^2) = x$.*

For example, matrices M_2 and M_3 above are rectangular, while M_1 is not.

Theorem 10.12. *A game form g and its corresponding matrix M are rectangular if and only if every prime implicant of $f_1(M)$ and every prime implicant of $f_2(M)$ have exactly one variable in common.*

Proof. Obviously, any two such prime implicants must have *at least* one common variable, because every row and every column in M intersect, i.e., row s^1 and column s^2 always have a common outcome $x = g(s^1, s^2)$. Let us suppose that they have another common outcome, i.e., there exist strategies s_i^1 and s_j^2 such that $g(s^1, s_j^2) = g(s_i^1, s^2) = x' \neq x$. Then, $g(s^1, s^2) = x$, thus g is not rectangular.

Conversely, let us assume that g is not rectangular, that is, $g(s_1^1, s_1^2) = g(s_2^1, s_2^2) = x$, while $g(s_1^1, s_2^2) = x' \neq x$. Then row s_1^1 and column s_2^2 have at least two outcomes in common, namely, x and x'. □

Theorem 10.13. *(Gurvich [394, 395]). A normal game form g is positional if and only if it is tight and rectangular.*

Proof. The normal form g corresponding to an extensive game form $T(f)$ is tight in view of Theorem 10.11, and g is rectangular in view of Theorem 10.12 and Theorem 10.6(iv).

Conversely, if g is tight and rectangular, then by definition, $f_1(g)$ and $f_2(g)$ are dual. Further, according to Theorem 10.12, every prime implicant of $f_1(g)$ and every prime implicant of $f_2(g)$ have exactly one variable in common. Hence, by Theorem 10.6(iv), $f_1(g)$ and $f_2(g)$ are read-once, thus g is positional. □

Remark 10.8. In [394] this theorem is generalized for game forms of n players. The criterion is the same: a game form is positional if and only if it is tight and rectangular. The proof is based on the cotree decomposition of P_4-free graphs; see Sections 10.3, 10.5. □

10.8 Historical notes

We conclude this chapter with a few brief remarks about the history of read-once functions. It is important to distinguish between

(A) the algorithms to verify read-onceness based on (*) the $\vee - \wedge$ *disjoint* decomposition,

and

(B) the criteria of read-onceness based on P_4-freeness and the "rectangularity" or "normality" for the pair f and f^d.

In fact, (A) is at least 20 years older than (B). The oldest reference which we know is by Kuznetsov [552] in 1958 (in the same famous MIAN-51-volume where Trakhtenbrot's paper appeared.) Kuznetsov claims that the decomposition (*) is well defined (i.e., it is unique), and he also says a few words on how to get it; De Morgan's formulae are mentioned, too. This implies (A), though read-onceness is not mentioned explicitly in this paper.

In his 1978 doctoral thesis, Gurvich [394] remarked that the decomposition (*) is a must for any minimum $\vee - \wedge$-formula for f in both the monotone and general cases. However, a bit earlier, Michel Chein has a short paper [173] based on his doctoral thesis of 1967, which may be the earliest one mentioning "read-once" functions. J. Kuntzmann (Chein's thesis advisor) raised the question a few years earlier in the first edition (1965) of his book "Algèbre de Boole" [549], mentioning a problem called "dédoublement de variables", and in the second edition (1968) already cites Chein's work.

What Chein does (using our notation) is to look at the bipartite graph $B(f) = (\mathcal{P}, V, E)$, where $\mathcal{P}$ is the set of prime implicants, V is the set of variables, and edges represent containment, that is, for all $P \in \mathcal{P}, v \in V$,

$$(P, v) \in E \iff v \in P.$$

The reader can easily verify that $B(f)$ is connected if and only if the (Gurvich) graph $G(f)$ is connected if and only if the (Peer-Pinter) graph $H(f)$ is connected.

Chein's method is to check which of $B(f)$ or $B(f^d)$ is disconnected (failing if both are connected) and continuing recursively, just like Peer-Pinter do on $H(f)$ and $H(f^d)$. The exponential price is paid for dualizing.

In contrast, as the reader also knows now, the polynomial time algorithm of Golumbic-Mintz-Rotics acts similarly on $G(f)$ and $G(f^d)$, but $G(f^d)$ is gotten for free without dualizing thanks to Gurvich's theorem, that $G(f^d)$ equals the graph complement of $(G(f))$ paying only an extra lower price to check for normality.

Finally, just to clarify complexities, using our notation: Of course, building $B(f^d)$ involves dualization of f, however, building $G(f^d)$ can be done in polynomial time for any positive Boolean function (i.e., without any dualization)!! The implication is that one can compute a top-level (unique) read-once decomposition for any positive Boolean function in polynomial time (see also Ramamurthy's book [723]).

To summarize, the decomposition (*) is a must in the minimization of f by a $\vee - \wedge$-formula and the read-onceness is just an extreme case in such a minimization. Yet, (*) implies (A) and has been known since 1958, while (B) has been known since 1977 and rediscovered independently several times thereafter. Dominique de Werra has described it as, "an additional interesting example of rediscovery by people from the same scientific community. It shows that the problem has kept its importance and [those involved] have good taste."

10.9 Exercises

1. Prove that a Boolean function f for which some variable appears in its positive form x in one prime implicant and in its negative form $\overline{x}$ in another prime implicant cannot be a read-once function.

2. Verify Remark 10.1, namely, if T is a proper dual subimplicant of f, then there exists a prime implicant of f, say P, such that $P \cap T = \emptyset$.

3. Consider the positive Boolean function

$$f = x_1x_2 \vee x_1x_5 \vee x_2x_3 \vee x_2x_4 \vee x_3x_4 \vee x_4x_5.$$

(a) Draw the co-occurrence graph $G(f)$. Prove that f is not a read-once function.

(b) Let $T = \{x_1, x_4\}$. What are the sets $\mathcal{P}_0, \mathcal{P}_{x_1}, \mathcal{P}_{x_4}$? Prove that T is a dual subimplicant of f, by finding a non-covering selection.

(c) Let $T' = \{x_3, x_4, x_5\}$. What are the sets $\mathcal{P}'_0, \mathcal{P}'_{x_3}, \mathcal{P}'_{x_4}, \mathcal{P}'_{x_5}$? Prove that T' is not a dual subimplicant of f.

4. Consider the function $f = ab \vee bc \vee cd$. Verify that $\{a, d\}$ is not a dual subimplicant.

5. Verify that the function

$$f = adg \vee adh \vee bdg \vee bdh \vee eag \vee ebg \vee ecg \vee eh$$

in Example 10.2 is not a normal function. Find the collection $\mathcal{D}$ of dual prime implicants of f. Is f^d normal?

6. Let f be a positive Boolean function over the variable set $\{x_1, x_2, ..., x_n\}$, and let T be a subset of the variables. Prove the following:

(a) T is a dual prime implicant if and only if $\mathcal{P}_0 = \emptyset$ and there is a non-empty selection $\mathcal{S}$ for T (i.e., $\mathcal{P}_{x_i} \neq \emptyset$ for every $x_i \in T$).

(b) T is a dual super implicant (i.e., $D \subset T$ for some dual prime implicant $D \in \mathcal{D}$) if and only if $\mathcal{P}_0 = \emptyset$ and $\mathcal{P}_{x_i} = \emptyset$ for some $x_i \in T$ (i.e., no selection $\mathcal{S}$ is possible).

7. Prove that for any graph G, $\overline{G}$ must be connected if G is disconnected.

8. Give a direct proof (using the dual subimplicant theorem) of the implication (iii) $\Longrightarrow$ (ii) of Theorem 10.4, namely, if $G(f)$ and $G(f^d)$ do not share a common edge then $G(f)$ and $G(f^d)$ are complementary graphs.

9. Using Lemma 10.1, prove that the read-once expression is unique for a read-once function (up to commutativity of the operations $\vee$ and $\wedge$).

10. Verify that the function $f = abc \vee bde \vee ceg$ from Example 10.3 is not normal, though its three prime implicants correspond to maximal cliques of the co-occurrence graph $G(f)$, see Figure 10.4. Verify that $G(f)$ contains an induced P_4. How many P_4's does it contain?

11. Consider two functions:

$$f_1 = x_1x_3x_5 \vee x_1x_3x_6 \vee x_1x_4x_5 \vee x_1x_4x_6 \vee x_2x_3x_5 \vee x_2x_3x_6 \vee x_2x_4x_5 \vee x_2x_4x_6$$

and

$$f_2 = x_1x_3x_5 \vee x_1x_3x_6 \vee x_1x_4x_5 \vee x_1x_4x_6 \vee x_2x_3x_5 \vee x_2x_3x_6 \vee x_2x_4x_5.$$

Verify that they generate the same co-occurrence graph G which is P_4-free and that all prime implicants of f_1 and f_2 correspond to maximal cliques of G; yet, f_1 is normal, while f_2 is not. Find the cotree for G and the read-once expression for f_1.

12. Give an example of a pair of functions g and f with same co-occurrence graph $G = G(g) = G(f)$ which is P_4-free and where the number of prime implicants of g and f are equal; yet, g is normal, and thus read-once, while f is not. (Hint: Combine non-normal functions that you have seen in this chapter whose graphs are P_4-free.)

13. Prove that for a positive Boolean function given by its complete DNF expression, it is possible to check normality in $O(n^3k)$ time, where n is the number of essential variables and k is the number of prime implicants of the function. (Hint: use the results of [504].)

14. Prove Lemma 10.3: Let f be a non-constant read-once function over the variables $N = \{x_1, x_2, \ldots, x_n\}$. Suppose that D_i is a dual prime implicant containing x_i but not x_j, and that D_j is a dual prime implicant containing x_j but not x_i. Let $R_{i,j} = (N \setminus (D_i \cup D_j)) \cup \{x_i, x_j\}$. Then (x_i, x_j) is an edge in the co-occurrence graph $G(f)$ if and only if $R_{i,j}$ contains a prime implicant. (Hint: use (iv) of Theorem 10.6 or see reference [20].)

15. Prove that the recursive definition of cographs based on rules $(1), (2), (3)$ in Section 10.4 is equivalent to the alternative definition using rules $(1), (2), (3')$.

16. Let f be a positive Boolean function over the variables $N = \{x_1, x_2, \ldots, x_n\}$, and let $U \subseteq N$.

    (a) Prove that the following greedy algorithm FIND-PI-IN(U) finds a prime implicant $P \subseteq U$ of f, if one exists, and can be implemented to run in $O(n)$ time using $|U|$ membership queries. (We denote by X^U the characteristic vector of U where $x_i^U = 1$ for $x_i \in U$, and $x_i^U = 0$ otherwise.)

    Algorithm FIND-PI-IN(U)

```
Step 1: Verify that $f(X^U) = 1$.
        Otherwise, exit with no solution since $U$ contains no prime implicant.
Step 2: Set $S \leftarrow U$.
Step 3: For all $x_i \in U$, do
        if $f(X^{S \setminus \{x_i\}}) = 1$ then $S \leftarrow S \setminus \{x_i\}$
        end-do
```

```
Step 4: Set $P \leftarrow S$ and output $P$.
```

(b) Write an analogous dual **Algorithm** Find-DualPI-In(U) to find a dual prime implicant $D \subseteq U$ of f, if one exists.

17. The next three exercises are due to [20].

    Prove the following: Let f be a non-constant read-once function, and let Y be a non-empty subset of its variables. Then Y is the set of essential variables of f if and only if for every variable $x_i \in Y$, x_i is contained in a prime implicant of f that is a subset of Y, and x_i is contained in a dual prime implicant of f that is a subset of Y.

18. Let f be a read-once function over the set of variables $N = \{x_1, x_2, \ldots, x_n\}$. Prove the following: If S is a prime implicant of f containing the variable x_i, then $(N \setminus S) \cup \{x_i\}$ contains a dual prime implicant of f, and any such dual prime implicant contains x_i. Dually, if T is a dual prime implicant of f containing the variable x_i, then $(N \setminus T) \cup \{x_i\}$ contains a prime implicant of f, and any such prime implicant contains x_i.

19. Let f be a read-once function over the set of variables $N = \{x_1, x_2, \ldots, x_n\}$. Using Exercises 16, 17 and 18, prove that the following algorithm finds the set Y of essential variables and can be implemented to run in $O(n^2)$ time using $O(n^2)$ membership queries. In the process, for each variable x_i in Y, it generates a prime implicant $P[i]$ and a dual prime implicant $D[i]$ containing x_i.

    Algorithm Find-Essential-Variables

```
Step 1: Set $P[i] \leftarrow D[i] \leftarrow \emptyset$ for $i = 1, \ldots, n$.
Step 2: Set $W \leftarrow P \leftarrow$ Find-PI-In($N$), and
        for each $x_j \in P$, set $P[j] \leftarrow P$.
Step 3: While there exists $x_i \in N$ such that exactly one of $P[i]$ and $D[i]$ is $\emptyset$, do
    (3a:) if $D[i] = \emptyset$, then set $D \leftarrow$ Find-DualPI-In($(N \setminus P[i]) \cup \{x_i\}$), and
          for each $x_j \in D$, set $D[j] \leftarrow D$, and set $W \leftarrow W \cup D$.
    (3b:) if $P[i] = \emptyset$, then set $P \leftarrow$ Find-PI-In($(N \setminus D[i]) \cup \{x_i\}$), and
          for each $x_j \in P$, set $P[j] \leftarrow P$, and set $W \leftarrow W \cup P$.
          end-do
Step 4: Set $Y \leftarrow W$ and output $Y$.
```

20. (From Lisa Hellerstein.) Consider the function

$$f_1 = x_1 \vee x_2 \vee ... \vee x_n$$

and the class of functions $\mathcal{F} = \{f_A\}$, where A is an element in $\{0,1\}^n$ having at least two 1's, and

$$f_A(X) = 1 \iff f_1(X) = 1 \text{ and } X \neq A.$$

(a) Prove that the functions f_A are not monotone.

(b) Prove that determining that a function is equal to f_1 and not some f_A requires querying all possible A's, and there are $\Theta(2^n)$ of them.

21. Prove directly that the normal form of any extensive game form is rectangular. In other words, if two pairs of strategies (s_1^1, s_1^2) and (s_2^1, s_2^2) result in the same play p, that is, $p(s_1^1, s_1^2) = p(s_2^1, s_2^2) = p$, then (s_1^1, s_2^2) and (s_2^1, s_1^2) also result in the same play, that is, $p(s_1^1, s_2^2) = p(s_2^1, s_1^2) = p$.

22. Verify that the following two game forms are tight:

$$M_4 = \begin{bmatrix} x_1 & x_2 & x_1 & x_2 \\ x_3 & x_4 & x_4 & x_3 \\ x_1 & x_4 & x_1 & x_5 \\ x_3 & x_2 & x_6 & x_2 \end{bmatrix},$$

$$M_5 = \begin{bmatrix} x_1 & x_1 & x_2 \\ x_1 & x_1 & x_3 \\ x_2 & x_4 & x_2 \end{bmatrix}.$$

Questions for thought

23. To what extent is Lemma 10.1 true for all expressions, that is, not just the read-once formula and the DNF formula of prime implicants?

24. The polynomial time complexity given in Theorem 10.5 can (almost certainly) be improved by a more careful choice of data structures. In this direction, what is the complexity of calculating $\mathcal{P}_0$ and P_{x_i} for all x_i? Consider using bit vectors to represent sets of variables.

25. What can be said about the complexity of recognizing read-once functions if the input formula is not a DNF, but some other type of representation such as a BDD or an arbitrary Boolean expression? In such a case, we might have to pay a high price to convert the formula into a DNF or CNF and use the GMR method of Section 10.5. When is there an efficient alternative way to build the co-occurrence graph $G(f)$ directly from a representation of f which is different from the DNF or CNF expression? What assumptions must be made regarding f? When can normality also be tested?

    Lisa Hellerstein (private communication) has pointed out that if ψ is a non-monotone DNF expression, the read-once recognition problem is coNP-complete. How does this impact the answer?

26. Give a counterexample to show that the statement in Exercise 17 may fail when f is a positive Boolean function, but f is not read-once. Show that for an arbitrary positive Boolean function f, identifying the set of essential variables may require an exponential number of calls on a membership oracle.

27. What would happen if we attempted to apply the read-once oracle learning method to a positive function f which was *not* read-once? In other words, in the building of the co-occurrence graph (Step 1), how did we rely upon the read-once assumption? Would the oracle fail, in which case we would know that f is not read-once, or would it produce some other graph? What graph would we get? When would it still yield the correct co-occurrence graph $G(f)$? If so, we can easily test whether it is a cograph, but how could we test whether the function is normal? For example, consider what would happen for the functions f_1 and f_2 of Section 10.1. Could the oracle generate all prime implicants? What would be the complexity?

28. The two game forms M_4 and M_5 in Exercise 22 represent the normal form of some extensive games on graphs which have no terminal positions and their cycles are the outcomes of the game. Find two graphs which generate M_4 and M_5.