Electronic Colloquium on Computational Complexity, 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. Recently, Ikenmeyer et al. [*J. ACM*, 66:4 (2019), Article 25] have 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. We also show that a circuit is hazard-free if and only if the circuit and its dual produce (purely syntactically) all prime implicants of the functions they compute. This extends a classical result of Eichelberger [*IBM J. Res. Develop.*, 9 (1965)] showing this property for depth-two circuits producing no terms containing a variable together with its negation. Finally, we give a very simple non-monotone Boolean function whose hazard-free circuit complexity is super-polynomially larger than its unrestricted circuit complexity.

1. Introduction

The problem of designing hazard-free circuits naturally occurs when implementation circuits in hardware ([3, 9], but is also closely related to questions in logic ([11, 12, 14]) and even in cybersecurity ([8]). The importance of hazard-free circuits is already highlighted in the classical textbook [3].

If not stated otherwise, by a *circuit* we will understand a *DeMorgan circuit*, that is, a Boolean circuit over $\{\wedge, \vee, \neg\}$, 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 occurring on the output of circuits during an input transition, stipulated by physical delays at wires or gates: circuits are usually asynchronous. Having designed a hazard-free circuit for a given Boolean function f, one is sure that no glitches will occur regardless of delays in *any* hardware implementation of this circuit.

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 (due to different delays at wires), that is, for a short moment,

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

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

ISSN 1433-8092

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 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 is, there will be no glitches in *any* hardware implementation of the circuit F', regardless of the delays along the wires.

That every Boolean function f can be computed by a hazard-free DeMorgan circuit was shown by Huffman [9] already in 1957: the OR of all prime implicants of f is hazard-free. In 1965, Eichelberger [5, 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 3).

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. [10] 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

To analyze hazards in DeMorgan circuits, we will use a natural notion of the "formal DNF" of a circuit. The point is that every DeMorgan circuit F not only computes a unique Boolean function, but also *produces* (purely syntactically) a unique set $T(F)$ of terms (ANDs of literals) 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, and 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 $T(F)$ of terms produced by the entire circuit is the set produced at its output gate. The *formal DNF* of a circuit F is the OR of all terms in $T(F)$.

Let us stress that, when forming the set of produced terms, the idempotence laws $x \wedge x = x$ and $x \vee x = x$ as well the absorbtion laws $x \vee xy = x$ can be used, but 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, i.e., may contain a variable and its negation[2]. 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 $z\overline{z}$.

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.

Our first result shows that the formal DNF D of F contains all prime implicants of f as terms precisely when the circuit F has no 1-hazards.

Theorem 1. *A DeMorgan circuit has no* 1*-hazards if and only if it produces all prime implicants of the computed Boolean function.*

In particular, every hazard-free DeMorgan circuit must produce all prime implicants of the Boolean function it computes.

The following lemma formalizes a (not very surprising) intuition that the notions of 0-hazards and 1-hazards are, in fact, dual. Recall that the *dual* of a boolean function $f(x_1, \ldots, x_n)$ is the boolean function $f^*(x_1, \ldots, x_n) = \neg f(\overline{x}_1, \ldots, \overline{x}_n)$. The *dual* F^* of a DeMorgan circuit F is obtained from F by exchanging the operators AND and OR, as well as the constants 0 and 1. It is well known and easy to show (see, e.g., [4, Theorem 1.3]) that F computes f iff F^* computes f^*.

Lemma 1. *A circuit has a* 0*-hazard if and only if its dual circuit has a* 1*-hazard.*

Theorem 1 and Lemma 1 directly yield the following *exact* characterization of hazard-freeness in terms of prime implicants.

Corollary 1. *A DeMorgan circuit computing a Boolean function f is hazard-free if and only if the circuit produces all prime implicants of f and the dual circuit produces all prime implicants of f^*.*

Example 1. Consider the circuit $F = (x \vee \overline{z})(y \vee z)$ computing the Boolean function $f = xz \vee y\overline{z}$. The formal DNF $D = xy \vee xz \vee y\overline{z} \vee z\overline{z}$ of F contains all three prime implicants xy, xz and $y\overline{z}$ of f. But the formal DNF of the dual circuit $F^* = x\overline{z} \vee yz$ does not contain the prime implicant xy of the dual function $f^* = x\overline{z} \vee yz$. Hence, the circuit F is not hazard-free. To give an even simpler (albeit more artificial) example, consider the circuit $F = x \vee y\overline{y}$ computing $f(x, y) = x$. The unique prime implicant x of f is a term of the formal DNF $x \vee y\overline{y}$ of F. But the formal DNF $xy \vee x\overline{y}$ of the dual circuit $F^* = x(y \vee \overline{y})$ does not contain the (also unique) prime implicant x of the dual function $f^* = x$. Hence, the circuit F is not hazard-free.

A DeMorgan circuit is *zero-term free* if it produces no zero terms. Since such circuits cannot have any 0-hazards (see Lemma 4), Theorem 1 yields the classical result of Eichelberger [5, Theorem 2].

[2]This is different from DNFs of Boolean functions f, where zero terms are not present: these terms contribute nothing to the values of f. But presence or absence of zero terms in formal DNFs of circuits is important when dealing with hazards, so we keep them.

Corollary 2 (Eichelberger [5])**.** *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, Corollary 2 directly yields the following fact; [10, Lemma 4.2] gives an alternative proof of this fact by induction on the circuit size.

Corollary 3. *Monotone circuits are hazard-free.*

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 $z \leqslant x$. 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.

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 lemma gives us a general condition for $F_+ = f^\triangledown$ to hold. The *positive factor* of a term is obtained by replacing every its negated literal with constant 1.

Lemma 2. *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 $f^\triangledown$.*

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

Theorem 2. *Let F be a DeMorgan circuit computing a Boolean function f such that $f(\vec{0}) = 0$. If the circuit F has no 0-hazards, then F_+ computes $f^\triangledown$.*

Remark 2. Together with Lemma 2, Theorem 2 gives the following *necessary* condition for a circuit F to have no 0-hazards: it is necessary that the positive factor of every zero term produced by F is an implicant of $f^\triangledown$.

Remark 3. The converse of Theorem 2 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. Also, Theorem 2 does not hold for 1-hazards: even if a circuit F has no 1-hazards, F_+ does not need compute $f^\triangledown$. Counterexamples are given in Remarks 7 and 8. Still, Theorem 2 implies that $F_+ = f^\triangledown$ holds for every hazard-free circuit F: such circuits have neither 0-hazards not 1-hazards.

For a Boolean function f, let $L(f)$ denote the minimum number of gates in a (unrestricted) DeMorgan circuit computing f. Let also $L_{\mathfrak{U}}(f)$, $L_{\rm m}(f)$ and $L_*(f)$ 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

$$L_{\rm m}(f^{\triangledown}) \leqslant L_{\mathfrak{U}}(f) \leqslant L_*(f)\,, \tag{1}$$

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

Let us note that the first inequality in Eq. (1) was already proved in a recent paper by Ikenmeyer et al. [10] using different arguments. Namely, the proof in [10] 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 [1] to compute all partial derivatives of a multivariate polynomial by an arithmetic circuit.

In contrast, our proof of Theorem 2 (in Section 6) is short and amazingly simple. Actually, for monotone Boolean functions f, Theorem 2 tells us a bit more than just the equality $L_{\mathfrak{U}}(f) = L_{\rm m}(f)$: 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 $L_{\mathfrak{U}}(f) \geqslant L_{\rm m}(f^{\triangledown})$, implies that the gap $L_{\mathfrak{U}}(f)/L(f)$ can be super-polynomial and even exponential. This consequence was shown in [10], when f is either the logical permanent [17] or the logical determinant, or the Tardos function [18]. 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: [13, 7] for logical permanent and [6] for the Tardos function. In Section 7, 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 5): 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. A trivial DeMorgan circuit (actually, a formula) shows $L(f_n) = O(n^3)$.

Notation. We use a standard notation concerning Boolean functions and circuits [4, 19]. 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. A term is a *zero term* if it contains a variable together with its negation. A *DNF* (disjunctive normal form) is an OR of terms. 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, $t(a) \leqslant f(a)$ holds for all $a \in \{0,1\}^n$. An implicant t is a *prime implicant* of f if no proper subterm of t is a implicant of f. A Boolean function f is *monotone* if $x \leqslant y$ implies $f(x) \leqslant f(y)$, where $x \leqslant y$ means $x_i \leqslant y_i$ for all positions i.

3. Hazard-free circuits

In this paper, we ignore the electronic aspect of hazards, and stick on their idealized, mathematical model as, for example, in [2, 5, 15, 20]. The classical Kleene's three-valued "strong logic of indeterminacy" [11] 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:

$$x \wedge y = \min(x, y), \;\; x \vee y = \max(x, y) \text{ and } \overline{x} = 1 - x\,.$$

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 [20]. 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 lows $x \wedge \overline{x} = 0$ 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 $\mathfrak{u} \wedge \overline{\mathfrak{u}} = \mathfrak{u} \neq 0$, and $1 \vee \mathfrak{u} = 1$ but $\mathfrak{u} \vee \overline{\mathfrak{u}} = \mathfrak{u} \neq 1$.

For us, it will be important that AND distributes over OR also in the ternary domain $\mathbb{T}$: for all $x, y, z \in \mathbb{T}$, we have $x(y \vee z) = xy \vee xz$. So, since the annihilation lows are *not* used when constructing the formal DNF D of a given circuit F, the ternary function computed by D is the same as that computed by the circuit F, that is, $D(\alpha) = F(\alpha)$ holds for every ternary vector $\alpha \in \mathbb{T}^n$. This 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.

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

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

of all refinements of α. Hence, $|A_\alpha| = 2^d$, where d is the numbers of unstable bits $\mathfrak{u}$ in α. For a Boolean function f and a set $A \subseteq \{0,1\}^n$, let $f(A_\alpha) = \{f(a) : a \in A\} \subseteq \{0,1\}$ 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)**.** Let $b \in \{0,1\}$. A circuit F has an *b-hazard* at $\alpha \in \mathbb{T}^n$ if $F(A_\alpha) = \{b\}$ but still $F(\alpha) = \mathfrak{u}$ holds. 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 computed at individual gates values 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 $f : \mathbb{B}^n \to \mathbb{B}$ computes a (unique) ternary function $F : \mathbb{T}^n \to \mathbb{T}$, the "ternary extension" of f. 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 $\alpha \in T$ 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* [2, 5, 20].

Example 2. 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 $\alpha = (1,1,\mathbb{u})$: although $f(1,1,0) = f(1,1,1) = 1$, we still have $F(1,1,\mathbb{u}) = (1 \wedge \mathbb{u}) \vee (1 \wedge \overline{\mathbb{u}}) = \mathbb{u}$. The circuit $F'' = (x \vee \overline{z})(y \vee z)$ for the same function f has a 0-hazard at $\alpha = (0,0,\mathbb{u})$: although $f(0,0,0) = f(0,0,1) = 0$, we still have $F(0,0,\mathbb{u}) = \overline{\mathbb{u}} \wedge \mathbb{u} = \mathbb{u}$. 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 Lemma 4(a)), and has no 1-hazards because D contains all three prime implicant of f (see Theorem 1).

Remark 4. 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 $\alpha \in \{0,\mathbb{u},1\}^n$ with $\alpha_i = \mathbb{u}$ when $a_i \neq b_i$, and $\alpha_i = a_i$ when $a_i = b_i$. Then $[a,b] = A_\alpha$.

4. Proof of Lemma 1

The complement of a vector $\alpha \in \{0,\mathbb{u},1\}^n$ is the vector $\overline{\alpha} = (\overline{\alpha}_1,\ldots,\overline{\alpha}_n)$ in $\{0,\mathbb{u},1\}^n$. Recall that the *dual* of a boolean function $f(x_1,\ldots,x_n)$ is the boolean function $f^*(x_1,\ldots,x_n) = \neg f(\overline{x}_1,\ldots,\overline{x}_n)$. For example, the dual of a term $t = \bigwedge_{i\in S} x_i^{c_i}$ is the clause $t^* = \bigvee_{i\in S} x_i^{c_i}$. The *dual* F^* of a DeMorgan circuit F is obtained from F by exchanging the operators AND and OR, as well as the constants 0 and 1. For example, the dual of $F = xz \vee y\overline{z}$ is $F^* = (x \vee z)(y \vee \overline{z})$. It is well known and easy to verify that a circuit F computes a Boolean function f iff the dual circuit F^* computes f^* (see, e.g., [4, Theorem 1.3]). Since the proof only uses de Morgan laws, and since these laws hold also over the ternary domain $\{0,\mathbb{u},1\}$, we also have $F^*(\alpha) = \neg F(\overline{\alpha})$ for every ternary vector $\alpha \in \{0,\mathbb{u},1\}^n$.

Lemma 1 is a direct consequence of the following lemma.

Lemma 3. *A circuit has a* 0*-hazard at* $\alpha \in \{0,\mathbb{u},1\}^n$ *if and only if the dual circuit has a* 1*-hazard at* $\overline{\alpha}$.

Proof. Let F be a DeMorgan circuit computing a Boolean function f. The circuit F has a 0-hazard at $\alpha \in \{0,\mathbb{u},1\}^n$ iff $f(A_\alpha) = \{0\}$ and $F(\alpha) = \mathbb{u}$. Since $F^*(\overline{\alpha}) = \neg F(\alpha)$ and $\overline{\mathbb{u}} = \mathbb{u}$, $F(\alpha) = \mathbb{u}$ holds iff $F^*(\overline{\alpha}) = \mathbb{u}$. On the other hand, after an appropriate permutation of the set $\{1,\ldots,n\}$ of positions, the subcube A_α has the form $A_\alpha = \{a\} \times \{0,1\}^k$ for some vector $a \in \{0,1\}^{n-k}$, where k is the number of unstable bits $\mathbb{u}$ in α. Then $A_{\overline{\alpha}} = \{\overline{a}\} \times \{0,1\}^k$. For every vector $(\overline{a},\overline{b}) \in A_{\overline{\alpha}}$, the vector (a,b) belongs to A_α, and we have $f^*(\overline{a},\overline{b}) = \neg f(a,b)$. So, $f(A_\alpha) = \{0\}$ holds iff $f^*(A_\alpha) = \{1\}$. □

Corollary 4. *A DeMorgan circuit is hazard-free if and only if its dual circuit is also hazard-free.*

5. Proof of Theorem 1

To prove Theorem 1, we first prove a technical Lemma 4 showing how hazards expose themselves in the formal DNFs of circuits. Recall that a *zero term* is a term containing a variable together with its negation. Note that for every such term and for every ternary vector $\alpha \in \{0,\mathbb{u},1\}^n$, we always have either $t(\alpha) = 0$ or $t(\alpha) = \mathbb{u}$, but never $t(\alpha) = 1$. For every nonzero term t, $t(\alpha) = 1$ holds precisely when $t(A_\alpha) = \{1\}$, and $t(\alpha) = 0$ holds precisely when $t(A_\alpha) = \{0\}$: in the former case, α sets to 1 all literals of t, whereas in the latter case, α sets to 0 at least one literal of t.

Lemma 4. *Let F be a DeMorgan circuit, D be its formal DNF and $\alpha \in \{0, \mathfrak{u}, 1\}^n$.*

(a) *F has a 0-hazard at α iff $F(A_\alpha) = \{0\}$ but $t(\alpha) = \mathfrak{u}$ for some zero term t of D.*

(b) *F has a 1-hazard at α iff $F(A_\alpha) = \{1\}$ but $t(\alpha) \neq 1$ for all nonzero terms t of D.*

In particular, circuits producing no zero terms cannot have any 0-hazards. Note, however, that even if no zero terms are produced, the circuit may have 1-hazards (see, e.g., Example 2).

Proof. (a) Let $F(A_\alpha) = \{0\}$. Then $t(A_\alpha) = \{0\}$ for every term t of D. Hence, $t(\alpha) = 0$ for every nonzero term t, and $t(\alpha) \in \{0, \mathfrak{u}\}$ for every zero term t. To show the direction ($\Leftarrow$) of (a), suppose that $t(\alpha) = \mathfrak{u}$ for some zero term t of D. Since the values of all other terms on input α lie in $\{0, \mathfrak{u}\}$, we have $F(\alpha) = \mathfrak{u}$. Thus, F has a 0-hazard at α. To show the converse direction ($\Rightarrow$) of (a), suppose that the circuit F has a 0-hazard at α, that is, $F(\alpha) = \mathfrak{u}$ holds. Since $t(\alpha) = 0$ for every nonzero term t, $t(\alpha) = \mathfrak{u} \neq 0$ must hold for at least one zero term.

(b) Let $F(A_\alpha) = \{1\}$. To show the direction ($\Leftarrow$) of (b), suppose that $t(\alpha) \neq 1$, i.e. $t(\alpha) \in \{0, \mathfrak{u}\}$ holds for all nonzero terms t of D. If $t(\alpha) = 0$ for all these terms, then $F(A_\alpha) = \{0\} \neq \{1\}$: on binary vectors, zero terms always output 0. So, $t(\alpha) = \mathfrak{u}$ must hold for at least one nonzero term t. Hence, $F(\alpha) = \mathfrak{u}$, meaning that F has a 1-hazard at α. To show the converse direction ($\Rightarrow$) of (b), suppose that F has a 1-hazard at α. Hence, $F(\alpha) = \mathfrak{u}$. Since $x \vee 1 = 1$ holds for all $x \in \{0, \mathfrak{u}, 1\}$, $D(\alpha) = \mathfrak{u}$ implies that $t(\alpha) \neq 1$ must hold for every term t of D. ∎

Remark 5. 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 Lemma 4(b) 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.

Proof of Theorem 1. Let F be a DeMorgan circuit computing a Boolean function f, and let D be the formal DNF of F. Our goal is to show that F has no 1-hazards if and only if D contains all prime implicants of f as terms.

($\Rightarrow$) Suppose that some prime implicant $p = \bigwedge_{i \in S} x_i^{c_i}$ of f is *not* a term of D, and consider 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 $i \notin S$. Then $A_\alpha = p^{-1}(1) := \{a \in \{0,1\}^n : p(a) = 1\}$. Since p is an implicant of f, we have $f(A_\alpha) = \{1\}$. Suppose contrariwise that the circuit F has no 1-hazard at α. Then, by Lemma 4(b), $t(\alpha) = 1$ and, hence, also $t(A_\alpha) = \{1\}$ must hold for some nonzero term t of D. Thus, $p^{-1}(1) = A_\alpha \subseteq t^{-1}(1)$ which can only hold is 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.

($\Leftarrow$) Suppose that the circuit F *has* a 1-hazard at some $\alpha \in \{0, \mathfrak{u}, 1\}^n$. Our goal is to show that then the DNF D must miss some prime implicant of f. Since F has a 1-hazard at α, we have $f(A_\alpha) = \{1\}$ and $F(\mathfrak{u}) = \mathfrak{u}$. By Lemma 4(b), $t(\alpha) \neq 1$ holds for every nonzero term t of D. The term $t_\alpha = \bigwedge_{i:\, \alpha_i \neq \mathfrak{u}} x_i^{\alpha_i}$ defined by the vector α accepts a vector $a \in \{0,1\}^n$ if and only if $a \in A_\alpha$. Hence, $t_\alpha(A_\alpha) = \{1\}$. Since also $f(A_\alpha) = \{1\}$, t_α is an implicant of f and, hence, contains some prime implicant p of f as a subterm. From $t_\alpha(A_\alpha) = \{1\}$, we have $p(A_\alpha) = \{1\}$ and, hence $p(\alpha) = 1$. Since $t(\alpha) \neq 1$ holds for every nonzero term t of D, the prime implicant p of f cannot be a term of D. ∎

Remark 6. Note that, in the case of 0-hazards, Theorem 1 holds in *neither* of the two directions. To show ($\nRightarrow$), 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 (one can check directly or just use Lemma 4). But the prime implicant $p = x$ of f is not produced by F. To show ($\nLeftarrow$), 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 iput $\alpha = (0, 0, \mathbb{u})$: $f(A_\alpha) = \{0\}$ but $F(\alpha) = z\overline{z}(\alpha) = \mathbb{u} \wedge \overline{\mathbb{u}} = \mathbb{u}$.

6. Proof of Theorem 2 and Lemma 2

We will prove Lemma 2 and Theorem 2 with one simple argument. 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. Consider the following three assertions.

(A) The circuit F has no 0-hazards.
(B) The circuit F_+ computes f^∇.
(C) The positive factor t_+ of every zero term t of D is an implicant of f^∇

Our goal is to prove that (A) $\Rightarrow$ (B) (Theorem 2), and (B) $\Leftrightarrow$ (C) (Lemma 2). 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^∇, then there must be a vector $a \in \{0,1\}^n$ such that $F_+(a) \geqslant t_+(a) = 1$ but $f^\nabla(a) = 0$, that is, then F_+ does not compute f^∇.

To show (A) $\Rightarrow$ (B) and (C) $\Rightarrow$ (B), suppose that $F_+(x)$ does not compute $f^\nabla(x) = \bigvee_{z \leqslant x} f(z)$. Since $F_+ \geqslant f^\nabla$ (see Remark 1), there must be a vector $a \in \{0,1\}^n$ such that $F_+(a) = 1$ but $f^\nabla(a) = 0$. 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. 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 $t(a) = 0$. Also, since $t_+(a) = 1$ but $f^\nabla(a) = 0$, $S \cap T \neq \varnothing$ must hold, that is, the term t must be a zero term. Indeed, otherwise $t(b) = 1$ and, hence, also $f^\nabla(a) = 1$ 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^∇. This shows the implication (C) $\Rightarrow$ (B).

To show the implication (A) $\Rightarrow$ (B), we have only to show that the circuit F contains a 0-hazard at some vector $\alpha \in \{0, \mathbb{u}, 1\}^n$. 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 $\alpha \in \{0, \mathbb{u}, 1\}^n$ obtained from the binary vector a by replacing with $\mathbb{u}$ all bits a_i for $i \in T'$ (and leaving other bits of a unchanged). On this vector, we clearly have $t_1(\alpha) = \overline{\mathbb{u}} = \mathbb{u}$ and $t_2(\alpha) = t_2(a) = 1$. Moreover, since $t_+(a) = 1$, we also have $t_+(\alpha) \in \{\mathbb{u}, 1\}$,

namely, $t_+(\alpha) = 1$ if $S \cap T = \varnothing$ (t is a nonzero term), and $t_+(\alpha) = \mathbb{u}$ if $S \cap T \neq \varnothing$ (t is a zero term). Since, in our case, t is a zero term, we actually have $t_+(\alpha) = \mathbb{u}$. Thus, $t(\alpha) = \mathbb{u}$. Since $f(a) \leqslant f^{\triangledown}(a) = 0$, we also know that $t'(a) = 0$ and, hence, also $t'(\alpha) \in \{0, \mathbb{u}\}$ holds for every remaining term t' of D. Thus $F(\alpha) = \mathbb{u}$.

On the other hand, since the ternary vector α has unstable bits $\mathbb{u}$ 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, as desired. □

Remark 7. 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 $F = x\overline{x}yz \vee y\overline{z}$ computing the Boolean function $f = y\overline{z}$. The circuit has a 0-hazard at $\alpha = (\mathbb{u}, 1, 1)$: $f(0,1,1) = f(1,1,1) = 0$ but $F(\mathbb{u},1,1) = \mathbb{u}$. However, the positive part $t_+ = xyz$ of the (unique) zero term $t = x\overline{x}yz$ *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 Lemma 2, the circuit $F_+ = xyz \vee y$ computes $f^{\triangledown}$.

Remark 8. 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 $F = (x \vee \overline{z})(y \vee z)$ computing $f = xz \vee y\overline{z}$. Since the circuit F produces all three prime implicants of f (see Remark 6), 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 $F = xz \vee y\overline{z}$ for the same function $f = xz \vee y\overline{z}$. 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 $\alpha = (1,1,\mathbb{u})$: $f(1,1,1) = f(1,1,0) = 1$ but $F(1,1,\mathbb{u}) = \mathbb{u} \vee \overline{\mathbb{u}} = \mathbb{u}$.

7. A gap between hazard-free and general circuits

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

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 [17] shows $L_{\mathrm{m}}(\mathrm{per}_n) = n^{\Omega(\log n)}$, and Theorem 2 yields $L_{\mathbb{u}}(f_n) \geqslant L_{\mathrm{m}}(\mathrm{per}_n) = n^{\Omega(\log n)}$. 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. [19, 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 5.

Remark 10. 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 [16, Theorem 4.2], every monotone circuit computing per_n has depth $\Omega(n)$. Since $f_n^{\triangledown} = \mathrm{per}_n$, Theorem 2 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.

8. Open problems

Recall that $L_*(f)$ 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 also $L_{\mathfrak{U}}(n)$ be the Shannon function for hazard-free circuits, that is, the 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 $L_*(f) \geqslant L_{\mathfrak{U}}(f)$ holds for any Boolean function f (see Eq. (1)). So, the following natural open problems remain.

1. How large the gaps $L_{\mathfrak{U}}(f)/L_{\mathrm{m}}(f^{\triangledown})$ and $L_*(f)/L_{\mathfrak{U}}(f)$ can be?
2. What is the asymptotic of the Shannon function $L_{\mathfrak{U}}(n)$?

Clearly, the Shannon lower bound for general DeMorgan circuits yields $L_{\mathfrak{U}}(n) \geqslant L(n) \geqslant 2^n/n$. On the other hand, we already know that the OR of all prime implicants of any Boolean function f is a hazard-free DeMorgan circuit computing f. All nonzero terms on a given set of $n/2$ variables can be simultaneously computed by a trivial hazard-free circuit using $O(3^{n/2}) = O(2^{.8n})$ gates. We then can combine the corresponding halves of prime implicants using at most 2^n gates. This gives $L_{\mathfrak{U}}(n) \leqslant (1 + o(1))2^n$. So, Problem 2 is about closing the "$1/n$ vs. $1 + o(1)$" gap.

Acknowledgment. I thank Igor Sergeev for interesting discussions and suggestions.

References

[1] W. Baur and V. Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983.
[2] 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.
[3] S. H. Cadwell. *Switching Circuits and Logical Design*. John Willey & Sons, 1958.
[4] 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.
[5] E. Eichelberger. Hazard detection in combinational and sequential switching circuits. *IBM J. Res. Develop.*, 9:90–99, 1965.
[6] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.
[7] 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.
[8] 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.
[9] D. A. Huffman. The design and use of hazard-free switching networks. *J. ACM*, 4(1):47–62, 1957.
[10] C. Ikenmeyer, K. Balagopal, C. Lenzen, V. Lysikov, A. Mokhov, and K. Sreenivasaiah. On the complexity of hazard-free circuits. *J. ACM*, 66(4):Article 25, 2019.

[11] S. C. Kleene. *Introduction to Metamathematics*. North Holland, 1952.
[12] S. Körner. *Experience and Theory: An Essay in the Philosophy of Science*. Routledge & Kegan Paul, London, 1966.
[13] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
[14] G. Malinowski. Kleene logic and inference. *Bull. Section Logic*, 43(1/2):42–52, 2014.
[15] M. Mukaidono. On the B-ternary logical function–A ternary logic considering ambiguity. *Syst. Comput. Controls*, 3(3):27–36, 1972.
[16] R. Raz and A. Wigderson. Monotone circuits for matching require linear depth. *J. ACM*, 39(3):736–744, 1992.
[17] 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.
[18] É. Tardos. The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica*, 7(4):141–142, 1987.
[19] I. Wegener. *The complexity of Boolean functions*. Wiley-Teubner, 1987.
[20] M. Yoeli and S. Rinon. Application of ternary algebra to the study of static hazards. *J. ACM*, 11(1):84–97, 1964.

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il