Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 102 (2020)

Notes on Hazard-Free Circuits

Stasys Jukna[1]

Faculty of Mathematics and Computer Science, Vilnius University, Lithuania.

Abstract

The problem of constructing hazard-free Boolean circuits (those avoiding electronic glitches) dates back to the 1940s and is an important problem in circuit design. We show that a DeMorgan circuit is hazard-free if and only if the circuit produces (purely syntactically) all prime implicants as well as all prime implicates of the Boolean function it computes. This extends to arbitrary DeMorgan circuits a classical result of Eichelberger [*IBM J. Res. Develop.*, 9 (1965)] showing this property for special depth-two circuits. Also, Ikenmeyer et al. [*J. ACM*, 66:4 (2019), Article 25] have recently shown that the hazard-free circuit complexity of any Boolean function $f(x)$ is lower-bounded by the monotone circuit complexity of the monotone Boolean function which accepts an input x iff $f(z) = 1$ for some vector $z \leqslant x$. We give a short and amazingly simple proof of this interesting result. Finally, we give a very simple (non-monotone) Boolean function whose hazard-free circuit complexity is super-polynomially larger than its unrestricted circuit complexity.

Keywords: hazard-free circuits, monotone circuit, lower bounds

1. Introduction

The problem of designing hazard-free circuits naturally occurs when implementation circuits in hardware ([4, 10], but is also closely related to questions in logic ([12, 14, 17]) and even in cybersecurity ([9]). The importance of hazard-free circuits is already highlighted in the classical textbook [4].

If not stated otherwise, by a *circuit* we will understand a *DeMorgan circuit*, that is, a Boolean circuit over $\{\wedge, \vee, ^-\}$, where negations are applied only to input variables. Thus, such a circuit uses AND and OR operations at gates. Inputs are constants 0 and 1, variables $x_1, \ldots, x_n$ and their negations $\overline{x}_1, \ldots, \overline{x}_n$. A *monotone* circuit is a DeMorgan circuit without negated inputs.

Hazards are spurious pulses or electronic glitches that may occur on the output of circuits during an input transition, stipulated by physical delays at wires or gates in a specific hardware implementation of the circuit: circuits are usually asynchronous. Having designed a hazard-free circuit for a given Boolean function f, one is sure that no glitches will occur in *any* hardware implementation of this circuit, regardless of the physical delays.

To be a bit more specific, associate with every pair of vectors $a, b \in \{0,1\}^n$ the subcube $[a, b] \subseteq \{0,1\}^n$ consisting of all 0-1 vectors c such that $a_i = b_i$ implies $c_i = a_i$. Hence, if a and b differ in d positions, then $[a, b]$ contains 2^d vectors. A Boolean circuit F computing a Boolean function f has a 0-*hazard* if there are two vectors $a \neq b$ such that $f(c) = 0$ for all $c \in [a, b]$ but after the input a is replaced by b, some hardware implementation of this circuit may have a spurious 0-1-0 glitch, that is, for a short moment, the circuit may output the (wrong) value 1 on the new input b. If $f(c) = 1$ for all $c \in [a, b]$ but after the input a is replaced by b, the circuit may have a spurious 1-0-1 glitch, then

Email address: `stjukna@gmail.com` (Stasys Jukna)

[1]Research supported by the DFG grant JU 3105/1-2 (German Research Foundation).

ISSN 1433-8092

we have a 1-*hazard*. A circuit is *hazard-free* if it has no hazards; see Section 3 for a precise definition (without using the vague notion of "possible glitches").

For example, if the output of the AND gate xz of the circuit $F = xz \vee y\overline{z}$ reaches the output OR gate *later* than that of the AND gate $y\overline{z}$ (due to different delays at wires or gates), and if we (suddenly) replace input $a = (1,1,0)$ by $b = (1,1,1)$, then the circuit will shortly output 0 before it outputs the correct value 1 fired by the AND gate xz on the new input: for a short moment, the OR gate will see the value 0 of $y\overline{z}$ on the *new* input b, and the value 0 of xz on the *old* input a. Thus, some hardware implementations of the circuit F may have spurious 1-0-1 glitches (the circuit F has a 1-hazards). However, the circuit $F' = xy \vee xz \vee y\overline{z}$ for the same Boolean function is already hazard-free.

That every Boolean function f can be computed by a hazard-free DeMorgan circuit was shown by Huffman [10] already in 1957: the OR of all prime implicants of f is hazard-free. In 1965, Eichelberger [6, Theorem 2] extended this result: any DNF without zero terms (those containing a variable together with its negation) for f is hazard-free if and *only if* it contains all prime implicants of f as terms. In particular, Eichelberger's theorem implies that every monotone circuit is hazard-free (see also Corollary 2). Let us also mention that detecting the presence of hazards is a difficult computational task. In particular, as recently shown by Komarath and Saurabh [13], if the strong exponential time hypothesis (a statement stronger that P≠NP) is true, then for any $\epsilon > 0$ there is no algorithm that runs in time $3^{(1-\epsilon)n}\mathrm{poly}(s)$ and detects a presence of hazards already in depth-4 DeMorgan formulas of size s and with n variables.

In this paper, we (1) extend Eichelberger's theorem to arbitrary DeMorgan circuits, (2) present a very simple non-monotone Boolean function whose hazard-free circuit complexity is super-polynomially larger than its unrestricted circuit complexity, and (3) show that the monotone circuit obtained from any hazard-free circuit F by just replacing negated input literals by constant 1 computes the monotone "downwards closure" $f^{\triangledown}$ of the Boolean function f computed by F, where $f^{\triangledown}(x) = 1$ iff $f(z) = 1$ for some $z \leqslant x$. In particular, (3) implies that the *hazard-free* circuit complexity of any Boolean function f is lower-bounded by the *monotone* circuit complexity of $f^{\triangledown}$. This latter lower bound itself is not new: it was proved in a recent paper by Ikenmeyer et al. [11] using different arguments—our contribution to this bound is the amazing simplicity of the proof.

We now describe the results in more details, and put them in the context of the previous work on hazard-free circuits.

2. Results

Every DeMorgan circuit F not only computes a unique Boolean function, but also *produces* (purely syntactically) a unique set of terms (ANDs of literals) as well as a unique set of clauses (ORs of literals) in a natural way. The *formal DNF* of F is the OR of all terms produced by F, and the *formal CNF* of F is the AND of all clauses produced by F. Roughly, these DNFs and CNFs are obtained by just applying the distributive laws to reduce the circuit F to a depth-2 formula. The blow up in size is here irrelevant: important only is that the circuit F has a hazard iff the resulting depth-2 formula contains a hazard; see Section 3 for a precise definition of these natural concepts, as well as for a reminder of what "prime implicants" and "prime implicates" of a Boolean function are.

At this point, we only note that when forming the sets of produced terms and clauses, the annihilation laws $x \wedge \overline{x} = 0$ and $x \vee \overline{x} = 1$ are not used. In particular, some produced terms may be *zero terms* (or *trivial terms*), and some produced clauses may be *trivial clauses*, that is, may contain a variable together with its negation.

If a DeMorgan circuit F computes a Boolean function f, then every nonzero term t of the formal DNF D of F must be an implicant of f. But, in general, a prime implicant of f does not need to be even a subterm of some term of D. For example, no term of the formal DNF$D = x\overline{y} \vee \overline{x}y \vee x\overline{z}$ of the circuit $F = x(\overline{y} \vee \overline{z}) \vee \overline{x}y$ computing the function $f = x\overline{y} \vee \overline{x}y \vee x\overline{z}$ contains the prime implicant $y\overline{z}$ of f as a subterm, and the circuit

Theorem 1. *Let F be a DeMorgan circuit computing a Boolean function f.*

1. *F has no 1-hazards if and only if F produces all prime implicants of f.*
2. *F has no 0-hazards if and only if F produces all prime implicates of f.*

Theorem 1 is a special case of two general Theorems 4 and 5 proved in Section 5. These two theorems also give the following sufficient conditions for hazard-freeness.

Theorem 2. *Let F be a DeMorgan circuit.*

1. *F has no 1-hazards if F produces no trivial clauses.*
2. *F has no 0-hazards if F produces no zero terms.*

A DeMorgan circuit is *zero-term free* if it produces no zero terms. By Theorem 2, such circuit cannot have any 0-hazards. So, Theorem 1 yields the following classical result of Eichelberger [6, Theorem 2]. This result is stated and proved in [6] only for zero-term free DNFs, but the argument works also for zero-term free DeMorgan circuits: formal DNFs of such circuits do not have zero terms.

Corollary 1 (Eichelberger [6])**.** *A zero-term free DeMorgan circuit F computing a Boolean function f is hazard-free if and only if F produces all prime implicants of f.*

Since monotone circuits cannot produce any zero terms (they have no negated inputs at all), and since they must produce all prime implicants of the computed monotone Boolean function (see Theorem 4.2 in [24, Chapter 2] if not sure), Corollary 1 directly yields the following fact; [11, Lemma 4.2] gives an alternative proof of this fact by induction on the circuit size.

Corollary 2. *Monotone circuits are hazard-free.*

After these structural results, we turn to the relation of the hazard-free circuit complexity with the monotone circuit complexity. The *downwards closure* of a Boolean function $f(x)$ is the monotone Boolean function $f^{\triangledown}(x) := \bigvee_{z \leqslant x} f(z)$. That is, $f^{\triangledown}(x) = 1$ iff $f(z) = 1$ for some vector $z \leqslant x$, that is, if $z_i \leqslant x_i$ for all positions i. For example, if $f(x) = 1$ precisely when the graph encoded by vector x consists of a clique on k vertices and $n - k$ isolated vertices, then $f^{\triangledown}(x)$ is the well-known k-clique function. Note that $f^{\triangledown} = f$ holds for all monotone Boolean functions f. Recall that a Boolean function f is *monotone* if $z \leqslant x$ implies $f(z) \leqslant f(x)$.

We can view every DeMorgan circuit $F(x)$ computing a Boolean function $f(x)$ of n variables as a monotone circuit $H(x, y)$ on $2n$ variables with the property that $F(x) = H(x, \overline{x})$ holds for all $x \in \{0, 1\}^n$. The *positive version* of the circuit $F(x)$ is the monotone circuit $F_+(x) = H(x, \vec{1})$ obtained by replacing every negated input literal $\overline{x}_i$ with constant 1.

Remark 1. Since the circuit $H(x, y)$ is monotone, and since the circuit $F_+(x) = H(x, \vec{1})$ is obtained from H by replacing with constant 1 all y-inputs, we have $F_+(x) \geqslant f(x)$, and since the circuit $F_+(x)$ is monotone, we also have $F_+(x) \geqslant F_+(z)$ for every $z \leqslant x$. So, $F_+(x) \geqslant f^{\triangledown}(x)$ holds for all $x \in \{0, 1\}^n$.

But $F_+ \neq f^{\triangledown}$, in general. Take, for example, the circuit $F = xy \vee \overline{x}x\overline{y}$ computing $f = xy$. Then $F_+ = xy \vee x$, and $F_+(1, 0) = 1$ but $f^{\triangledown}(1, 0) = f(1, 0) \vee f(0, 0) = 0$. The following theorem gives us a general condition for $F_+ = f^{\triangledown}$ to hold, and shows that 0-hazard free circuits fulfill this condition. The *positive factor* of a term is obtained by replacing every its negated literal with constant 1.

Theorem 3. *Let F be a DeMorgan circuit computing a Boolean function f such that $f(\vec{0}) = 0$. Then the circuit F_+ computes $f^\triangledown$ if and only if the positive factor of every zero term produced by F is an implicant of $f^\triangledown$. If the circuit F has no 0-hazards, then F_+ computes $f^\triangledown$.*

In particular, $F_+ = f^\triangledown$ also holds for every zero-term free circuit F, as well as for every hazard-free circuit F.

For a Boolean function f, let $L(f)$ denote the minimum number of gates in a (unrestricted) DeMorgan circuit computing f. Let also $L_{\mathfrak{u}}(f)$, $L_{\mathrm{m}}(f)$ and $L_{\mathrm{pz}}(f)$ denote the versions of this measure when restricted, respectively, to hazard-free circuits, to monotone circuits, and to zero-term free circuits producing all prime implicants of f. For every Boolean function f, we have

$$L_{\mathrm{m}}(f^\triangledown) \leqslant L_{\mathfrak{u}}(f) \leqslant L_{\mathrm{pz}}(f)\,, \tag{1}$$

where the first inequality follows from Theorem 3 and the second from Corollary 1. If the function f is monotone, then Eq. (1) and Corollary 2 give the equality $L_{\mathfrak{u}}(f) = L_{\mathrm{m}}(f)$.

Let us note that the first inequality in Eq. (1) was already proved in a recent paper by Ikenmeyer et al. [11] using different arguments. Namely, the proof in [11] first introduces an interesting concept of "hazard derivatives", shows a chain rule for these derivatives, and uses this rule to transform a given hazard-free circuit into a monotone circuit. The argument is reminiscent of that used by Baur and Strassen [2] to compute all partial derivatives of a multivariate polynomial by an arithmetic circuit.

In contrast, our proof of Theorem 3 (in Section 4) is direct and amazingly simple. Actually, for monotone Boolean functions f, Theorem 3 tells us a bit more than just the equality $L_{\mathfrak{u}}(f) = L_{\mathrm{m}}(f)$: it shows that not only hazard-free and monotone circuit *complexities* for monotone Boolean function f do coincide but, in fact, that every minimal hazard-free circuit for f *is* a monotone circuit *itself*, that is, does not use negated input gates to compute its values.

Together with already known lover bounds on the monotone circuit complexity, the inequality $L_{\mathfrak{u}}(f) \geqslant L_{\mathrm{m}}(f^\triangledown)$, implies that the gap $L_{\mathfrak{u}}(f)/L(f)$ can be super-polynomial and even exponential. This consequence was shown in [11], when f is either the logical permanent [20] or the logical determinant, or the Tardos function [23]. However, the known circuits for these functions (demonstrating that $L(f)$ is polynomial) are far from being trivial. Actually, we even do not have *explicit* constructions of these circuits—we only have general algorithms: [15, 8] for logical permanent and [7] for the Tardos function. In Section 6, we observe that the super-polynomial gap $L_{\mathfrak{u}}(f_n)/L(f_n) = n^{\Omega(\log n)}$ is already achieved on a very simple Boolean function f_n in n^2 variables (Lemma 1): view the input x as an $n \times n$ matrix, and let $f_n(x) = 1$ iff x is a permutation matrix, that is, if every row and every column has exactly one 1.

3. Preliminaries

Here we give definitions of hazards, formal DNFs and formal CNFs of DeMorgan circuits.

3.1. Hazard-free circuits

In this paper, we ignore the electronic aspect of hazards, and stick on their idealized, mathematical model as, for example, in [3, 6, 18, 25]. The classical Kleene's three-valued "strong logic of indeterminacy" [12] extends the Boolean operations AND, OR and NOT from the Boolean domain $\mathbb{B} = \{0, 1\}$ to the ternary domain $\mathbb{T} = \{0, \mathfrak{u}, 1\}$ (where the bits 0 and 1 are interpreted as *stable*, and the bit $\mathfrak{u}$ as *unstable*):

and	0	$\mathfrak{u}$	1
0	0	0	0
$\mathfrak{u}$	0	$\mathfrak{u}$	$\mathfrak{u}$
1	0	$\mathfrak{u}$	1

or	0	$\mathfrak{u}$	1
0	0	$\mathfrak{u}$	1
$\mathfrak{u}$	$\mathfrak{u}$	$\mathfrak{u}$	1
1	1	1	1

not	0	$\mathfrak{u}$	1
	1	$\mathfrak{u}$	0

Note that, if we define the unstable bit as $\mathfrak{u} = \frac{1}{2}$, then these ternary operations turn into tropical operations: $x \wedge y = \min(x, y)$, $x \vee y = \max(x, y)$ and $\overline{x} = 1 - x$. It is easy to verify that the system $(\mathbb{T}, \vee, \wedge)$ forms a distributive lattice with zero element 0 and universal element 1; see, for example, Yoeli and Rinon [25]. In particular, the operations $\vee$ and $\wedge$ are associative, and commutative. Moreover, the elements 0 and 1 satisfy for every $x \in \mathbb{T}$: $x \wedge 0 = 0$, $x \wedge 1 = x$, $x \vee 0 = x$ and $x \vee 1 = 1$. The absorbtion law $x \vee xy = x$ as well as the rules of de Morgan $\overline{x \vee y} = \overline{x} \wedge \overline{y}$ and $\overline{x \wedge y} = \overline{x} \vee \overline{y}$ also hold. The *only* difference from the Boolean algebra is that the annihilation laws $x \wedge \overline{x} = 0$ and $x \vee \overline{x} = 1$ *do not* hold over the ternary domain $\mathbb{T} = \{0, \mathfrak{u}, 1\}$: $0 \wedge \mathfrak{u} = 0$ but $\mathfrak{u} \wedge \bar{\mathfrak{u}} = \mathfrak{u} \neq 0$, and $1 \vee \mathfrak{u} = 1$ but $\mathfrak{u} \vee \bar{\mathfrak{u}} = \mathfrak{u} \neq 1$.

A *refinement* of a ternary vector $\alpha \in \mathbb{T}^n$ is a vector in $\mathbb{B}^n$ obtained from α by replacing every occurrence of the unstable bit $\mathfrak{u}$ by constant 0 or 1. The *subcube* defined by α is the set

$$A_\alpha = \{a \in \{0,1\}^n \colon a \text{ is a refinement of } \alpha\}$$

of all refinements of α. After an appropriate permutation of the set $\{1, \ldots, n\}$ of positions, the subcube A_α has the form $A_\alpha = \{b\} \times \{0, 1\}^m$ for some vector $b \in \{0, 1\}^{n-m}$, where m is the number of unstable bits $\mathfrak{u}$ in α. To match the notation $[a, b]$ for subcubes of $\{0, 1\}^n$ we used in the introduction, just note that any pair of vectors $a, b \in \{0, 1\}^n$ defines the ternary vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$ with $\alpha_i = \mathfrak{u}$ when $a_i \neq b_i$, and $\alpha_i = a_i$ when $a_i = b_i$. Then $[a, b] = A_\alpha$.

For a Boolean function f and a set $A \subseteq \{0, 1\}^n$, let $f(A_\alpha) = \{f(a) \colon a \in A\} \subseteq \{0, 1\}$ denote the set of values taken by f on A. Hence, we always have $|f(A)| = 1$ or $|f(A)| = 2$, and $|f(A)| = 1$ iff f is constant on A.

Definition 1 (Hazards). A circuit F has a 0-*hazard* at $\alpha \in \mathbb{T}^n$ if $F(A_\alpha) = \{0\}$ but still $F(\alpha) = \mathfrak{u}$ holds. If $F(A_\alpha) = \{1\}$ but $F(\alpha) = \mathfrak{u}$, then F has a 1-*hazard* at α. The circuit is *hazard-free* if it has a hazard at none of the inputs $\alpha \in \mathbb{T}^n$.

That is, after the functions AND, OR, NOT computed at individual gates are extended from the binary domain $\mathbb{B} = \{0, 1\}$ to the ternary domain $\mathbb{T} = \{0, \mathfrak{u}, 1\}$, every DeMorgan $\{\wedge, \vee, \neg\}$ circuit F computing a given Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ turns into a circuit computing a (unique) ternary function $F : \mathbb{T}^n \to \mathbb{T}$, a "ternary extension" of F, which coincides with f on $\mathbb{B}^n$. Even if two circuits compute the same Boolean function, their ternary extensions may be different. Whether a circuit F is hazard-free or not depends entirely on the properties of its ternary extension which, in turn, depends on the specific *form* of the circuit F. Namely, the circuit F has a hazard at some vector $\alpha \in \mathbb{T}^n$ if the Boolean function f computed by F does not depend on the unstable bits of α, but still F outputs the unstable bit $\mathfrak{u}$ on the input α. There are several types of hazards—that given in Definition 1 is of so-called *static logical hazards* [3, 6, 25].

Example 1. Consider the function $f(x, y, z) = xz \vee y\overline{z}$. The trivial circuit $F = xz \vee y\overline{z}$ for this function has a 1-hazard at $\alpha = (1, 1, \mathfrak{u})$: although $f(1, 1, 0) = f(1, 1, 1) = 1$, we still have $F(1, 1, \mathfrak{u}) = (1 \wedge \mathfrak{u}) \vee (1 \wedge \bar{\mathfrak{u}}) = \mathfrak{u}$. The circuit $F'' = (x \vee \overline{z})(y \vee z)$ for the same function f has a 0-hazard at $\alpha = (0, 0, \mathfrak{u})$: although $f(0, 0, 0) = f(0, 0, 1) = 0$, we still have $F(0, 0, \mathfrak{u}) = \bar{\mathfrak{u}} \wedge \mathfrak{u} = \mathfrak{u}$. But, for example, the circuit $F'' = xy \vee xz \vee y\overline{z}$ for the same function is already hazard-free: it has no 0-hazards because the formal DNF $D = xy \vee xz \vee y\overline{z}$ has no zero terms (see Theorem 5), and has no 1-hazards because D contains all three prime implicant of f (see Theorem 1).

3.2. Prime implicants and implicates of functions

We use a standard notation concerning Boolean functions and circuits [5, 24]. In particular, a *literal* is either a variable $x_i = x_i^1$ or its negation $\overline{x}_i = x_i^0$. A *term* is an AND of literals or a constant 0 or 1, and a *clause* if an OR of literals. A term is a *zero term* if it contains a variable x_i together with its negation $\overline{x}_i$ (note that 0 is not a zero term, it is a constant term). Similarly, a *clause* is an OR of literals. A clause containing a variable together with its negation is a *trivial clause* (we do not use the term "one clause" just to avoid misinterpretation).

An *implicant* of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is a nonzero term t such that $t \leqslant f$ holds, that is, for every $a \in \{0,1\}^n$, $t(a) = 1$ implies $f(a) = 1$. An implicant t is a *prime implicant* of f if no proper subterm of t is an implicant of f. Dually, an *implicate* of f is a nontrivial clause q such that $f \leqslant q$ holds, that is, for every $a \in \{0,1\}^n$, $q(a) = 0$ implies $f(a) = 0$. An implicate q is a *prime implicate* of f if no proper subclause of q is an implicate of f.

3.3. Formal DNFs and CNFs of circuits

As we already mentioned in Section 2, every DeMorgan circuit F not only computes a unique Boolean function, but also *produces* (purely syntactically) a unique set of terms as well as a unique set of clauses in a natural way.

Namely, each input gate holding a literal x_i or $\overline{x}_i$ produces this literal as a single term; constants produce the corresponding constant terms. The set of terms produced at an OR gate is a union of sets of terms produced at its two inputs, while the set of terms produced at an AND gate is obtained by taking the AND of every term produced at one of the inputs with every term produced at the second input. The set of terms produced by the entire circuit is the set produced at its output gate. The *formal DNF* D_F of a circuit F is the OR of all terms produced by F.

Dually, every DeMorgan circuit F *produces* (purely syntactically) a unique set of clauses. Namely, each input gate holding a literal x_i or $\overline{x}_i$ produces this literal as a single clause (consisting of this literal); constants produce the corresponding constant clauses. The set of clauses produced at an AND gate is a union of sets of clauses produced at its two inputs, while the set of terms produced at an OR gate is obtained by taking the OR of every clause produced at one of the inputs with every clause produced at the second input. The set of clauses produced by the entire circuit is the set produced at its output gate. The *formal CNF* C_F of a circuit F is the AND of all clauses produced by F.

Remark 2. Note the duality: the DNF D_F is obtained by multiplying out all clauses of the CNF C_F, that is, each term t of D_F is obtained by picking one literal from each of the causes of C_F. Dually, each clause q of C_F is obtained by picking one literal from each of the terms of D_F. Note also that if f is the Boolean function computed by the circuit F, then every nonzero term of D_F is an implicant of f, and every nontrivial clause of C_F is an implicate of f.

Let us stress an important point: when forming the sets of produced terms and clauses, the annihilation laws $x \wedge \overline{x} = 0$ and $x \vee \overline{x} = 1$ are *not* used (these two laws do not hold over $\{0, \mathfrak{u}, 1\}$). In particular, some produced terms may be *zero terms* and some produced clauses may be *trivial clauses*, that is, may contain a variable together with its negation. For example, the formal DNF $D = xy \vee xz \vee y\overline{z} \vee z\overline{z}$ of the circuit $F = (x \vee \overline{z})(y \vee z)$ computing $f = xz \vee y\overline{z}$ has a zero term $z\overline{z}$. Dually, the formal CNF $C = (x \vee y)(x \vee \overline{z})(y \vee z)(z \vee \overline{z})$ of the circuit $F = xz \vee y\overline{z}$ computing the same function f has a trivial clause $z \vee \overline{z}$.

As we already mentioned above, the system $(\{0, \mathfrak{u}, 1\}, \vee, \wedge)$ forms a distributive lattice with zero element 0 and universal element 1. In particular, AND distributes over OR, and vice versa. So, since the annihilation laws $x \wedge \overline{x} = 0$ and $x \vee \overline{x} = 1$ are *not* used when constructing the formal DNF D of the formal CNF C of a given circuit F (only these laws holding over the binary domain $\{0, 1\}$ do not extend to the ternary domain $\{0, \mathfrak{u}, 1\}$), the ternary functions computed by D and C are the *same* as that computed by the circuit F, that is, $D(\alpha) = C(\alpha) = F(\alpha)$ holds for every ternary vector $\alpha \in \mathbb{T}^n$. This is a simple but important observation which allows us to analyze the properties of ternary functions $F : \mathbb{T}^n \to \mathbb{T}$ defined by DeMorgan circuits F by analyzing the properties of their formal DNFs and formal CNFs.

We now turn to the proofs of Theorems 1 and 3. We start with that of Theorem 3 since our argument here is direct and requires no additional concepts.

4. Proof of Theorem 3

Let F be a DeMorgan circuit computing a Boolean function f such that $f(\vec{0}) = 0$, D be the formal DNF of F, and F_+ be the monotone version of F. Recall that the *positive factor* t_+ of a term t is obtained from t by replacing every negated literal with constant 1, and F_+ is obtained from the circuit F by replacing with constant 1 all negated input variables. The *downwards closure* of a Boolean function $f(x)$ is the monotone Boolean function $f^{\triangledown}(x) := \bigvee_{z \leqslant x} f(z)$. Consider the following three assertions.

(A) The circuit F has no 0-hazards.

(B) The circuit F_+ computes $f^{\triangledown}$.

(C) The positive factor t_+ of every zero term t of D is an implicant of $f^{\triangledown}$.

Our goal is to prove that (A) $\Rightarrow$ (B), and (B) $\Leftrightarrow$ (C). The implication (B) $\Rightarrow$ (C) is immediate. Indeed, if the DNF D contains a zero term t such that t_+ is not an implicant of $f^{\triangledown}$, then there must be a vector $a \in \{0,1\}^n$ such that $F_+(a) \geqslant t_+(a) = 1$ but $f^{\triangledown}(a) = 0$, that is, then F_+ does not compute $f^{\triangledown}$.

To show (A) $\Rightarrow$ (B) and (C) $\Rightarrow$ (B), assume $\neg$(B), i.e., that $F_+(x)$ does not compute $f^{\triangledown}(x) = \bigvee_{z \leqslant x} f(z)$. Since $F_+ \geqslant f^{\triangledown}$ (see Remark 1), there must be a vector $a \in \{0,1\}^n$ such that $F_+(a) = 1$ but $f^{\triangledown}(a) = 0$. The formal DNF of the circuit F_+ is obtained by replacing with constant 1 every negated literal in the formal DNF D, that is, terms produced by the circuit F_+ are positive factors of the terms of D. So, since $F_+(a) = 1$, there must be a term

$$t = \bigwedge_{i \in S} x_i \wedge \bigwedge_{i \in T} \overline{x}_i$$

in D such that $t_+(a) = 1$ holds for the positive factor $t_+ = \bigwedge_{i \in S} x_i$ of t. Since $f(\vec{0}) = 0$, we have $t(\vec{0}) = 0$ and, hence, $S \neq \varnothing$. The set $T' = \{i \in T \colon a_i = 1\}$ must be also nonempty because $f(a) = 0$ and, hence, also $t(a) = 0$. Moreover, since $t_+(a) = 1$ but $f^{\triangledown}(a) = 0$, the term t must be a zero term, that is, $S \cap T \neq \varnothing$ must hold: otherwise $t(b) = 1$ and, hence, also $f^{\triangledown}(a) \geqslant f(b) = 1$ would hold on the input $b \leqslant a$ with $b_i = 0$ for all $i \in T$ and $b_i = a_i$ for $i \notin T$. Thus, t is a zero term and t_+ is not an implicant of $f^{\triangledown}$. This shows the equivalence (B) $\Leftrightarrow$ (C).

To show the remaining implication (A) $\Rightarrow$ (B) (the second claim of Theorem 3), we have only to show that the circuit F contains a 0-hazard at some vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$. For this, write the term t as the product $t = t_1 \wedge t_2 \wedge t_3$ of three its subterms terms, where

$$t_+ = \bigwedge_{i \in S} x_i\,, \quad t_1 = \bigwedge_{j \in T'} \overline{x}_j \quad \text{and} \quad t_2 = \bigwedge_{j \in T \setminus T'} \overline{x}_j\,,$$

and consider the vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$ obtained from the binary vector a by replacing with $\mathfrak{u}$ all bits a_i for $i \in T'$ (and leaving other bits of a unchanged). On this vector, we clearly have $t_1(\alpha) = \bar{\mathfrak{u}} = \mathfrak{u}$ and $t_2(\alpha) = t_2(a) = 1$. Moreover, since $t_+(a) = 1$, we also have $t_+(\alpha) \in \{\mathfrak{u}, 1\}$, namely, $t_+(\alpha) = 1$ if $S \cap T = \varnothing$ (t is a nonzero term), and $t_+(\alpha) = \mathfrak{u}$ if $S \cap T \neq \varnothing$ (t is a zero term). Since, in our case, t is a zero term, we actually have $t_+(\alpha) = \mathfrak{u}$. Thus, $t(\alpha) = \mathfrak{u}$. Since $f(a) \leqslant f^{\triangledown}(a) = 0$, we also know that $t'(a) = 0$ and, hence, also $t'(\alpha) \in \{0, \mathfrak{u}\}$ holds for every remaining term t' of D. Thus $F(\alpha) = \mathfrak{u}$.

On the other hand, since the ternary vector α has unstable bits $\mathfrak{u}$ only in positions where the binary vector a has 1s, every refinement $b \in A_\alpha$ of α satisfies $b \leqslant a$. Since $\bigvee_{b \leqslant a} f(b) = f^{\triangledown}(a) = 0$, we have that $f(b) = 0$ holds for every refinement $b \in A_\alpha$ of α, meaning that $f(A_\alpha) = \{0\}$. Thus, the circuit F has a 0-hazard at α. ∎

Remark 3. The implication (B) $\Rightarrow$ (A) does not hold: there are circuits F such that F_+ computes the downwards closure $f^{\triangledown}$ of the Boolean function f computed by F even though F has 0-hazards. Consider the circuit $F = x\bar{x}yz \vee y\bar{z}$ computing the Boolean function $f = y\bar{z}$. The circuit has a 0-hazard at $\alpha = (\mathfrak{u}, 1, 1)$: $f(0, 1, 1) = f(1, 1, 1) = 0$ but $F(\mathfrak{u}, 1, 1) = \mathfrak{u}$. However, the positive part $t_+ = xyz$ of the (unique) zero term $t = x\bar{x}yz$ *is* an implicant of the downwards closure $f^{\triangledown}$ of the function f: $t_+(a) = 1$ can only hold for the input $a = (1, 1, 1)$, and on this input we also have $f^{\triangledown}(1, 1, 1) \geqslant f(1, 1, 0) = 1$. So, by Theorem 3, the circuit $F_+ = xyz \vee y$ computes $f^{\triangledown}$.

Remark 4. In the case of 1-hazards (instead of 0-hazards), *neither* of the implications (A) $\Rightarrow$ (B) and (B) $\Rightarrow$ (A) holds. To show (A) $\not\Rightarrow$ (B), consider the circuit $F = (x \vee \bar{z})(y \vee z)$ computing $f = xz \vee y\bar{z}$. Since the circuit F produces all three prime implicants of f (see Remark 8), Theorem 1 implies that F has no 1-hazards. But on the input $(0, 0, 1)$, the monotone version $F_+ = y \vee z$ of F outputs $F_+(0, 0, 1) = 1$ whereas $f^{\triangledown}(0, 0, 1) = f(0, 0, 0) \vee f(0, 0, 1) = 0$. To show (B) $\not\Rightarrow$ (A), consider the circuit $F = xz \vee y\bar{z}$ for the same function $f = xz \vee y\bar{z}$. Since this circuit produces no zero terms at all, the assertion (C) and, hence, also (B) holds for this circuit. That is, $F_+ = f^{\triangledown}$ holds. But the circuit F has a 1-hazard at $\alpha = (1, 1, \mathfrak{u})$: $f(1, 1, 1) = f(1, 1, 0) = 1$ but $F(1, 1, \mathfrak{u}) = \mathfrak{u} \vee \bar{\mathfrak{u}} = \mathfrak{u}$.

5. Proof of Theorem 1

Theorem 1 is a special case of two more general theorems Theorems 4 and 5 we prove in this section.

Following Mukaidono [18], let us equip $\{0, \mathfrak{u}, 1\}$ with a partial order $\preccurlyeq$ such that $\mathfrak{u}$ is the least element and 0 and 1 are incomparable elements greater than $\mathfrak{u}$. This order obviously extends to the vectors in $\{0, \mathfrak{u}, 1\}^n$ by letting $\alpha \preccurlyeq \beta$ iff $\alpha_i \preccurlyeq \beta_i$ for all positions i. Thus, $\alpha \preccurlyeq \beta$ means that the vector β is obtained from α by replacing some unstable bits $\mathfrak{u}$ by stable bits 0 or 1. In particular, the subcube A_α defined by a vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$ is $A_\alpha = \{a \in \{0, 1\}^n : \alpha \preccurlyeq a\}$.

Since the gates AND, OR and NOT are monotone with respect to $\preccurlyeq$, the function $F : \{0, \mathfrak{u}, 1\}^n \to \{0, \mathfrak{u}, 1\}$ computed by a DeMorgan circuit F must be monotone with respect to $\preccurlyeq$. In particular, if $\alpha \preccurlyeq \beta$ and $F(\beta) = \mathfrak{u}$, then also $F(\alpha) = \mathfrak{u}$. That is, replacing stable bits 0/1 by the unstable bit $\mathfrak{u}$ in the input vector cannot change the unstable output $\mathfrak{u}$ of the circuit to a stable output 0 or 1.

We say that a ternary vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$ is a 1-*witness* of a Boolean function f if $f(A_\alpha) = \{1\}$. Such a vector is a *prime* 1-*witness* of f if $f(A_\beta) \neq \{1\}$ for every vector $\beta \preccurlyeq \alpha$, $\beta \neq \alpha$. Dually, a vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$ is a 0-*witness* of a Boolean function f if $f(A_\alpha) = \{0\}$. Such a vector is a *prime* 0-*witness* of f if $f(A_\beta) \neq \{0\}$ for every vector $\beta \preccurlyeq \alpha$, $\beta \neq \alpha$.

In other words, a vector α is a 1 witness (0-witness) of f if already the assignment of the stable bits of α to the variables forces $f = 1$ (resp. $f = 0$) *regardless* of the 0/1 values given to the remaining variables (where vector α has $\mathfrak{u}$'s). Prime witnesses are minimal under $\preccurlyeq$ vectors with these properties, i.e. witnesses that are no longer witnesses if any one stable $0/1$ entry is replaced by the unstable entry $\mathfrak{u}$. For example, $(1,1,\mathfrak{u})$ is a prime 1-witness, and $(0,0,\mathfrak{u})$ a prime 0-witness of the function $f(x,y,z) = xz \vee y\overline{z}$.

Remark 5. There is an obvious correspondence between witness-vectors and implicants (implicates). Namely, a vector $\alpha \in \{0,\mathfrak{u},1\}^n$ is a (prime) 1-witness of f iff the term $p = \bigwedge_{i\colon \alpha_i \neq \mathfrak{u}} x_i^{\alpha_i}$ is a (prime) implicant of f; note that $p(\alpha) = 1$ and, hence, also $p(A_\alpha) = \{1\}$. Dually, α is a (prime) 0-witness of f iff the clause $q = \bigvee_{i\colon \alpha_i \neq \mathfrak{u}} x_i^{\overline{\alpha_i}}$ is a (prime) implicate of f; note that $q(\alpha) = 0$ and, hence, also $q(A_\alpha) = \{0\}$.

Recall that a term t (and AND of literals) is a *zero term* if it contains at least one complementary literal, that is, a literal z whose negation $\overline{z}$ also occurs in t. Similarly, clauses q (ORs of literals) containing at least one complementary literal are *trivial clauses*. Note that, over the binary domain, such terms and clauses can take only one value: $t(a) = 0$ and $q(a) = 1$ for all $a \in \{0,1\}^n$. This is no longer the case over the ternary domain $\{0,\mathfrak{u},1\}^n$.

Remark 6. If t is a zero term and q is a trivial clause, then for every $\alpha \in \{0,\mathfrak{u},1\}^n$, we have $t(\alpha) \in \{0,\mathfrak{u}\}$ and $q(\alpha) \in \{\mathfrak{u},1\}$. Namely, $t(\alpha) = 0$ if $z(\alpha) = 0$ for some literal z of t, and $t(\alpha) = \mathfrak{u}$ otherwise: in this later case, all complementary literals of t must be set to $\mathfrak{u}$ by α. Similarly, $q(\alpha) = 1$ if $z(\alpha) = 1$ for some literal z of q, and $t(\alpha) = \mathfrak{u}$ otherwise: in this later case, all complementary literals of q must be set to $\mathfrak{u}$ by α.

Theorem 4 (1-hazards). *Let F be a DeMorgan circuit computing a Boolean function f, D be the formal DNF and C the formal CNF of F. The following assertions are equivalent.*

(a) *F has a 1-hazard.*
(b) *There is a prime 1-witness α of f such that $F(\alpha) = \mathfrak{u}$.*
(c) *There is a prime 1-witness α of f such that $q(\alpha) = \mathfrak{u}$ holds for some trivial clause q of C.*
(d) *There is a prime 1-witness α of f such that $t(\alpha) \neq 1$ holds for every term t of D.*
(e) *Some prime implicant of f is not a term of D.*

In particular, by (c), 1-hazards can only occur in circuits producing trivial clauses.

Proof. **(a) ⇔ (b)**: The implication (b) ⇒ (a) holds just because $F(A_\alpha) = \{1\}$ holds for every 1-witness α for f. So, $F(\alpha) = \mathfrak{u}$ means that there is a 1-hazard at α. To show the converse implication (a) ⇒ (b), suppose that F has a 1-hazard at some vector $\beta \in \{0,\mathfrak{u},1\}^n$; hence, $F(A_\beta) = \{1\}$ and $F(\beta) = \mathfrak{u}$. Since $f(A_\beta) = F(A_\beta) = \{1\}$, vector β is a 1-witness of f. So, $\alpha \preccurlyeq \beta$ holds for some prime 1-witness α of f. From $F(\beta) = \mathfrak{u}$ and $\alpha \preccurlyeq \beta$, we have $F(\alpha) = \mathfrak{u}$.

(b) ⇔ (c): Let $\alpha \in \{0,\mathfrak{u},1\}^n$ be a prime 1-witness of f; hence, $F(A_\alpha) = \{1\}$, implying that $q(A_\alpha) = \{1\}$ and, hence, also $q(\alpha) = 1$ must hold for every nontrivial clause q of C: held $q(\alpha) \in \{0,\mathfrak{u}\}$, then $q(a) = 0$ would hold for a particular refinement $a \in A_\alpha$. So, if $F(\alpha) = \mathfrak{u}$, then $q(\alpha) = \mathfrak{u}$ must hold for some trivial clause q of C. This shows the implication (b) ⇒ (c). To show the converse implication (c) ⇒ (b), suppose that $q(\alpha) = \mathfrak{u}$ holds for some trivial clause q of C. For every other clause q' of C, we have $q'(\alpha) = 1$ if the clause q' is nontrivial, or $q'(\alpha) \in \{\mathfrak{u},1\}$ if the clause q' is trivial (see Remark 6). So, since $q(\alpha) = \mathfrak{u}$ holds for our (trivial) clause q, and since $1 \wedge \mathfrak{u} = \mathfrak{u}$, we have $F(\alpha) = C(\alpha) = \mathfrak{u}$.

(b) ⇔ (d): Let $\alpha \in \{0,\mathfrak{u},1\}^n$ be a (prime) 1-witness of f; hence, $F(A_\alpha) = \{1\}$. If $F(\alpha) = \mathfrak{u}$, then $t(\alpha) \neq 1$ must hold for every term t of D just because $1 \vee \mathfrak{u} = 1 \neq \mathfrak{u}$. This shows the implication (b) ⇒ (d). To show the converse implication (d) ⇒ (b), suppose that $t(\alpha) \neq 1$, that is, $t(\alpha) \in \{0,\mathfrak{u}\}$ holds for every term t of D. Held $t(\alpha) = 0$, that is, $t(A_\alpha) = \{0\}$ for all terms t of D, then we would have

$F(A_\alpha) = \{0\}$, a contradiction with $F(A_\alpha) = \{1\}$. Thus, $t(\alpha) = \mathfrak{u}$ must hold for at least one term t of D and, hence, $F(\alpha) = \mathfrak{u}$.

(d) ⇔ (e): To show the implication (d) ⇒ (e), suppose that there is a prime 1-witness $\alpha \in \{0, \mathfrak{u}, 1\}^n$ of f such that $t(\alpha) \neq 1$ holds for every term t of D. Since α is a prime 1-witness of f, the term $p = \bigwedge_{i\colon \alpha_i \neq \mathfrak{u}} x_i^{\alpha_i}$ is a prime implicant of f. But since $p(\alpha) = 1$, this prime implicant is not among the terms of D.

To show the converse implication (e) ⇒ (d), suppose that some prime implicant $p = \bigwedge_{i \in S} x_i^{c_i}$ of f is *not* a term of D. Since p if a prime implicant of f, the ternary vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$ with $\alpha_i = c_i$ for $i \in S$ and $\alpha_i = \mathfrak{u}$ for $i \notin S$ is a prime 1-witness of f. Take now an arbitrary term t of D. If t is a zero term, then we clearly have $t(\alpha) \neq 1$ (see Remark 6). So let t be a nonzero term of D, and suppose contrariwise that $t(\alpha) = 1$, that is, $t(A_\alpha) = \{1\}$ holds. Since $A_\alpha = p^{-1}(1)$, we then have $p^{-1}(1) = A_\alpha \subseteq t^{-1}(1)$ which can only hold if t is a subterm of p. Since t is an implicant of f and p is a *prime* implicant of f, this is only possible if $t = p$, a contradiction with p *not* being a term of D.

(c) ⇔ (d): This follows from the already proven equivalences (b) ⇔ (c) and (b) ⇔ (d). But for a "sanity check" and to see how the DNF D and the CNF C are related in the case of 1-hazards, let us show the equivalence (c) ⇔ (d) *directly*. So, let $\alpha \in \{0, \mathfrak{u}, 1\}^n$ be a prime 1-witness of f. In particular, then $f(A_\alpha) = \{1\}$ holds. To show (c) ⇒ (d), suppose that $q(\alpha) = \mathfrak{u}$ holds for some (not necessarily trivial) clause q of C. The clause q picks one literal from each term of D. Held $t(\alpha) = 1$ for at least one term t of D, then we would have $q(\alpha) = 1 \neq \mathfrak{u}$. So, $t(\alpha) \in \{0, \mathfrak{u}\}$ must hold for every term t of D.

To show the converse implication (d) ⇒ (c), suppose that $t(\alpha) \in \{0, \mathfrak{u}\}$ holds for every term t of D. Thus, for every term t there are only two possibilities: either (1) $z(\alpha) = 0$ for some literal $z \in t$, or (2) $z(\alpha) = \mathfrak{u}$ for some literal $z \in t$ and $z'(\alpha) \in \{\mathfrak{u}, 1\}$ for all literals $z' \in t$. Case (1) cannot hold for *all* clauses, since then we could form a clause q by picking from each term one literal evaluated to 0 by α. Then $q(\alpha) = 0$ and, hence, $f(A_\alpha) = \{0\} \neq \{1\}$. So, case (2) must happen for at least one term t_0, and we can form a clause q of the CNF C as follows: pick a literal $z \in t_0$ with $z(\alpha) = \mathfrak{u}$ from the term t_0, and pick from each other term t' any literal z' with $z'(\alpha) \in \{0, \mathfrak{u}\}$. Then $q(\alpha) = \mathfrak{u}$, and it remains to show that q must be a trivial clause. Suppose the opposite. Then, by replacing the instable bits $\mathfrak{u}$ of α with appropriate constants 0 or 1, we can obtain a refinement $a \in A_\alpha$ of α for which $q(a) = 0$ holds. But then also $f(a) = 0$, a contradiction with $f(A_\alpha) = \{1\}$. □

Remark 7. Let D be the formal DNF of a DeMorgan circuit F, and $\alpha \in \{0, \mathfrak{u}, 1\}^n$. If $F(A_\alpha) = \{1\}$, that is, if F accepts all vectors of the subcube A_α, then we only know that every vector of A_α must be accepted by some t term of D, but for different vectors, these terms may be different. If, however, the circuit F has no 1-hazards, then Theorem 4(d) gives us a much stronger property: $t(A_\alpha) = \{1\}$ must hold for at least one term t, that is, a *single* term of D must accept *all* vectors of the subcube A_α.

Remark 8. The equivalence (a) ⇔ (e) of Theorem 4 gives the first claim of Theorem 1: a DeMorgan circuit has no 1-hazards if and only if it produces all prime implicants of the computed Boolean function. Note, however, that in the case of 0-hazards, this claim of Theorem 1 holds in *neither* of the two directions. To show ($\not\Rightarrow$), consider the circuit $F = x\overline{y} \vee y$ for the function $f = x \vee y$. Since the circuit produces no zero terms, it has no 0-hazards (Theorem 5). But the prime implicant $p = x$ of f is not produced by F. To show ($\not\Leftarrow$), consider the circuit $F = (x \vee \overline{z})(y \vee z)$ computing $f = xz \vee y\overline{z}$. The formal DNF $D = xy \vee xz \vee y\overline{z} \vee z\overline{z}$ of the circuit F contains all three prime implicants xy, xz and $y\overline{z}$ of f, but F still has a 0-hazard at input $\alpha = (0, 0, \mathfrak{u})$: $f(A_\alpha) = \{0\}$ but $F(\alpha) = z\overline{z}(\alpha) = \mathfrak{u} \wedge \overline{\mathfrak{u}} = \mathfrak{u}$.

Theorem 5 (0-hazards)**.** *Let F be a DeMorgan circuit computing a Boolean function f, D be the formal DNF and C the formal CNF of F. The following assertions are equivalent.*

(a) *F has a 0-hazard.*
(b) *There is a prime 0-witness α of f such that $F(\alpha) = \mathfrak{u}$.*
(c) *There is a prime 0-witness α of f such that $t(\alpha) = \mathfrak{u}$ holds for some zero term t of D.*
(d) *There is a prime 0-witness α of f such that $q(\alpha) \neq 0$ holds for every clause q of C.*
(e) *Some prime implicate of f is not a clause of C.*

In particular, by (c), 0-hazards can only occur in circuits producing zero terms. The proof of Theorem 5 is just "dual" to the proof of Theorem 4, but we include it only for completeness.

Proof. **(a) ⇔ (b)**: The implication (b) ⇒ (a) holds just because $F(A_\alpha) = \{0\}$ holds for every 0-witness α for f. So, $F(\alpha) = \mathfrak{u}$ means that there is a 0-hazard at α. To show the converse implication (a) ⇒ (b), suppose that F has a 0-hazard at some vector $\beta \in \{0, \mathfrak{u}, 1\}^n$; hence, $F(A_\beta) = \{0\}$ and $F(\beta) = \mathfrak{u}$. Since $F(A_\beta) = \{0\}$, vector β is a 0-witness of f. So, $\alpha \preccurlyeq \beta$ holds for some prime 0-witness α of f. Then $F(\alpha) = \mathfrak{u}$ because $F(\beta) = \mathfrak{u}$ and $\alpha \preccurlyeq \beta$.

(b) ⇔ (c): Let $\alpha \in \{0, \mathfrak{u}, 1\}^n$ be a prime 0-witness of f; hence, $F(A_\alpha) = \{0\}$, implying that $t(A_\alpha) = \{0\}$ and, hence $t(\alpha) = 0$ for every nonzero term t of D: held $t(\alpha) \in \{\mathfrak{u}, 1\}$, then $t(a) = 1$ would hold for a particular refinement $a \in A_\alpha$. So, if $F(\alpha) = \mathfrak{u}$, then $t(\alpha) = \mathfrak{u}$ must hold for some zero term t of D. This shows the implication (b) ⇒ (c). To show the converse implication (c) ⇒ (b), suppose that $t(\alpha) = \mathfrak{u}$ holds for some zero term t of D. For every other term t' of D, we have $t'(\alpha) = 0$ if t' is nonzero, or $t'(\alpha) \in \{0, \mathfrak{u}\}$ if t' is a zero term (see Remark 6). So, since $t(\alpha) = \mathfrak{u}$ holds for our (zero) term t, and since $0 \vee \mathfrak{u} = \mathfrak{u}$, we have $F(\alpha) = D(\alpha) = \mathfrak{u}$.

(b) ⇔ (d): Let $\alpha \in \{0, \mathfrak{u}, 1\}^n$ be a (prime) 0-witness of f; hence $F(A_\alpha) = \{0\}$. If $F(\alpha) = \mathfrak{u}$, then $q(\alpha) \neq 0$ must hold for every clause q of the CNF C just because $0 \wedge \mathfrak{u} = 0 \neq \mathfrak{u}$. This shows the implication (b) ⇒ (d). To show the converse implication (d) ⇒ (b), suppose that $q(\alpha) \neq 0$, that is, $q(\alpha) \in \{\mathfrak{u}, 1\}$ holds for every clause q of C. Held $q(\alpha) = 1$, that is, $q(A_\alpha) = \{1\}$ for all clauses q of C, then we would have $F(A_\alpha) = \{1\}$, a contradiction with $F(A_\alpha) = \{0\}$. Thus, $q(\alpha) = \mathfrak{u}$ must hold for at least clause q of C and, hence, $F(\alpha) = \mathfrak{u}$.

(d) ⇔ (e): To show the implication (d) ⇒ (e), suppose that there is a prime 0-witness $\alpha \in \{0, \mathfrak{u}, 1\}^n$ of f such that $q(\alpha) \neq 0$ holds for every clause q of C. Since α is a prime 0-witness of f, the clause $q = \bigvee_{i:\, \alpha_i \neq \mathfrak{u}} x_i^{\overline{\alpha}_i}$ is a prime implicate of f. But since $q(\alpha) = 0$, this prime implicate is not among the clauses of D. To show the converse implication (e) ⇒ (d), suppose that some prime implicate $q = \bigvee_{i \in S} x_i^{c_i}$ of f is *not* a clause of C. Since q if a prime implicate of f, the ternary vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$ with $\alpha_i = \overline{c}_i$ for $i \in S$ and $\alpha_i = \mathfrak{u}$ for $i \notin S$ is a prime 0-witness of f. Take now an arbitrary clause r of C. If r is a trivial clause (contains a literal together with its negation), then we clearly have $r(\alpha) \in \{\mathfrak{u}, 1\}$ (see Remark 6). So let r be a nontrivial clause of C, and suppose contrariwise that $r(\alpha) = 0$, that is, $r(A_\alpha) = \{0\}$ holds. Since $A_\alpha = q^{-1}(0)$, we then have $q^{-1}(0) = A_\alpha \subseteq r^{-1}(0)$ which can only hold if r is a sub-clause of q. Since r is an implicate of f and q is a *prime* implicate of f, this is only possible if $r = q$, a contradiction with q *not* being a clause of C.

(c) ⇔ (d): This follows from the already proven equivalences (b) ⇔ (c) and (b) ⇔ (d). But for a "sanity check" and to see how the DNF D and the CNF C are related in the case of 0-hazards, let us show the equivalence (c) ⇔ (d) *directly*. So, let $\alpha \in \{0, \mathfrak{u}, 1\}^n$ be a prime 0-witness of f. In particular, then $f(A_\alpha) = \{0\}$ holds. To show (c) ⇒ (d), suppose that $t(\alpha) = \mathfrak{u}$ holds for some (not necessarily zero) term t of D. The term t picks one literal from each clause of C. Held $q(\alpha) = 0$ for at least one cause q of C, then we would have $t(\alpha) = 0 \neq \mathfrak{u}$. So, $q(\alpha) \in \{\mathfrak{u}, 1\}$ must hold for every clause q of C.

To show the converse implication (d) ⇒ (c), suppose that $q(\alpha) \in \{\mathfrak{u}, 1\}$ holds for every clause q of the CNF C. Thus, for every clause q there are only two possibilities: either (1) $z(\alpha) = 1$ for some literal $z \in q$, or (2) $z(\alpha) = \mathfrak{u}$ for some literal $z \in q$ and $z'(\alpha) \in \{0, \mathfrak{u}\}$ for all literals $z' \in q$. Case (1) cannot hold for *all* clauses, since then we could form a term t by picking from each clause one literal evaluated to 1 by α. Then $t(\alpha) = 1$ and, hence, $f(A_\alpha) = \{1\} \neq \{0\}$. So, case (2) must happen for at least one clause q_0, and we can form a term t of the DNF D as follows: pick a literal $z \in q_0$ with $z(\alpha) = \mathfrak{u}$ from the clause q_0, and pick from each other clause q' any literal z' with $z'(\alpha) \in \{\mathfrak{u}, 1\}$. Then $t(\alpha) = \mathfrak{u}$, and it remains to show that t must be a zero term. Suppose the opposite. Then, by replacing the instable bits $\mathfrak{u}$ of α with appropriate constants 0 or 1, we can obtain a refinement $a \in A_\alpha$ of α for which $t(a) = 1$ holds. But then also $f(a) = 1$, a contradiction with $f(A_\alpha) = \{0\}$. ◻

6. Complexity gaps

Let f_n be a (non-monotone) Boolean function of n^2 variables, whose inputs are $n \times n$ matrices $x = (x_{i,j})$, and $f_n(x) = 1$ if and only if x is permutation matrix, that is, every row and every column of x has exactly one 1.

Lemma 1. *$L(f_n) = O(n^3)$ but $L_{\mathfrak{u}}(f_n) = n^{\Omega(\log n)}$.*

Proof. The *logical permanent* function per_n accepts a matrix x iff $f_n(z) = 1$ holds for at least one matrix $z \leqslant x$. Hence, $\mathrm{per}_n = f_n^{\triangledown}$ is the downwards closure of f_n. A well-known result of Razborov [20] shows $L_{\mathrm{m}}(\mathrm{per}_n) = n^{\Omega(\log n)}$, and Theorem 3 yields $L_{\mathfrak{u}}(f_n) \geqslant L_{\mathrm{m}}(\mathrm{per}_n) = n^{\Omega(\log n)}$. To show the upper bound $L(f_n) = O(n^3)$, just associate with every entry (i, j) the terms $R_{i,j}$ and $C_{i,j}$, where $R_{i,j}$ is the AND of $x_{i,j}$ and all $\overline{x}_{i,k}$ for $k = 1, \ldots, j-1, j+1, \ldots, n$, and $C_{i,j}$ is the AND of $x_{i,j}$ and all $\overline{x}_{l,j}$ for $l = 1, \ldots, i-1, i+1, \ldots, n$. Consider the DeMorgan circuit $F = F_1 \wedge F_2$, where $F_1 = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} R_{i,j}$ and $F_2 = \bigwedge_{j=1}^{n} \bigvee_{i=1}^{n} C_{i,j}$. Note that $R_{i,j}(x) = 1$ iff the ith row of x has exactly one 1 in the jth column, and $C_{i,j}(x) = 1$ iff the jth column of x has exactly one 1 in the ith row. Hence, $F_1(x) = 1$ iff every row of x has exactly one 1, and $F_2(x) = 1$ iff every column of x has exactly one 1, meaning that the circuit F (which, actually, is a *formula*) computes f_n. ◻

Remark 9. Actually, the unrestricted circuit complexity of f_n is $L(f_n) = O(n^2)$, i.e., is linear in the number n^2 of variables of f_n: every symmetric Boolean function of n variables can be computed by a DeMorgan circuit using $O(n)$ gates (see, e.g. [24, Chapter 3.4]). The Boolean function detecting whether an input vector has exactly one 1 is clearly symmetric. Note, however, that the only slightly worse $O(n^3)$ bound is already achieved by a *trivial* circuit. This (triviality of the upper bound) is the main message of Lemma 1.

Remark 10. If one is satisfied with having a bit less trivial circuit than that for f_n, then one can actually increase the gap from super-polynomial to exponential. A *k-clique* in a complete graph K_n on n vertices consists of a complete graph on some k vertices and $n-k$ isolated vertices. The *exact k-clique function* g_n has $m = \binom{n}{2}$ variables corresponding the edges of K_n, and $g_n(x) = 1$ iff the graph encoded by x is a k-clique. A subgraph of K_n is a k-clique iff it has exactly k vertices of degree $k-1$ and $k(k-1)/2$ edges in total. Every Boolean function, which accepts an input vectors iff it has exactly a given number of 1s, is clearly symmetric. So, we have $L(g_n) = O(m) = O(n^2)$. On the other hand, the downwards closure $g_n^{\triangledown}$ of g_n is the well-known k-clique function k-CLIQUE. So, if we take $k = \lfloor (n/\log n)^{1/3} \rfloor$, then Theorem 3 and the monotone circuit lower bound of Alon and Boppana [1] for k-CLIQUE, yield $L_{\mathfrak{u}}(g_n) \geqslant L_{\mathrm{m}}(g_n^{\triangledown}) = \exp\left((n/\log n)^{1/3}\right)$.

Remark 11. The depth of our (trivial) DeMorgan circuit for f_n is only $O(\log n)$. On the other hand, as shown by Raz and Wigderson [19, Theorem 4.2], every monotone circuit computing per_n has depth $\Omega(n)$. Since $f_n^{\triangledown} = \mathrm{per}_n$, Theorem 3 implies that every hazard-free DeMorgan circuit computing f_n must have depth $\Omega(n)$. Thus, the function f_n also shows that the tradeoff between the depths of hazard-free and general DeMorgan circuits can be even exponential.

7. Final remarks

Recall that $L_{\mathrm{pz}}(f)$ is the minimum number of gates in a DeMorgan circuit F computing the Boolean function f such that F produces all prime implicants and produces no zero term. Let $L_{\mathfrak{u}}(n)$ be the Shannon function for hazard-free circuits, that is, the maximum of $L_{\mathfrak{u}}(f)$ over all Boolean functions f of n variables. We know that $L_{\mathfrak{u}}(f) \geqslant L_{\mathrm{m}}(f^{\triangledown})$ and $L_{\mathrm{pz}}(f) \geqslant L_{\mathfrak{u}}(f)$ holds for any Boolean function f (see Eq. (1)). The following natural open problems remain:

1. How large can the gap $L_{\mathfrak{u}}(f)/L_{\mathrm{m}}(f^{\triangledown})$ be?
2. How large can the gap $L_{\mathrm{pz}}(f)/L_{\mathfrak{u}}(f)$ be?
3. What is the asymptotic value of $L_{\mathfrak{u}}(n)$?

To show $L(f) = O(2^n/n)$ for ay Boolean function f of n variables, Shannon [22] used the decomposition $F = \overline{x}_n \cdot F_0 \vee x_n \cdot F_1$ with $F_b := F(x_1, \ldots, x_{n-1}, b)$ for $b \in \{0,1\}$ to show that all Boolean functions of m variables can be simultaneously computed by a circuit All_m with $2 \cdot 2^{2^m}$ gates (just two gates per function). Then he used the decomposition to fix the last $n - m$ variables of f, and constructs the circuit for f by taking the corresponding outputs of the circuit All_m. Taking m about $\log(n - \log n)$ yields $L(f) \leqslant (4 + o(1)) \cdot 2^n/n$ (see the exposition [21]).

Nitin Saurabh (personal communication) observed that a slight modification of Shannon's construction [22] (as exposed in [21]) yields $L_{\mathfrak{u}}(n) \leqslant 8 \cdot 2^n/n$ also for hazard-free circuits. Since the resulting from Shannon's decomposition circuit has no zero terms, it has no 0-hazards (Theorem 2), but (by Theorem 1) it may have 1-hazards: all produced terms by a Shannon circuit for f have length n, while some prime implicants of f may be shorter. To avoid also 1-hazards, Nitin Saurabh suggested to use an extended decomposition

$$F = \overline{x}_n \cdot F_0 \vee x_n \cdot F_1 \vee F_2\,, \tag{2}$$

where $F_2 = F_0 \cdot F_1$. Because of this additional subcircuit F_2, the circuit for f constructed using this recursion will already produce all prime implicants of f as terms. Hence, by Theorem 1, the circuit F will have no 1-hazards. Let us show that they will have no 0-hazards either.

Note that the latter claim is *not* obvious: the additional subcircuit $F_2 = F_0 \cdot F_1$ *may* produce zero terms, even if neither of the subcircuits F_0 nor F_1 does this; and zero terms *may* lead to 0-hazards (see Theorem 5). Still, we have the following

Lemma 2. *Circuits constructed using the decomposition Eq. (2) have no 0-hazards.*

Proof. Argue by induction on the number n of variables. The basis case $n = 1$ is trivial: $F(x_1) = \overline{x}_1 \cdot F(0) \vee x_1 \cdot F(1) \vee F(0) \cdot F(1)$ clearly has no hazards.

Induction Step. *Let $F = \overline{x}_n \cdot F_0 \vee x_n \cdot F_1 \vee F_2$, where $F_2 = F_0 \cdot F_1$. If the subcircuits F_0 and F_1 have no 0-hazards, then also the circuit F has no 0-hazards.*

To prove this, suppose that F has a 0-hazard at some $\alpha \in \{0, \mathfrak{u}, 1\}^n$. Hence, $F(A_\alpha) = \{0\}$ but $F(\alpha) = \mathfrak{u}$; recall that $A_\alpha \subseteq \{0,1\}^n$ is the set of all refinements of α. Since, $1 \vee \mathfrak{u} = 1$, we know that $F_i(\alpha) \in \{0, \mathfrak{u}\}$ for all $i = 1, 2, 3$. Our goal is to show that then at least one of the sucircuits F_0 and F_1 must have a 0-hazard.

Case 1: $x_n(\alpha) \in \{0,1\}$, say, $x_n(\alpha) = 0$. Then $F(\alpha) = F_0(\alpha) \vee F_2(\alpha) = \mathfrak{u}$ and, hence, $F_0(\alpha) = \mathfrak{u}$. Since $\overline{x}_n \cdot F_0(A_\alpha) = \{0\}$, and since $x_n(\alpha) = 0$, we also have $F_0(A_\alpha) = \{0\}$. Hence, the subcircuit F_0 has a 0-hazard at α.

Case 2: $x_n(\alpha) = \mathfrak{u}$. From $F(A_\alpha) = \{0\}$ we then have $\overline{x}_n \cdot F_0(A_\alpha) = \{0\}$ and $x_n \cdot F_1(A_\alpha) = \{0\}$. Since $x_n(\alpha) = \mathfrak{u}$, this must hold when $\overline{x}_n = 1$ as well as when $x_n = 1$. So, $F_0(A_\alpha) = \{0\}$ and $F_1(A_\alpha) = \{0\}$ must hold.

If $F_2(\alpha) = \mathfrak{u}$, then $F_0(\alpha) = \mathfrak{u}$ or $F_1(\alpha) = \mathfrak{u}$, that is, at least one of the subcircuits F_0 or F_1 has a 0-hazard at α. So, suppose that $F_2(\alpha) \neq \mathfrak{u}$. Then $F_2(\alpha) = 0$, for $F_2(\alpha) = 1$ would yield $F(\alpha) = 1 \neq \mathfrak{u}$. Both $F_0(\alpha) = 0$ and $F_1(\alpha) = 0$ cannot hold, because then we would have $F(\alpha) = 0 \neq \mathfrak{u}$. Hence, $F_i(\alpha) = \mathfrak{u}$ must hold for some $i \in \{0, 1\}$. Since $F_i(A_\alpha) = \{0\}$, the subcircuit F_i has a 0-hazard at α. $\square$

So, thank to the refinement of Eq. (2) by Nitin Saurabh, the order of magnitude is already known: $L_{\mathfrak{u}}(n) = \Theta(2^n/n)$. It remains, however, open whether $L_{\mathfrak{u}}(n) \sim 2^n/n$ holds. One possibility here could be to try to adopt Lupanov's construction [16] leading to $L(n) \leqslant (1 + o(1))2^n/n$.

Acknowledgment. I am thankful to Igor Sergeev for very interesting general discussions, and to Nitin Saurabh for sharing his interestingd observation concerning the Shannon function $L_{\mathfrak{u}}(n)$ and for allowing to mention it here.

References

[1] N. Alon and R. Boppana. The monotone circuit complexity of boolean functions. *Combinatorica*, 7(1):1–22, 1987.

[2] W. Baur and V. Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983.

[3] J. G. Bredeson and P. T. Hulona. Elimination of static and dynamic hazards formultiple input changes in combinational switching circuits. *Information and Control*, 20:114–124, 1972.

[4] S. H. Cadwell. *Switching Circuits and Logical Design.* John Willey & Sons, 1958.

[5] Y. Crama and P. L. Hammer, editors. *Boolean Functions: Theory, Algorithms, and Applications*, volume 142 of *Encyclopedia of Mathematics and Its Applications.* Cambridge University Pess, 2011.

[6] E. Eichelberger. Hazard detection in combinational and sequential switching circuits. *IBM J. Res. Develop.*, 9:90–99, 1965.

[7] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

[8] J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

[9] W. Hu, J. Oberg, A. Irturk, M. Tiwari, T. Sherwood, D. Mu, and R. Kastner. On the complexity of generating gate level information flow tracking logic. *IEEE Trans. Info. Forensics Secur.*, 7(3):1067–1080, 2012.

[10] D. A. Huffman. The design and use of hazard-free switching networks. *J. ACM*, 4(1):47–62, 1957.

[11] C. Ikenmeyer, B. Komarath, C. Lenzen, V. Lysikov, A. Mokhov, and K. Sreenivasaiah. On the complexity of hazard-free circuits. *J. ACM*, 66(4):Article 25, 2019.

[12] S. C. Kleene. *Introduction to Metamathematics.* North Holland, 1952.

[13] N. B. Komarath and N. Saurabh. On the complexity of detecting hazards. *Inf. Process. Letters*, 2020.

[14] S. Körner. *Experience and Theory: An Essay in the Philosophy of Science.* Routledge & Kegan Paul, London, 1966.

[15] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[16] O. B. Lupanov. A method of circuit synthesis. *Izvesitya VUZ, Radiofiz*, 1:120–140, 1958. (in Russian).

[17] G. Malinowski. Kleene logic and inference. *Bull. Section Logic*, 43(1/2):42–52, 2014.

[18] M. Mukaidono. On the B-ternary logical function–A ternary logic considering ambiguity. *Syst. Comput. Controls*, 3(3):27–36, 1972.

[19] R. Raz and A. Wigderson. Monotone circuits for matching require linear depth. *J. ACM*, 39(3):736–744, 1992.

[20] A. A. Razborov. Lower bounds on monotone complexity of the logical permanent. *Math. Notes of the Acad. of Sci. of the USSR*, 37(6):485–493, 1985.

[21] N. Saurabh. Lupanov's upper bound, gate elimination technique, linear lower bounds against circuits, 2019. Lecture Note.

[22] C. E. Shannon. The synthesis of two-terminal switching circuits. *Bell Systems Technical Journal*, 28:59–98, 1949.

[23] É. Tardos. The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica*, 7(4):141–142, 1987.

[24] I. Wegener. *The complexity of Boolean functions.* Wiley-Teubner, 1987.

[25] M. Yoeli and S. Rinon. Application of ternary algebra to the study of static hazards. *J. ACM*, 11(1):84–97, 1964.