SIAM J. Discrete Math.
Vol. 0, No. 0, pp. 000–000

NOTES ON HAZARD-FREE CIRCUITS[*]

STASYS JUKNA[†]

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 and even in cybersecurity. We show that a DeMorgan circuit, that is, a Boolean AND, OR, NOT circuit with negations applied to only input variables, is hazard-free iff 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), pp. 90–99] showing this property for circuits producing no terms containing a variable together with its negation. Via an amazingly simple proof, we also strengthen a recent result of Ikenmeyer et al. [*J. ACM*, 66 (2019), 25]: not only do the complexities of hazard-free and monotone circuits for monotone Boolean functions coincide, but every minimal hazard-free circuit for a monotone Boolean function must be monotone. We also observe that hazard-free implementations of already very simple Boolean functions require a superpolynomial increase of circuit size and depth, such as, for example, the Boolean function which accepts a Boolean square matrix iff every row and every column has exactly one 1. Finally, we show that the order of growth of the Shannon function of hazard-free circuits is the same as that of unrestricted circuits.

Key words. hazard-free circuits, monotone circuits, lower bounds, Shannon function

AMS subject classifications. 68Q17, 94C10

DOI. 10.1137/20M1355240

1. Introduction.

The problem of designing hazard-free circuits naturally occurs when implementing circuits in hardware [6, 12] but is also closely related to questions in logic [14, 16, 18] and even in cybersecurity [11, 29]. The importance of hazard-free circuits is highlighted in the classical textbook [6].

In this paper, under a *circuit* we will understand a *DeMorgan circuit*, that is, a Boolean circuit with fanin-2 AND and OR operations as 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 input variables. The *size* of a circuit is the number of non-input gates.

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

A mathematical definition of hazards not relating on a vague notion of "possible glitches" is given in section 2. But to give an illuminating example right now, let us consider an optimal circuit $F = xz \vee y\overline{z}$ for the Boolean function which outputs x if $z = 1$ and outputs y if $z = 0$ (this function is known as *multiplexer*). If due to different delays at wires or gates, the output of the AND gate xz of the circuit F reaches the OR gate *later* than that of the AND gate $y\overline{z}$, and if we replace input $a = (1, 1, 0)$ by

[*]Received by the editors July 24, 2020; accepted for publication (in revised form) January 31, 2021; published electronically DATE.
https://doi.org/10.1137/20M1355240
Funding: This research was supported by DFG grant JU 3105/1-2 (German Research Foundation).

[†]Faculty of Mathematics and Computer Science, Vilnius University, Vilnius, Lithuania (stjukna@gmail.com, http://www.thi.cs.uni-frankfurt.de/∼jukna/).

$b = (1,1,1)$, then the circuit will output 0 before it outputs the correct value 1: for a short moment, the output OR gate will "see" the *new* value 0 of $y\overline{z}$ on the input b and the *old* value 0 of xz on the input a. Thus, some hardware implementations of the circuit F may have a spurious 1-0-1 glitch during the input transition $a \to b$.

That every Boolean function f can be computed by a hazard-free DeMorgan circuit was shown by Huffman [12] in 1957: the DNF (disjunctive normal form) whose terms are prime implicants of f is hazard-free. In 1965, Eichelberger [8, Theorem 2] extended this result: if a DNF D representing f contains no zero-terms, that is, terms with a variable together with its negation, then D is hazard-free iff D contains all prime implicants of f as terms. For example, this theorem implies that the circuit $F = xy \vee xz \vee y\overline{z}$ for the multiplexer function $f = xz \vee y\overline{z}$ is hazard-free. After these structural results, research mainly concentrated on developing algorithms for detecting hazards, and many important results were obtained by Yoeli and Rinon [31], Eichelberger [8], Bredeson and Hulina [5], Máté, Das, and Chuang [19], Mukaidono [20], and other authors.

But somewhat surprisingly, the following natural question remained unanswered: by how much must the size of a circuit be increased to ensure the hazard-freeness? Only recently, important progress toward this question was made by Ikenmeyer et al. [13]: the hazard-free circuit complexity of any Boolean function $f : \{0,1\}^n \to \{0,1\}$ is at least the monotone circuit complexity of the monotone Boolean function $f^{\triangledown}(x) = \bigvee_{z \leq x} f(z)$. Together with known lower bounds on the later complexity, this implies that achieving hazard-freeness can require a superpolynomial blow up in circuit size. Since monotone circuits are hazard free (a "folklore" fact), and since $f^{\triangledown} = f$ holds for monotone functions f, this also implies that the hazard-free circuit complexity of monotone Boolean functions coincides with their monotone circuit complexity. Also, Ikenmeyer et al. [13] and Komarath and Saurabh [15] have shown that detecting the presence of hazards is computationally hard.

Our results are the following:

- A DeMorgan circuit computing a Boolean function f is hazard-free iff it produces (purely syntactically) all prime implicants and all prime implicates of f (Theorem 5 in subsection 6.4). This removes the "no zero-terms" requirement in the aforementioned Eichelberger theorem [8, Theorem 2].
- If a circuit computing a Boolean function f is hazard-free, then the monotone circuit obtained by replacing every negated input variable $\overline{x}_i$ with constant 1 computes $f^{\triangledown}$. This strengthens the aforementioned result of Ikenmeyer et al. [13]; the proof is short and amazingly simple (Theorem 1 in section 3). In particular, minimal hazard-free circuits for monotone Boolean functions are monotone.
- Already very simple Boolean functions require a superpolynomial blow up in the circuit size and depth to be computed by hazard-free circuits (section 5).
- The order of growth of the Shannon function for hazard-free circuits is the same as that for unrestricted circuits: it is $\Theta(2^n/n)$ (section 7).

2. Preliminaries.

We will use standard concepts concerning Boolean functions as, for example, in the standard references [7, 30], but let us recall them for completeness.

2.1. Implicants and implicates.

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, and a *clause* is an OR of literals. A term is a *zero-term* if it contains a variable x_i together with its negation $\overline{x}_i$. Terms 0 and 1 are constant terms (note that 0 is not a zero-term, it is a constant term). By analogy with the usual convention for products, we often omit the operator $\wedge$ and denote conjunction by mere juxtaposition. For example, we will write $x\overline{y}z$ instead of $x \wedge \overline{y} \wedge z$. Similarly, a *clause* is an OR of literals. A clause containing a variable together with its negation is a *one-clause*. A *non-zero* term is a term which is not a zero-term, and a *non-one* clause is a clause which is not a one-clause. A *DNF* is an OR of terms, and a *CNF* (conjunctive normal form) is an AND of clauses.

An *implicant* of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is a non-zero term t such that $t \leq f$ holds, that is, for every $a \in \{0,1\}^n$, $t(a) = 1$ implies $f(a) = 1$. In other words, a non-zero term t is an implicant of f if every evaluation of the literals of t to 1 already forces the function f to take value 1, regardless of the 0/1 values given to the remaining variables. An implicant t of f is a *prime implicant* of f if no proper subterm t' of t has this property, that is, if $t \leq t' \leq f$, then $t' = t$. For example, if $f(x,y,z) = xy \vee x\overline{y}z$, then xy, $x\overline{y}z$, and xz are implicants of f, but $x\overline{y}z$ is not a prime implicant. Dually, an *implicate* of a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is a non-one clause c such that $f \leq c$ holds, that is, for every $a \in \{0,1\}^n$, $c(a) = 0$ implies $f(a) = 0$. An implicate c of f is a *prime implicate* of f if no proper subclause of c has this property. For example, if $f(x,y,z) = xz \vee y\overline{z}$, then $c = x \vee y$ is a prime implicate of f. Implicants and implicates of f describe subcubes of the binary cube $\{0,1\}^n$ on which f is constant.

2.2. Terms and clauses produced by circuits. Every DeMorgan circuit F not only computes a unique Boolean function but also *produces* (purely syntactically) a unique set T_F of terms as well as a unique set C_F of clauses in a natural way. Namely, if $F = z$, where z is a constant or an input literal, then $T_F = C_F = \{z\}$. For non-input gates, we have

$$\begin{aligned} T_{F_1 \vee F_2} &= T_{F_1} \cup T_{F_2} && \text{and} && T_{F_1 \wedge F_2} = \{t_1 \wedge t_2 \colon t_1 \in T_{F_1}, t_2 \in T_{F_2}\}\,; \\ C_{F_1 \wedge F_2} &= C_{F_1} \cup C_{F_2} && \text{and} && C_{F_1 \vee F_2} = \{c_1 \vee c_2 \colon c_1 \in C_{F_1}, c_2 \in C_{F_2}\}\,. \end{aligned}$$

The *formal DNF* of F is the OR of all terms in T_F, and the *formal CNF* of F is the AND of all clauses in C_F. In particular, the formal DNF at an OR gate is obtained by taking the OR of the formal DNFs of the two subcircuits, while the formal DNF at an AND gate is obtained by multiplying out these two DNFs. The situation with formal CNFs is dual.

Let us stress an important point: when forming the formal DNFs or formal CNFs, all Boolean laws, except two, can be used to simplify the resulting formulas. In particular, constant laws $x \wedge 0 = 0$, $x \wedge 1 = x$, $x \vee 0 = x$, $x \vee 1 = 1$ as well as absorbtion laws $x \vee xy = x$ and $x(x \vee y) = x$ can be used. The two exceptions are the annihilation laws $x \wedge \overline{x} = 0$ and $x \vee \overline{x} = 1$: they are *not* used! Thus, some produced terms may be zero-terms, and some produced clauses may be one-clauses, that is, may contain a variable together with its negation. For example, the formal DNF $xy \vee xz \vee y\overline{z} \vee z\overline{z}$ of the circuit $F = (x \vee \overline{z})(y \vee z)$ contains a zero-term $z\overline{z}$. These "redundant" terms and clauses have no influence on the Boolean function computed by the circuit but are decisive in the context of hazards (Proposition 1 below).

2.3. Ternary logic. In this paper, we ignore the electronic aspect of hazards and stick to their idealized, mathematical model as, for example, in [5, 8, 31, 13, 15]. The classical Kleene's three-valued "strong logic of indeterminacy" [14] extends the Boolean operations AND, OR, and NOT from the Boolean domain $\{0,1\}$ to the ternary domain $\{0,\mathfrak{u},1\}$, where the bits 0 and 1 are interpreted as *stable* and the bit $\mathfrak{u}$ as *unstable*:

(2.1)

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: $x \land y = \min(x, y)$, $x \lor y = \max(x, y)$, and $\overline{x} = 1 - x$. It is easy to verify that the system $(\{0, \mathfrak{u}, 1\}, \lor, \land)$ forms a distributive lattice with zero element 0 and universal element 1; see, for example, Yoeli and Rinon [31]. In particular, the operations $\lor$ and $\land$ are associative and commutative and each distributes over the other. Moreover, the elements 0 and 1 satisfy for every $x \in \{0, \mathfrak{u}, 1\}$: $x \land 0 = 0$, $x \land 1 = x$, $x \lor 0 = x$, and $x \lor 1 = 1$. The absorbtion laws $x \lor xy = x$ and $x(x \lor y) = x$ as well as the rules of de Morgan $\overline{x \lor y} = \overline{x} \land \overline{y}$ and $\overline{x \land y} = \overline{x} \lor \overline{y}$ also hold. The only difference from the Boolean algebra is that, over the ternary domain $\{0, \mathfrak{u}, 1\}$, the annihilation laws $x \land \overline{x} = 0$ and $x \lor \overline{x} = 1$ *do not* hold: $0 \land \mathfrak{u} = 0$ but $\mathfrak{u} \land \overline{\mathfrak{u}} = \mathfrak{u} \neq 0$, and $1 \lor \mathfrak{u} = 1$ but $\mathfrak{u} \lor \overline{\mathfrak{u}} = \mathfrak{u} \neq 1$.

Remark 2.1 (from circuits to DNFs and CNFs). Since the annihilation laws $x \land \overline{x} = 0$ and $x \lor \overline{x} = 1$ are *not* used when constructing the formal DNF D or the formal CNF C produced by a given circuit F, the ternary functions computed by D and C coincide with that computed by the circuit F, that is, $D(\alpha) = C(\alpha) = F(\alpha)$ holds for every ternary vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$. This is a simple but important observation which allows us to analyze the properties of ternary functions $F : \{0, \mathfrak{u}, 1\}^n \to \{0, \mathfrak{u}, 1\}$ defined by DeMorgan circuits F by analyzing the properties of the formal DNFs and formal CNFs of these circuits.

Remark 2.2 (ternary monotonicity). One can equip the set $\{0, \mathfrak{u}, 1\}^n$ of ternary vectors with a partial order $\preccurlyeq$, where $\alpha \preccurlyeq \beta$ means that the vector β is obtained from α by replacing some unstable bits $\mathfrak{u}$ by stable bits 0 or 1. In particular, $\mathfrak{u} \preccurlyeq 0$ and $\mathfrak{u} \preccurlyeq 1$, but 0 and 1 are incomparable. Since the extensions (2.1) of gates AND, OR, and NOT to the ternary domain $\{0, \mathfrak{u}, 1\}$ are monotone with respect to $\preccurlyeq$, the obtained ternary function $F : \{0, \mathfrak{u}, 1\}^n \to \{0, \mathfrak{u}, 1\}$ computed by a DeMorgan circuit F is monotone with respect to $\preccurlyeq$. In particular, if $\alpha \preccurlyeq \beta$ and $F(\beta) = \mathfrak{u}$, then also $F(\alpha) = \mathfrak{u}$, and if $F(\alpha) = \varepsilon$ for a stable bit $\varepsilon \in \{0, 1\}$, then also $F(\beta) = \varepsilon$.

2.4. Hazards. After the functions AND, OR, NOT computed at individual gates are extended from the binary domain $\{0, 1\}$ to the ternary domain $\{0, \mathfrak{u}, 1\}$ using the truth-tables (2.1), every DeMorgan circuit F computing a given Boolean function $f : \{0, 1\}^n \to \{0, 1\}$ turns into a circuit computing some ternary function $F : \{0, \mathfrak{u}, 1\}^n \to \{0, \mathfrak{u}, 1\}$, a "ternary extension" of f, which coincides with f on $\{0, 1\}^n$. 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 $\alpha \in \{0, \mathfrak{u}, 1\}^n$ 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. To be more precise, let us fix our notation.

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

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

of all resolutions of α; hence, $|A_\alpha| = 2^m$, where m is the number of unstable bits $\mathfrak{u}$ in α. ~~For~~ example, if $\alpha = (0, \mathfrak{u}, 1, \mathfrak{u})$, then $A_\alpha = \{0\} \times \{0,1\} \times \{1\} \times \{0,1\}$. If $\alpha \in \{0,1\}^n$ (there are no unstable bits at all), then we let $A_\alpha = \{\alpha\}$. For a Boolean function f and a set $A \subseteq \{0,1\}^n$, let $f(A) = \{f(a) \colon a \in A\} \subseteq \{0,1\}$ denote the set of values taken by f on A. Hence, $f(A) = \{0\}$ iff $f(a) = 0$ for all $a \in A$, and $f(A) = \{1\}$ iff $f(a) = 1$ for all $a \in A$. In particular, $|f(A)| = 1$ means that the function f is constant on A.

Remark 2.3. If F is a DeMorgan circuit, and $\alpha \in \{0, \mathfrak{u}, 1\}^n$, then $F(A_\alpha) = \{\varepsilon\}$ for $\varepsilon \in \{0,1\}$ implies $F(\alpha) \neq \overline{\varepsilon}$, that is, $F(\alpha) \in \{\varepsilon, \mathfrak{u}\}$. This follows from Remark 2.2 but can be also shown directly. Suppose that $F(A_\alpha) = \{0\}$ but $F(\alpha) = 1$. Then $t(\alpha) = 1$ must hold for at least one term t produced by F. Since $1 \wedge \mathfrak{u} = \mathfrak{u} \neq 1$, this means that the vector α evaluates to 1 all literals of t. But then also every resolution $a \in A_\alpha$ of α evaluates these literals to 1, and we obtain $F(a) = t(a) = 1$, a contradiction with $F(a) = 0$. If $F(A_\alpha) = \{1\}$, then $F(\alpha) \in \{1, \mathfrak{u}\}$ follows by considering the clauses produced by F.

There are several types of hazards—we will consider only the so-called static logical hazards [5, 8, 31].

Definition 1 (hazards). *A circuit F of n variables has a* hazard *at $\alpha \in \{0, \mathfrak{u}, 1\}^n$ if F is constant on the subcube A_α but $F(\alpha) = \mathfrak{u}$ holds. This is a* 0-hazard *if $F(A_\alpha) = \{0\}$ and is a* 1-hazard *if $F(A_\alpha) = \{1\}$. A circuit is* hazard-free *if it has a hazard at none of the inputs $\alpha \in \{0, \mathfrak{u}, 1\}^n$.*

Example 2.4. Trivial examples of circuits with hazards are the two circuits $x \wedge \overline{x}$ and $x \vee \overline{x}$ computing the two constant functions 0 and 1. A less trivial example of such a circuit is $F = xz \vee y\overline{z}$ computing the multiplexer function $f(x,y,z) = xz \vee y\overline{z}$; we already considered this circuit in the introduction to give an intuition behind the hazards. This circuit has a 1-hazard at $\alpha = (1,1,\mathfrak{u})$: $F(\alpha) = \mathfrak{u} \vee \overline{\mathfrak{u}} = \mathfrak{u}$ even though $f(1,1,0) = f(1,1,1) = 1$. The circuit $H = (x \vee \overline{z})(y \vee z)$ for the same function has a 0-hazard at $\alpha = (0,0,\mathfrak{u})$: $H(\alpha) = \overline{\mathfrak{u}} \wedge \mathfrak{u} = \mathfrak{u}$ even though $f(0,0,0) = f(0,0,1) = 0$.

Remark 2.5. Let us stress that the mere fact that a circuit F outputs $\mathfrak{u}$ on some $\alpha \in \{0, \mathfrak{u}, 1\}^n$ does not mean that F has a hazard at α: for the latter to happen, the circuit must take the *same* value on the entire Boolean subcube A_α, that is, $|F(A_\alpha)| = 1$ must also hold. In particular, if every two vectors in $f^{-1}(\varepsilon)$ differ in at least two positions, then every DeMorgan circuit computing f is free from ε-hazards per se: if α contains at least one $\mathfrak{u}$, then $F(A_\alpha) \neq \{\varepsilon\}$. So, for example, every circuit F computing the parity function $x_1 \oplus x_2 \oplus \cdots \oplus x_n$ is hazard-free.

A "folklore" observation is that 0-hazards can only be introduced by zero-terms, and 1-hazards can only be introduced by one-clauses. Recall that a circuit is *monotone* if it has no negated variables as inputs; note that such circuits cannot produce any zero-terms or one-clauses.

Proposition 1. *If a circuit F produces no zero-terms, then F has no 0-hazards, and if F produces no one-clauses, then F has no 1-hazards. In particular, monotone circuits are hazard-free.*

Proof. Let D be the formal DNF of the circuit F, and suppose that all terms of D are non-zero terms. Assume to the contrary that the circuit F has a 0-hazard at some vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$; hence, $F(A_\alpha) = \{0\}$ but $F(\alpha) = \mathfrak{u}$. Since $D(\alpha) = F(\alpha) = \mathfrak{u}$ and $1 \vee \mathfrak{u} = 1 \neq \mathfrak{u}$, there must be a term t in D with $t(\alpha) = \mathfrak{u}$. Since $0 \wedge \mathfrak{u} = 0 \neq \mathfrak{u}$, the vector α evaluates every literal of t either to 1 or to $\mathfrak{u}$. Since t has no variable together with its negation, we can evaluate every literal of t to 1. On every such resolution $a \in A_\alpha$ of α, we have $t(a) = 1$ and, hence, also $F(a) = D(a) = 1$, a contradiction with $F(A_\alpha) = \{0\}$. The proof of the second claim (for 1-hazards) is dual by considering the formal CNF of F. □

Example 2.6. Consider the circuit $F = x(y \vee z) \vee y\overline{z}$ computing the multiplexer function $f(x, y, z) = xz \vee y\overline{z}$ from Example 2.4. Since the circuit F produces no zero-terms, it has no 0-hazards, by Proposition 1. On the other hand, the function f can take value 1 only if $x = z = 1$ or $y = \overline{z} = 1$ or $x = y = 1$. But $F(1, \mathfrak{u}, 1) = 1 \vee \mathfrak{u} = 1 \neq \mathfrak{u}$, $F(\mathfrak{u}, 1, 0) = \mathfrak{u} \vee 1 = 1 \neq \mathfrak{u}$, and $F(1, 1, \mathfrak{u}) = 1 \vee \mathfrak{u} = 1 \neq \mathfrak{u}$. So, F has no 1-hazards as well.

3. Hazard-free and monotone circuits. Recall that a Boolean function $f : \{0,1\}^n \to \{0,1\}$ is *monotone* if $f(x) = 1$ and $x \leq y$ imply $f(y) = 1$, where $x \leq y$ means that $x_i \leq y_i$ holds for all positions i. The *upward closure* of a not necessarily monotone Boolean function $f(x)$ is the monotone Boolean function

$$f^\triangledown(x) := \bigvee_{z \leq x} f(z)\,.$$

That is, $f^\triangledown(x) = 1$ iff $f(z) = 1$ holds for some vector $z \leq x$. The reason to use the adverb "upward" is the following. A *lower one* of a Boolean function f is a vector a such that $f(a) = 1$ but $f(b) = 0$ for all $b \leq a$, $b \neq a$. If $L_f \subseteq f^{-1}(1)$ denotes the set of all lower ones of f, then $f^\triangledown(x) = 1$ iff $x \geq a$ for some $a \in L_f$. For example, the set of lower ones of the parity function $f(x) = x_1 \oplus \cdots \oplus x_n$ consists of n vectors, each with exactly one 1. So, $f^\triangledown(x) = x_1 \vee \cdots \vee x_n$. In particular, $f^\triangledown = f$ holds for every monotone Boolean function f.

Note that many NP-hard problems are upward closures of relatively simple non-monotone Boolean functions. Consider, for example, a (nonmonotone) Boolean function $f(x)$ of $n = \binom{m}{2}$ variables such that $f(x) = 1$ iff the m-vertex graph G_x encoded by a 0-1 vector x consists of a complete graph on some $m/2$ vertices and $m/2$ isolated vertices. Then $f^\triangledown(x) = \mathrm{CLIQUE}(x)$ is the well-known NP-complete clique function: $\mathrm{CLIQUE}(x) = 1$ iff G_x contains a complete subgraph on $m/2$ vertices.

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 $x \in \{0,1\}^n$, where $\overline{x} = (\overline{x}_1, \ldots, \overline{x}_n)$ is the complement of $x = (x_1, \ldots, x_n)$. The *monotone 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. For example, the monotone version of the circuit $F = y\overline{z} \vee x(\overline{y} \vee \overline{x}y)$ is $F_+ = y \cdot 1 \vee x(1 \vee 1 \cdot y) = x \vee y$.

Remark 3.1. 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 some of its inputs (namely, all negated input literals), we have $F_+(x) \geq f(x)$. Since the circuit $F_+(x)$ is monotone, we also have $F_+(x) \geq F_+(z)$ for every $z \leq x$. Thus, $F_+(x) \geq f^\triangledown(x)$ holds for all $x \in \{0,1\}^n$.

THEOREM 1. *Let F be a DeMorgan circuit computing a Boolean function f. If the circuit F has no 0-hazards, then F_+ computes $f^\triangledown$.*

Proof. Assume that the circuit F_+ does not compute $f^\triangledown$. Then, by Remark 3.1, there must be a vector $a \in \{0,1\}^n$ such that $F_+(a) = 1$ but $f^\triangledown(a) = 0$. The formal DNF D_+ of the circuit F_+ is obtained by replacing with constant 1 every negated literal in the formal DNF D of the circuit F. Since $D_+(a) = F_+(a) = 1$, there must be a term $t = \bigwedge_{i \in A} x_i \wedge \bigwedge_{i \in B} \overline{x}_i$ in the DNF D such that $t_+(a) = 1$ holds for its subterm $t_+ = \bigwedge_{i\in A} x_i$. From $D(a) = f(a) = 0$, we have $t(a) = 0$; hence, the set $B' = \{i \in B \colon a_i = 1\}$ is nonempty. Take the ternary vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$ with $\alpha_i = \mathfrak{u}$ for all $i \in B'$, and $\alpha_i = a_i$ otherwise.

On this vector, we have $t(\alpha) = 1 \wedge \mathfrak{u} = \mathfrak{u}$. Since the Boolean vector a evaluates every term of D to 0, the ternary vector α evaluates every other term of D to either 0 or $\mathfrak{u}$. Hence, $F(\alpha) = D(\alpha) = \mathfrak{u}$. On the other hand, since the vector α has unstable bits $\mathfrak{u}$ only in positions where the binary vector a has 1s, every resolution $b \in A_\alpha$ of α satisfies $b \leq a$. Since $\bigvee_{b\leq a} f(b) = f^\triangledown(a) = 0$, we have $f(b) = 0$ for every resolution $b \in A_\alpha$ of α, meaning that $f(A_\alpha) = \{0\}$. Thus, the circuit F has a 0-hazard at α. □

The following example shows that the converse of Theorem 1 does not hold: the monotone version F_+ of a circuit F may compute $f^\triangledown$ even though the circuit F has 0-hazards as well as 1-hazards.

Example 3.2. Consider the circuit $F = y\overline{z} \vee x(\overline{y} \vee \overline{x}y)$ computing the Boolean function $f(x, y, z) = x\overline{y} \vee y\overline{z}$. The upward closure of f is $f^\triangledown = x \vee y$. The monotone version $F_+ = y \cdot 1 \vee x(1 \vee 1 \cdot y) = x \vee y$ of the circuit F computes $f^\triangledown$. But on the input vector $\alpha = (\mathfrak{u}, 1, 1)$, we have $F(\alpha) = 0 \vee \mathfrak{u}(0 \vee \overline{\mathfrak{u}}) = \mathfrak{u}\overline{\mathfrak{u}} = \mathfrak{u}$ even though $f(0, 1, 1) = f(1, 1, 1) = 0$, while on the vector $\beta = (1, \mathfrak{u}, 0)$, we have $F(\beta) = \mathfrak{u} \vee (\overline{\mathfrak{u}} \vee 0) = \mathfrak{u} \vee \overline{\mathfrak{u}} = \mathfrak{u}$ even though $f(1, 0, 0) = f(1, 1, 0) = 1$.

By Theorem 1, hazard-freeness of a circuit F computing a Boolean function f is a sufficient condition for $F_+ = f^\triangledown$ to hold, but Example 3.2 shows that this condition is not necessary. The following theorem shows what is actually necessary. The *positive factor* t_+ of a term t is obtained by replacing every its negated literal with constant 1.

Theorem 2. *Let F be a DeMorgan circuit computing a Boolean function f. Then the circuit F_+ computes $f^\triangledown$ iff the positive factor of every zero-term produced by F is an implicant of $f^\triangledown$.*

In particular, $F_+ = f^\triangledown$ if F produces no zero-terms.

Proof. Let $D = \bigvee_{t\in T} t$ be the formal DNF of F. Then the formal DNF of the monotone version F_+ of F is the OR $D_+ = \bigvee_{t\in T} t_+$ of positive factors of terms of D. Now take an arbitrary term $t = \bigwedge_{i\in A} x_i \wedge \bigwedge_{j\in B} \overline{x}_j$ of D,. If t is a zero-term ($A \cap B \neq \emptyset$), then $t(a) = 0$ holds for all vectors $a \in \{0, 1\}^n$; hence, $t^\triangledown(a) = 0$ holds for all a as well. It t is not a zero-term ($A \cap B = \emptyset$), then the vector $b \in \{0, 1\}^n$ with $b_i = 1$ iff $i \in A$ is the only lower one of t. Thus, $t^\triangledown(a) = 1$ iff $a \geq b$ iff $t_+(a) = 1$.

Thus, for every term t we have $t^\triangledown = 0$ if t is a zero-term and $t^\triangledown = t_+$ if t is a non-zero term. Since $(g \vee h)^\triangledown = g^\triangledown \vee h^\triangledown$ holds for any Boolean functions $g, h : \{0, 1\}^n \to \{0, 1\}$, we obtain

$$f^\triangledown = \bigvee_{t\in T} t^\triangledown = \bigvee_{t\in T'} t_+ \leq \bigvee_{t\in T} t_+ = F_+\,, \tag{3.1}$$

where $T' \subseteq T$ is the set of all non-zero terms of D. By (3.1), the equality $f^\triangledown = F_+$ holds iff $t_+ \leq f^\triangledown$ holds for every term $t \in T \setminus T'$, that is, iff the positive factor t_+ of every zero-term t of D is an implicant of $f^\triangledown$. □

By Proposition 1, if a circuit F has a 0-hazard, then F must produce at least one zero-term. Theorems 1 and 2 yield a partial converse: if F produces a zero-term whose positive factor is not an implicant of the upward closure of the Boolean function computed by F, then F has a 0-hazard.

Corollary 1. *Let F be a DeMorgan circuit computing a Boolean function f. If F produces a zero-term t such that $t_+ \not\leq f^\triangledown$, then F has a 0-hazard.*

Example 3.2 shows that the converse of Corollary 1 does not hold.

4. Complexity bounds. For a Boolean function f, let $L(f)$ denote the minimum number of gates in a DeMorgan circuit computing f. By $L_{\mathfrak{u}}(f)$ and $L_+(f)$ we denote the versions of this measure when restricted, respectively, to hazard-free circuits and to monotone circuits. Let also $L_{\mathrm{z}}(f)$ denote the minimum number of gates in a DeMorgan circuit that computes f and produces no zero-terms. In the measure $L_{\mathrm{pz}}(f)$, we additionally require that the circuit must produce all prime implicants of f; hence, $L_{\mathrm{z}}(f) \leq L_{\mathrm{pz}}(f)$. We only state the following corollary for the circuit *size* measures, but it also holds for the circuit *depth* measures.

COROLLARY 2. *For every Boolean function f, we have $L_+(f^{\triangledown}) \leq L_{\mathfrak{u}}(f) \leq L_{\mathrm{pz}}(f)$ and $L_+(f^{\triangledown}) \leq L_{\mathrm{z}}(f)$. If f is monotone, then $L_+(f) = L_{\mathfrak{u}}(f) = L_{\mathrm{z}}(f) = L_{\mathrm{pz}}(f)$.*

Proof. The inequality $L_+(f^{\triangledown}) \leq L_{\mathfrak{u}}(f)$ follows from Theorem 1, and the inequality $L_+(f^{\triangledown}) \leq L_{\mathrm{z}}(f)$ follows from Theorem 2. The inequality $L_{\mathfrak{u}}(f) \leq L_{\mathrm{pz}}(f)$ follows from a classical result of Eichelberger [8, Theorem 2]: if a DeMorgan circuit F computing a Boolean function f produces no zero-terms, then F is hazard-free iff the circuit F produces all prime implicants of f. Now let f be a monotone Boolean function; hence, $f^{\triangledown} = f$. To show that then $L_+(f) = L_{\mathfrak{u}}(f) = L_{\mathrm{z}}(f) = L_{\mathrm{pz}}(f)$ holds, it is enough to show that $L_{\mathrm{pz}}(f) \leq L_+(f)$ holds. So, let F be a monotone circuit of size $L_+(f)$ computing f. Since the circuit F has no negated inputs, it cannot produce any zero-terms, and it is enough to show that every prime implicant of f must be produced by F. This a well-known and easy to verify fact.

Assume for a contradiction that some prime implicant $p = \bigwedge_{i \in S} x_i$ of f is not produced by F, and consider the vector $a \in \{0,1\}^n$ with $a_i = 1$ for all $i \in S$ and $a_i = 0$ for all $i \in S$. On this vector, we have $f(a) = p(a) = 1$. But since every term $t \neq p$ produced by F must be an implicant of f, and since p is a *prime* implicant, t must have a variable x_i with $i \notin S$. Thus, $t(a) = 0$ holds for all terms t produced by F. But then $F(a) = 0 \neq f(a)$, a contradiction. $\square$

The lower bound $L_{\mathfrak{u}}(f) \geq L_+(f^{\triangledown})$ was already obtained by Ikenmeyer et al. [13] as a special case of a more general result proved using different arguments. The authors introduce a notion of so-called hazard derivatives of circuits and functions, prove a chain rule for these derivatives, and use it to transform a hazard-free circuit computing a Boolean function f into a monotone circuit computing the hazard derivatives of f without increasing the size of the resulting 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. Since, by its definition, the hazard derivative of f at the point $\vec{0}$ coincides with $f^{\triangledown}$, the lower bound $L_{\mathfrak{u}}(f) \geq L_+(f^{\triangledown})$ follows.

Theorem 1 gives an alternative, short and direct proof of the lower bound $L_{\mathfrak{u}}(f) \geq L_+(f^{\triangledown})$ by using the mere definition of hazards: if $F_+(a) \neq f^{\triangledown}(a)$, then the circuit F must produce a term t such that $t_+(a) = 1$ but $t(a) = 0$, and the circuit F has a 0-hazard at the ternary vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$ with $\alpha_i = \mathfrak{u}$ if $a_i = 1$ and $\overline{x}_i \in t$, and $\alpha_i = a_i$ otherwise.

Moreover, in Theorem 1, the desired monotone circuit F_+ computing $f^{\triangledown}$ is obtained from a hazard-free circuit F computing f by just replacing all negated inputs of F with constant 1: no further transformations of the circuit itself are necessary. Thus, for monotone Boolean functions f, Theorem 1 tells us a bit more than the mere equality $L_{\mathfrak{u}}(f) = L_+(f)$: it shows that not only do hazard-free and monotone circuit *complexities* for monotone Boolean functions f coincide but, in fact, that every minimal hazard-free circuit for f is a monotone circuit *itself*, that is, does not use negated input variables to compute its values. We will later show that the same holds even without the adjective "minimal" (Corollary 5 in subsection 6.3).

5. Complexity gaps. Together with already known lower bounds on the monotone circuit complexity, the lower bound $L_{\mathrm{u}}(f) \geq L_{+}(f^{\triangledown})$ implies that the gap $L_{\mathrm{u}}(f)/L(f)$ can be superpolynomial and even exponential. Such gaps were shown in [13], when f is either the logical permanent [24] or the logical determinant, or the Tardos function [28]. However, the known circuits for these functions demonstrating that $L(f)$ is polynomial are far from being trivial. Actually, except for determinant [3], we even do not have *explicit* constructions of these circuits—we only have general algorithms: [17, 10] for logical permanent and [9] for the Tardos function.

Actually, superpolynomial gaps $L_{\mathrm{u}}(f)/L(f)$ are already achieved on very simple Boolean functions f. Consider, for example, the *exact perfect matching* function f_n of $n = m^2$ variables. Inputs are Boolean $m \times m$ matrices $x = (x_{i,j})$, and $f_n(x) = 1$ iff x is permutation matrix, that is, if every row and every column of x has exactly one 1. By viewing x as the adjacency matrix of a bipartite $m \times m$ graph G_x, we have $f_n(x) = 1$ iff the graph G_x is a perfect matching. The *depth* of a circuit is the maximum, over all input-to-output paths, of the number of wires in these paths.

COROLLARY 3. *The exact perfect matching function f_n can be computed by a DeMorgan circuit of size $\mathcal{O}(n)$ and depth $\mathcal{O}(\log n)$, but any hazard-free circuit computing f_n must have size $n^{\Omega(\log n)}$ and depth $\Omega(\sqrt{n})$.*

Proof. The *logical permanent* function per_n accepts a Boolean $m \times m$ matrix x iff $f_n(z) = 1$ holds for at least one matrix $z \leq x$. Hence, $\mathrm{per}_n = f_n^{\triangledown}$ is the upward closure of f_n. Razborov [24] has shown that any monotone circuit computing per_n must have size $n^{\Omega(\log n)}$, and Raz and Wigderson [23, Theorem 4.2] have shown that any monotone circuit computing per_n has depth $\Omega(m) = \Omega(\sqrt{n})$. Together with Corollary 2, this implies that any hazard-free circuit computing f_n must have size $n^{\Omega(\log n)}$ and depth $\Omega(n)$. On the other hand, every *exact-k* function

$$\mathrm{E}_k(x_1, \ldots, x_m) = 1 \text{ iff } x_1 + \cdots + x_m = k,$$

which accepts an input vector iff it has exactly a given number k of 1s, is symmetric, and it is known (see, e.g., [30, Chapter 3.4]) that every symmetric Boolean function of m variables can be computed by a DeMorgan circuit of size $\mathcal{O}(m)$ and depth $\mathcal{O}(\log m)$. So, the exact perfect matching function f_n can be computed by a circuit of size $\mathcal{O}(m^2) = \mathcal{O}(n)$ and depth $\mathcal{O}(\log n)$. □

Remark 5.1. By allowing a slightly larger than linear number of gates, the exact permutation function f_n can be directly computed by a trivial circuit without using circuits for symmetric functions. Associate with every row i the circuit $R_i = \bigvee_{j=1}^{m} \overline{x}_{i,1} \cdots \overline{x}_{i,j-1} \cdot x_{i,j} \cdot \overline{x}_{i,j+1} \cdots \overline{x}_{i,m}$ and with every column j the circuit $C_j = \bigvee_{i=1}^{m} \overline{x}_{1,j} \cdots \overline{x}_{i-1,j} \cdot x_{i,j} \cdot \overline{x}_{i+1,j} \cdots \overline{x}_{m,j}$. Note that $R_i(x) = 1$ iff the ith row of x has exactly one 1 and $C_j(x) = 1$ iff the jth column of x has exactly one 1. Thus, the circuit $F = \bigwedge_{i=1}^{m} R_i \wedge \bigwedge_{j=1}^{m} C_j$ (which is actually a formula) computes f_n. The size of this circuit is $\mathcal{O}(m^3) = \mathcal{O}(n^{3/2})$ and the depth is $\mathcal{O}(\log n)$. If unbounded fanin gates are allowed, then the size of F is $\mathcal{O}(m^2) = \mathcal{O}(n)$ and the depth is 3.

By using less trivial circuits, one can increase the gap from superpolynomial to exponential. The *exact k-clique* function $f_{n,k}$ has $n = \binom{m}{2}$ variables corresponding to the edges of the complete graph K_m on the set $\{1, \ldots, m\}$ of vertices. Every assignment x of 0/1 values to these variables specifies a subgraph G_x of K_m, and $f_{n,k}(x) = 1$ iff G_x is an exact k-clique, that is, consists of a complete graph on some k vertices and $m - k$ isolated vertices.

Corollary 4. *For every $1 \leq k \leq m$, the exact k-clique function $f_{n,k}$ can be computed by a DeMorgan circuit of size $\mathcal{O}(n)$ and depth $\mathcal{O}(\log n)$, but for k about $(m/\log m)^{2/3}$, any hazard-free circuit computing $f_{n,k}$ must have size $2^{\Omega((n/\log n)^{1/6})}$ and depth $\Omega((n/\log n)^{1/6})$.*

Proof. The upward closure $f^{\triangledown}_{n,k}$ of $f_{n,k}$ is the well-known *k-clique function*: $\mathrm{CLIQUE}_{n,k}(x) = 1$ iff G_x contains a k-clique, that is, a complete subgraph on k vertices. Alon and Boppana [1, Theorem 3.9] have show that, for $k = \lfloor \frac{1}{4}(m/\log m)^{2/3} \rfloor$, every monotone circuit computing $\mathrm{CLIQUE}_{n,k}$ must have at least $\exp\big((n/\log n)^{1/6}\big)$ gates. Since the depth is at least logarithmic in the circuit size, we also have the claimed lower bound on the circuit depth. By Corollary 2, every hazard-free circuit computing $f_{n,k}$ must have at least as large size and depth.

To show the upper bounds, observe that a subgraph G of K_m on $\{1, \ldots, m\}$ is an exact k-clique iff G has exactly k vertices of degree $k-1$ and $k(k-1)/2$ edges in total. We first compute the values $y_i = \mathrm{E}_{k-1}(x_{i,1}, \ldots, x_{i,i-1}, x_{i,i+1}, \ldots, x_{i,m})$ for all vertices i of K_m. Since $y_i = 1$ iff the vertex i has degree $k-1$, the circuit $F(x) = \mathrm{E}_{k(k-1)/2}(x) \wedge \mathrm{E}_k(y_1, \ldots, y_m)$ computes $f_{n,k}$. The size of the circuit F is $\mathcal{O}(m^2) = \mathcal{O}(n)$ and the depth is $\mathcal{O}(\log n)$. □

Remark 5.2. An intuitive explanation for large gaps between the sizes of hazard-free and unrestricted circuits is given by Corollary 1. In an unrestricted circuit F computing the exact k-clique function $f_{n,k}$, we have no restrictions on the form of produced zero-terms. However, if F is hazard-free, then Corollary 1 implies that every produced zero-term t must have the following property: the graph encoded by the unnegated variables of t must contain a k-clique. This is a severe restriction on the usage of negations.

6. Structure of hazards. A classical result of Eichelberger [8, Theorem 2] states that if a DNF representing a Boolean function f has no zero-terms, then it is hazard-free iff contains all prime implicants of f as terms. The goal of this section is to remove the "no zero-terms" restriction from Eichelberger's theorem and to establish further structural properties of hazards.

6.1. Ternary vectors as terms and clauses. It will be convenient to identify implicants and implicates of boolean functions with ternary vectors. Namely, associate with every ternary vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$ the following term and clause:

$$t_\alpha := \bigwedge_{i:\alpha_i \neq \mathfrak{u}} x_i^{\alpha_i} \quad \text{and} \quad c_\alpha := \bigvee_{i:\alpha_i \neq \mathfrak{u}} x_i^{\overline{\alpha}_i} \,.$$

For example, if $\alpha = (1, \mathfrak{u}, 0, \mathfrak{u})$, then $t_\alpha = x_1\overline{x}_3$ and $c_\alpha = \overline{x}_1 \vee x_3$. Note that $t_\alpha(\alpha) = 1$ and $c_\alpha(\alpha) = 0$; hence,

$$t_\alpha^{-1}(1) = A_\alpha = c_\alpha^{-1}(0)\,.$$

We say that a ternary vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$ is a *1-witness* of a Boolean function f if the term t_α is an implicant of f, that is, if $t_\alpha \leq f$ holds, and that α is a *prime 1-witness* of f if the term t_α is such. Dually, $\alpha \in \{0, \mathfrak{u}, 1\}^n$ is a *0-witness* of f if the clause c_α is an implicate of f, that is, if $f \leq c_\alpha$ holds, and that α is a *prime 0-witness* of f if the clause c_α is such.

Since $t_\alpha \leq f$ iff $t_\alpha^{-1}(1) \subseteq f^{-1}(1)$, and $f \leq c_\alpha$ iff $c_\alpha^{-1}(0) \subseteq f^{-1}(0)$, for every $\varepsilon \in \{0, 1\}$, we have that

$$\alpha \in \{0, \mathfrak{u}, 1\}^n \text{ is an } \varepsilon\text{-witness of } f \text{ iff } f(A_\alpha) = \{\varepsilon\}.$$

Hence, a circuit F computing f has an ε-hazard iff $F(\alpha) = \mathfrak{u}$ holds for some ε-witness $\alpha \in \{0, \mathfrak{u}, 1\}^n$ of f. It is almost immediate that it is enough to consider only *prime* witnesses.

Proposition 2. *Let F be a DeMorgan circuit computing a Boolean function f, and $\varepsilon \in \{0, 1\}$. If F has an ε-hazard, then F has an ε-hazard at some prime ε-witness of f.*

Proof. Suppose that the circuit F has a 1-hazard at some vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$; hence, $f(A_\alpha) = \{1\}$ but $F(\alpha) = \mathfrak{u}$. Since $t_\alpha(a) = 1$ can hold only if $a \in A_\alpha$, we have $t_\alpha \leq f$, that is, t_α is an implicant of f. Then t_α must contain some *prime* implicant t of f as a (not necessarily proper) subterm. This subterm is of the form $t = t_\beta$ for the vector $\beta \in \{0, \mathfrak{u}, 1\}^n$ obtained from α by switching some stable bits to $\mathfrak{u}$. Since t_β is a prime implicant of f, the vector β is a prime 1-witness of f, and it remains to show that $f(A_\beta) = \{1\}$ and $F(\beta) = \mathfrak{u}$ hold. Since $t_\beta \leq f$, and since $t_\beta(a) = 1$ can hold only if $a \in A_\beta$, the equality $f(A_\beta) = \{1\}$ follows. On the other hand, replacing stable bits 0/1 by the unstable bit $\mathfrak{u}$ in the input vector α cannot change the unstable output $\mathfrak{u}$ of DeMorgan circuit to a stable output 0 or 1 (see Remarks 2.2 and 2.3). So, $F(\alpha) = \mathfrak{u}$ implies $F(\beta) = \mathfrak{u}$, as desired.

The case when the circuit F has a 0-hazard is similar by considering the clause c_α instead of the term t_α. □

6.2. Structure of 1-hazards. The following theorem gives us four necessary and sufficient conditions for a circuit to have a 1-hazard. For a term t and a clause c we write $t \cap c = \emptyset$ if t and c do not intersect, i.e., do not share a literal in common.

Theorem 3 (1-hazards). *Let F be a DeMorgan circuit computing a Boolean function f, and let $\alpha \in \{0, \mathfrak{u}, 1\}^n$ be a prime 1-witness of f. The following assertions are equivalent:*

(1) *F has a 1-hazard at α.*
(2) *$c(\alpha) = \mathfrak{u}$ for some one-clause c produced by F.*
(3) *$t_\alpha \cap c = \emptyset$ for some one-clause c produced by F.*
(4) *$t(\alpha) \in \{0, \mathfrak{u}\}$ for every term t produced by F.*
(5) *The prime implicant t_α of f is not produced by F.*

Proof. Let D be the formal DNF and C the formal CNF of F. Since α is a 1-witness of f, we have $f(A_\alpha) = \{1\}$.

(1) $\Leftrightarrow$ (2): To show (1) $\Rightarrow$ (2), suppose that F has a 1-hazard at α. Hence, $C(A_\alpha) = \{1\}$ but $C(\alpha) = \mathfrak{u}$. Then $c(\alpha) = \mathfrak{u}$ must hold for some clause c of C, and it remains to show that this must be a one-clause. Suppose for a contradiction that c is a non-one clause. Since $c(\alpha) = \mathfrak{u}$, the vector α evaluates every literal of c to 0 or to $\mathfrak{u}$. Since c has no variable together with its negation, negations of literals of c evaluated to $\mathfrak{u}$ do not appear in c, and we can evaluate every literal of c to 0. On every such resolution $a \in A_\alpha$ of α, we will have $c(a) = 0$ and, hence, also $C(a) = 0$, a contradiction with $C(A_\alpha) = \{1\}$.

To show the opposite implication (2) $\Rightarrow$ (1), suppose that $c_0(\alpha) = \mathfrak{u}$ holds for some one-clause c_0 of C. Since α is a 1-witness of f, we have $C(A_\alpha) = \{1\}$. So, since $1 \wedge \mathfrak{u} = \mathfrak{u}$, it remains to show that $c(\alpha) \in \{1, \mathfrak{u}\}$ holds for all clauses c of C; then $F(\alpha) = C(\alpha) = \mathfrak{u}$, meaning that F has a 1-hazard at α. Assume to the contrary that $c(\alpha) = 0$ holds for some clause c of C. Since $0 \vee \mathfrak{u} = \mathfrak{u} \neq 0$, this means that the vector α evaluates to 0 all literals of c. But then every resolution of $a \in A_\alpha$ of α also evaluates these literals to 0, and we obtain $c(a) = 0$, a contradiction with $C(a) = 1$.

(2) $\Leftrightarrow$ (3): Let c be a one-clause of C. If $c(\alpha) = \mathfrak{u}$, then $z(\alpha) \in \{0, \mathfrak{u}\}$ holds for all literals z of c. But $t_\alpha(\alpha) = 1$ implies that the vector α evaluates all literals of t_α

to $1 \not\in \{0, \mathfrak{u}\}$; hence, $t_\alpha \cap c = \emptyset$. To show the opposite implication (3) $\Rightarrow$ (2), note that (by the definition of the term t_α) the vector α evaluates to $\mathfrak{u}$ all variables not in t_α. So, if $t_\alpha \cap c = \emptyset$, then α evaluates every literal of c to 0 or to $\mathfrak{u}$. Since c is a one-clause, it contains some variable x_i together with its negation $\overline{x}_i$. Since $c(\alpha) \neq 1$, we have $x_i(\alpha) = \mathfrak{u}$ and, hence, also $c(\alpha) = \mathfrak{u}$.

(1) $\Leftrightarrow$ (4): If F has a 1-hazard at α, then $D(\alpha) = F(\alpha) = \mathfrak{u}$ holds. So, since $1 \vee \mathfrak{u} = 1 \neq \mathfrak{u}$, $t(\alpha) \in \{0, \mathfrak{u}\}$ must hold for all terms t of D. To show the opposite implication (4) $\Rightarrow$ (1), suppose that $t(\alpha) \in \{0, \mathfrak{u}\}$ holds for every term t of D. Our goal is to show that then the circuit F has a 1-hazard at α. Since $f(A_\alpha) = \{1\}$ (α is an implicant of f), it is enough to show that $t(\alpha) = \mathfrak{u}$ holds for at least one term t of D. Suppose to the contrary that $t(\alpha) = 0$ holds for all terms t of D. Then the vector α evaluates to 0 at least one literal in every term of D. But then also every resolution $a \in A_\alpha$ of α evaluates to 0 at least one literal in every term of D, and we have $D(A_\alpha) = \{0\} \neq \{1\}$, a contradiction with $f(A_\alpha) = \{1\}$.

(4) $\Leftrightarrow$ (5): If the prime implicant t_α of f is a term of D, then $t_\alpha(\alpha) = 1 \not\in \{0, \mathfrak{u}\}$ holds for this term. This shows the implication (4) $\Rightarrow$ (5). To show the opposite implication (5) $\Rightarrow$ (4), suppose that $t(\alpha) = 1$ holds for some term t of D. Then $t(A_\alpha) = \{1\}$, that is, $A_\alpha \subseteq t^{-1}(1)$. Since $A_\alpha = t_\alpha^{-1}(1)$, this yields the inclusion $t_\alpha^{-1}(1) \subseteq t^{-1}(1)$, which can hold only if t is a subterm of t_α. Since t is an implicant of f and t_α is a *prime* implicant of f, this is possible only if $t = t_\alpha$. Hence, the prime implicant t_α is a term of D, as desired. $\square$

Example 6.1. Consider the circuit $F = x(\overline{y} \vee \overline{z}) \vee \overline{x}y$ computing the Boolean function $f(x, y, z) = x\overline{y} \vee x\overline{z} \vee \overline{x}y$. The formal DNF of F is $D = x\overline{y} \vee x\overline{z} \vee \overline{x}y$ and the formal CNF is $C = (x \vee \overline{x})(\overline{x} \vee \overline{y} \vee \overline{z})(x \vee y)(y \vee \overline{y} \vee \overline{z})$. According to Theorem 3, the circuit F has a 1-hazard by any one of the following four reasons, where $\alpha = (\mathfrak{u}, 1, 0)$ is a prime 1-witness of f, and $c = x \vee \overline{x}$ is a one-clause produced by F:

- $c(\alpha) = \mathfrak{u} \vee \overline{\mathfrak{u}} = \mathfrak{u}$;
- $y\overline{z} \cap (x \vee \overline{x}) = \emptyset$;
- $x\overline{y}(\alpha) = 0 \neq 1$ and $x\overline{z}(\alpha) = \mathfrak{u} \neq 1$ and $\overline{x}y(\alpha) = \overline{\mathfrak{u}} = \mathfrak{u} \neq 1$;
- the prime implicant $y\overline{z}$ of f is not produced by F.

And indeed, on the vector $\alpha = (\mathfrak{u}, 1, 0)$, we have $F(\alpha) = \mathfrak{u}(0 \vee 1) \vee \overline{\mathfrak{u}} = \mathfrak{u} \vee \overline{\mathfrak{u}} = \mathfrak{u}$, even though $F(0, 1, 0) = F(1, 1, 0) = 1$.

6.3. Structure of 0-hazards. There is also an analogue of Theorem 3 for 0-hazards (Theorem 4 below), and it can be proved using "dual" arguments: just interchange the roles of constants 0 and 1 as well as of terms and clauses in the proof of Theorem 3. But to stress the duality between 0-hazards and 1-hazards as well as between formal DNFs and formal CNFs, we will show that Theorem 4 *itself* is the dual version of Theorem 3.

Recall that the *dual* of a Boolean function $f(x_1, \ldots, x_n)$ is the Boolean function $f^d(x) := \neg f(\overline{x})$, where $\overline{x} = (\overline{x}_1, \ldots, \overline{x}_n)$. That is, we negate each input bit as well as the result. It is well known and easy to show (see, e.g., [7, Theorem 4.6]) that (prime) implicates of a Boolean function f are (prime) implicants of the dual function f^d, and vice versa.

The *dual* F^d of a DeMorgan circuit F is obtained by exchanging the gates AND and OR, as well as the input constants 0 and 1. For example, the dual of the circuit $F = (\overline{x} \vee y)(y \vee \overline{z}) \vee xyz$ is the circuit $F^d = (\overline{x}y \vee y\overline{z})(x \vee y \vee z)$. In particular, the dual of a clause $c = z_1 \vee \cdots \vee z_m$ is the term $c^d = z_1 \wedge \cdots \wedge z_m$, and vice versa. Also, the dual of a DNF $D = t_1 \vee \cdots \vee t_l$ is the CNF $C = t_1^d \wedge \cdots \wedge t_l^d$. That is, in the "dual world," the roles of constants 0 and 1 as well as of operations AND and OR are interchanged.

By the DeMorgan laws $\overline{x \vee y} = \overline{x} \wedge \overline{y}$ and $\overline{x \wedge y} = \overline{x} \vee \overline{y}$, we have $(f \vee g)^d = f^d \wedge g^d$, $(f \wedge g)^d = f^d \vee g^d$, and $(\neg f)^d = \neg f^d$. Thus, an easy induction on the circuit size show that a circuit F computes a Boolean function f iff the dual circuit F^d computes f^d (see, for example, [7, Theorem 1.3]). Since the DeMorgan laws hold over the ternary domain $\{0, \mathfrak{u}, 1\}$, we also have $F^d(\alpha) = \neg F(\overline{\alpha})$ for every ternary vector $\alpha \in \{0, \mathfrak{u}, 1\}^n$.

For example, the dual $F^d = x\overline{z} \vee yz$ of the circuit $F = (x \vee \overline{z})(y \vee z)$ computing the multiplexer function $f = xz \vee y\overline{z}$ computes $f^d = x\overline{z} \vee yz$. The circuit F produces the clause $c = x \vee \overline{z}$ while the circuit F^d produces the term $c^d = x\overline{z}$. The circuit F has a 0-hazard at $\alpha = (0, 0, \mathfrak{u})$ while the circuit F^d has a 1-hazard at $\overline{\alpha} = (1, 1, \mathfrak{u})$. The following simple proposition shows that such a duality between the produced clauses and terms as well as between 0-hazards and 1-hazards is not accidental.

Proposition 3. *Let F be a DeMorgan circuit, and $\alpha \in \{0, \mathfrak{u}, 1\}^n$.*
(i) *A clause c is produced by F iff the term c^d is produced by F^d.*
(ii) *F has a 0-hazard at α iff F^d has a 1-hazard at $\overline{\alpha}$.*

Proof. (i) We argue by induction on the size of the circuit F. The induction basis (when the circuit is a single literal or a constant) is trivial. For the induction step, suppose that the claim holds for all circuits of size at most $s-1$, and let F be a circuit of size s. Suppose that a clause c is produced by F.

If $F = F_1 \wedge F_2$, then c is produced by the circuit F_i for some $i \in \{1, 2\}$. By the induction hypothesis, the term c^d is produced by F_i^d. The dual of the circuit F in this case is $F^d = {F_1}^d \vee {F_2}^d$. So, the term c^d is produced by F as well.

If $F = F_1 \vee F_2$, then $c = c_1 \vee c_2$ for some clauses c_1 and c_2 produced by F_1 and F_2. By the induction hypothesis, the terms $t_1 = c_1^d$ and $t_2 = c_2^d$ are produced by the dual circuits F_1^d and F_2^d. Hence, the term $t = t_1 \wedge t_2 = c_1^d \wedge c_2^d = (c_1 \vee c_2)^d = c^d$ is produced by the dual circuit $F^d = {F_1}^d \wedge {F_2}^d$. This shows the $(\Rightarrow)$ direction of claim (i); the opposite $(\Leftarrow)$ direction can be shown via the same argument by interchanging ANDs and ORs.

(ii) A Boolean vector a is a resolution of α iff its complement $\overline{a}$ is a resolution of $\overline{\alpha}$. That is, $a \in A_\alpha$ iff $\overline{a} \in A_{\overline{\alpha}}$. Since $F(a) = 0$ iff $F^d(\overline{a}) = \neg F(a) = 1$, we have $F(A_\alpha) = \{0\}$ iff $F^d(A_{\overline{\alpha}}) = \neg F(A_\alpha) = \{1\}$. On the other hand, $F(\alpha) = \mathfrak{u}$ holds iff $t(\alpha) \in \{0, \mathfrak{u}\}$ holds for all terms and $t(\alpha) = \mathfrak{u}$ holds for at least one term $t = z_1 \wedge \cdots \wedge z_m$ produced by F. By claim (i), the clauses produced by F^d are the duals $t^d = z_1 \vee \cdots \vee z_m$ of terms t produced by F. Since $t(\alpha) \in \{0, \mathfrak{u}\}$ iff $t^d(\overline{\alpha}) \in \{1, \mathfrak{u}\}$, and since $t(\alpha) = \mathfrak{u}$ iff $t^d(\overline{\alpha}) = \mathfrak{u}$, we have $F(\alpha) = \mathfrak{u}$ iff $F^d(\overline{\alpha}) = \mathfrak{u}$. Thus, F has a 0-hazard at α iff F^d has a 1-hazard at $\overline{\alpha}$. ∎

Theorem 4 (0-hazards). *Let F be a DeMorgan circuit computing a Boolean function f, and let $\alpha \in \{0, \mathfrak{u}, 1\}^n$ be a prime 0-witness of f. The following assertions are equivalent:*
(1) *F has a 0-hazard at α.*
(2) *$t(\alpha) = \mathfrak{u}$ for some zero-term t produced by F.*
(3) *$c_\alpha \cap t = \emptyset$ for some zero-term t produced by F.*
(4) *$c(\alpha) \in \{1, \mathfrak{u}\}$ for every clause c produced by F.*
(5) *The prime implicate c_α of f is not produced by F.*

Proof. We are going to apply Theorem 3 to the dual circuit F^d. For notational simplicity, assume without loss of generality that the vector α is of the form $\alpha = (a_1, \ldots, a_m, \mathfrak{u}, \ldots, \mathfrak{u})$ with all $a_i \in \{0, 1\}$. Then the clause associated with α is $c_\alpha = x_1^{\overline{a}_1} \vee \cdots \vee x_m^{\overline{a}_m}$, and the term $t_{\overline{\alpha}} = x_1^{\overline{a}_1} \wedge \cdots \wedge x_m^{\overline{a}_m}$ associated with the complementary vector $\overline{\alpha} = (\overline{a}_1, \ldots, \overline{a}_m, \mathfrak{u}, \ldots, \mathfrak{u})$ is the dual of the clause c_α. It is well known and easy to show (see, e.g., [7, Theorem 4.1]) that for any two Boolean functions g and f, we have $f \leq g$ iff $g^d \leq f^d$. Thus, $f \leq c_\alpha$ holds iff $t_{\overline{\alpha}} \leq f^d$ holds, that is, the vector α is a prime 0-witness of f iff the vector $\overline{\alpha}$ is a prime 1-witness of f^d.

When applied to the dual circuit F^d computing the dual function f^d, Theorem 3 implies that following assertions are equivalent:

(1*) The circuit F^d has a 1-hazard at $\overline{\alpha}$.
(2*) $c(\overline{\alpha}) = \mathfrak{u}$ for some one-clause c produced by F^d.
(3*) $t_{\overline{\alpha}} \cap c = \emptyset$ for some one-clause c produced by F^d.
(4*) $t(\overline{\alpha}) \in \{0, \mathfrak{u}\}$ for every term t produced by F^d.
(5*) The prime implicant $t_{\overline{\alpha}}$ of f^d is not produced by F^d.

It is therefore enough to show that the corresponding assertions are equivalent. The equivalences (1) $\Leftrightarrow$ (1*) and (5) $\Leftrightarrow$ (5*) follow directly from Proposition 3. The remaining three equivalences also follow from Proposition 3 and the following simple observations. We have $c(\alpha) = 1$ for a clause c iff the vector α evaluates some literal of c to 1, which happens precisely when the complementary vector $\overline{\alpha}$ evaluates this literal to 0, that is, when $t(\overline{\alpha}) = 0$ holds for the term $t = c^d$. So, $c(\alpha) \in \{1, \mathfrak{u}\}$ iff $t(\overline{\alpha}) \in \{0, \mathfrak{u}\}$, and $c(\alpha) = \mathfrak{u}$ iff $t(\overline{\alpha}) = \mathfrak{u}$. Since a clause is a one-clause iff its dual is a zero term, the equivalences (2) $\Leftrightarrow$ (2*) and (4) $\Leftrightarrow$ (4*) follow. Finally, the dual $t = c^d$ of any one-clause c from claim (3*) is a zero-term produced by F. Note that both t and c have the same literals. The sets of literals of the term $t_{\overline{\alpha}} = x_1^{\overline{a}_1} \wedge \cdots \wedge x_m^{\overline{a}_m}$ and of the clause $c_\alpha = x_1^{\overline{a}_1} \vee \cdots \vee x_m^{\overline{a}_m}$ are also the same. So, $t_{\overline{\alpha}} \cap c = \emptyset$ holds precisely when $c_\alpha \cap t = \emptyset$ holds, and the equivalence (3) $\Leftrightarrow$ (3*) follows as well. □

6.4. Extended Eichelberger's theorem. As we already mentioned in the introduction, a classical result of Huffman [12] is that the DNF whose terms are prime implicants of a Boolean function f is a hazard-free circuit computing f. An also classical result of Eichelberger [8, Theorem 2] extends Huffman's theorem.

EICHELBERGER'S THEOREM. *Let F be a DeMorgan circuit computing a Boolean function f. If F produces no zero-terms, then F is hazard-free iff the circuit F produces all prime implicants of f.*

This theorem is stated and proved in [8] only for zero-term free DNFs (depth-two circuits), but it also holds for DeMorgan circuits producing no zero-terms: formal DNFs of such circuits do not have such terms.

Together with Proposition 2, Theorems 3 and 4 remove the "no zero-terms" condition from Eichelberger's theorem.

THEOREM 5 (extended Eichelberger's theorem). *Let F be a DeMorgan circuit computing a Boolean function f.*

(1) *F has no 1-hazards iff F produces all prime implicants of f.*
(2) *F has no 0-hazards iff F produces all prime implicates of f.*

In the case of circuits computing *monotone* Boolean functions, Theorem 5 strengthens Theorem 1. As before, 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 $x \in \{0,1\}^n$. Thus, for a vector $a \in \{0,1\}^n$, $H(x, a)$ is the monotone circuit obtained from F by replacing the ith negated input variable $\overline{x}_i$ with the constant $a_i \in \{0,1\}$.

COROLLARY 5. *Let $F(x) = H(x, \overline{x})$ be a hazard-free DeMorgan circuit computing a monotone Boolean function $f(x)$. Then for every $a \in \{0,1\}^n$, the circuit $H(x, a)$ computes $f(x)$.*

Thus, hazard-free circuits for monotone Boolean functions are monotone, that is, do not use negated input variables to compute their values.

Proof. Since the circuit $H(x,y)$ is monotone, we have $H(x,\vec{0}) \leq H(x,a) \leq H(x,\vec{1})$. It is therefore enough to show that both circuits $H(x,\vec{0})$ and $H(x,\vec{1})$ compute $f(x)$. By Theorem 5, the circuit $F(x) = H(x,\overline{x})$ produces all prime implicants and all prime implicates of f. Since the function f is monotone, none of these implicants or implicates contains a negated variable. So, the circuit $H(x,\vec{0})$ will still produce all prime implicants. Since every zero-term produced by F will be evaluated to 0 in $H(x,\vec{0})$, the circuit $H(x,\vec{0})$ computes f. Similarly, the circuit $H(x,\vec{1})$ will produce all prime implicates of f. Since every one-clause produced by F will be evaluated to 1 in $H(x,\vec{1})$, the circuit $H(x,\vec{1})$ computes f. ◻

7. Final remarks. Recall that $L(f)$ denotes the minimum number of gates in a DeMorgan circuit computing a Boolean function f. We also have introduced versions of this complexity measure: $L_{\mathfrak{u}}(f)$ (the circuit must be hazard-free), $L_{\mathrm{pz}}(f)$ (the circuit must produce all prime implicants of f but must produce no zero-terms), and $L_{+}(f)$ (the circuit must be monotone, i.e., must have no negated variables as inputs). Finally, 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.

By Corollary 2, we know that for every Boolean function f the inequalities $L_{+}(f^{\triangledown}) \leq L_{\mathfrak{u}}(f) \leq L_{\mathrm{pz}}(f)$ hold. So, the following natural questions arise:

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

That the *order* of growth of the function $L_{\mathfrak{u}}(n)$ is $2^n/n$ can be shown by extending the construction of Shannon [27] and Muller [21] to show $L(n) = \mathcal{O}(2^n/n)$ for unrestricted circuits. They used the recursion $F(x_1,\ldots,x_n) = \overline{x}_n \cdot F_0 \vee x_n \cdot F_1$, where F_0 and F_1 are DeMorgan circuits computing the subfunctions $f_0 = f(x_1,\ldots,x_{n-1},0)$ and $f_1 = f(x_1,\ldots,x_{n-1},1)$ of a given Boolean function $f(x_1,\ldots,x_n)$.

The circuit F constructed using this recursion will produce no zero-terms and, by Proposition 1, will have no 0-hazards. But it can have 1-hazards even if both subcircuits F_0 and F_1 are hazard-free. For this to happen, it is enough that some prime implicant t of f contains neither x_n nor $\overline{x}_n$; then t is not produced by F and, by Theorem 3, F has a 1-hazard. And indeed, we can take any ternary vector $\alpha = (a,\mathfrak{u})$ with $a \in \{0,1\}^{n-1}$ and $t(a) = 1$. Since then also $F_0(a) = 1$ and $F_1(a) = 1$, we have $F(A_\alpha) = \{1\}$. But $F(\alpha) = \overline{\mathfrak{u}} \cdot 1 \vee \mathfrak{u} \cdot 1 = \mathfrak{u}$.

A classical idea to generate prime implicants, rediscovered by several independent researchers—Blake [4], Samson and Mills [25], Quine [22]—is to use the *consensus* recursion $F = \overline{x}_n \cdot F_0 \vee x_n \cdot F_1 \vee F_0 \cdot F_1$ (known also as the *consensus theorem* or *rule of consensus*): if a term t is an implicant of f, and if t contains neither $\overline{x}_n$ nor x_n, then t is an implicant of both subfunctions f_0 and f_1 of f.

It is, however, not a priori clear that this extended recursion will not introduce 0-hazards: even if neither of the subcircuits F_0 and F_1 produces zero-terms, the additional subcircuit $F_0 \cdot F_1$ could, in general, produce zero-terms and such terms may lead to 0-hazards (see Theorem 4). Fortunately, these zero-terms are "innocent," as long as both circuits F_0 and F_1 are hazard-free, that is, the consensus recursion introduces no hazards.

Proposition 4. *Let $F_0(x_1,\ldots,x_{n-1})$ and $F_1(x_1,\ldots,x_{n-1})$ be arbitrary hazard-free circuits, and let x_n be a new variable. Then the circuit*

$$F(x_1,\dots,x_n) = \overline{x}_n \cdot F_0 \vee x_n \cdot F_1 \vee F_0 \cdot F_1 \tag{7.1}$$

is hazard-free.

Proof. Assume that the circuit F has a hazard at some vector $\alpha \in \{0,\mathfrak{u},1\}^n$; hence, $F(A_\alpha) = \{\varepsilon\}$ for some $\varepsilon \in \{0,1\}$ but $F(\alpha) = \mathfrak{u}$. Our goal is to show that then at least one of the circuits F_0 and F_1 must have an ε-hazard at α.

Case 1: $\alpha_n \in \{0,1\}$, say, $\alpha_n = 0$. Then for every resolution $a \in A_\alpha$ of α, we have $F(a) = F_0(a) \vee F_0(a)\cdot F_1(a) = F_0(a)$; hence, $F_0(A_\alpha) = F(A_\alpha) = \{\varepsilon\}$. Since $\alpha_n = 0$, we have $x_n \cdot F_1(\alpha) = 0$ and, since the absorbtion law $x \vee xy = x$ holds also over $\{0,\mathfrak{u},1\}$, we obtain $F(\alpha) = F_0(\alpha) \vee F_0(\alpha)\cdot F_1(\alpha) = F_0(\alpha)$. Thus, $F_0(\alpha) = F(\alpha) = \mathfrak{u}$, meaning that the circuit F_0 has a ε-hazard at α.

Case 2: $\alpha_n = \mathfrak{u}$. Since $F(A_\alpha) = \{\varepsilon\}$, and since the circuits F_0 and F_1 do not depend on x_n, both $F_0(A_\alpha) = \{\varepsilon\}$ and $F_1(A_\alpha) = \{\varepsilon\}$ must hold in this case. Indeed if, say, $F_0(a) = \overline{\varepsilon}$ for some resolution $a \in A_\alpha$ of α, then also $F_0(a') = \overline{\varepsilon}$ for the resolution $a' = (a_1,\dots,a_{n-1},0)$ of α, and we obtain $F(a') = 1\cdot\overline{\varepsilon} \vee 0\cdot F_1(a') \vee \overline{\varepsilon}\cdot F_1(a') = \overline{\varepsilon} \neq \varepsilon$. Thus, both $F_0(A_\alpha) = \{\varepsilon\}$ and $F_1(A_\alpha) = \{\varepsilon\}$ hold. Then both values $F_0(\alpha)$ and $F_1(\alpha)$ must belong to $\{\varepsilon,\mathfrak{u}\}$ (see Remark 2.3). Since $\mathfrak{u}\wedge 0 = 0$ and $\mathfrak{u}\vee 1 = 1$, both $F_0(\alpha) = \varepsilon$ and $F_1(\alpha) = \varepsilon$ cannot hold because then we would have $F(\alpha) = \overline{\mathfrak{u}}\cdot\varepsilon \vee \mathfrak{u}\cdot\varepsilon \vee \varepsilon = \varepsilon \neq \mathfrak{u}$. So, $F_i(\alpha) = \mathfrak{u}$ holds for some $i \in \{0,1\}$, meaning that the circuit F_i has a ε-hazard at α. ∎

When directly applied, the recursion (7.1) yields $L_\mathfrak{u}(n) = \mathcal{O}(2^n)$. Nitin Saurabh [26] suggested to combine the recursion (7.1) with the argument used by Shannon [27] and Muller [21] to show $L(n) = \mathcal{O}(2^n/n)$ for unrestricted circuits. And indeed, the combination yields much better, tight up to the order of growth upper bound $L_\mathfrak{u}(n) = \mathcal{O}(2^n/n)$.

Take an arbitrary Boolean function $f(x_1,\dots,x_n)$, and let $1 \leq m \leq n$ be an integer parameter to be specified latter. Apply the recursion (7.1) for $n-m$ steps to obtain a hazard-free circuit F_{n-m} of size $5\cdot 2^{n-m}$ computing the function f from all its 2^{n-m} subfunctions $f_b(x_1,\dots,x_m) = f(x_1,\dots,x_m,b_1,\dots,b_{n-m})$ on the first m variables. Inputs to F_{n-m} are Boolean functions from the set $\mathcal{H}_m$ of all $|\mathcal{H}_m| = 2^{2^m}$ Boolean functions $h(x_1,\dots,x_m)$ on the first m variables. Shannon's idea is this: if $2^{n-m} \gg 2^{2^m}$, then same functions from $\mathcal{H}_m$ will appear many times among the inputs of F_{n-m}. It is then more economical to simultaneously compute all the functions in $\mathcal{H}_m$ once beforehand, rather than to recompute the residual functions f_b at each of the 2^{n-m} inputs of F_{n-m}.

Using the recursion (7.1), we can construct a hazard-free circuit H_m of size at most $5\cdot 2^{2^m}$ which simultaneously computes all 2^{2^m} Boolean functions in $\mathcal{H}_m$. This can be shown by induction on m (with basis cases $m=1$ and $m=2$ being trivial). Assume that we can compute all $2^{2^{m-1}}$ functions in $\mathcal{H}_{m-1}$ using at most $5\cdot 2^{2^{m-1}}$ gates. Every function h in $\mathcal{H}_m$ is of the form $h = \overline{x}_m\cdot h_0 \vee x_m\cdot h_1 \vee h_0\cdot h_1$ where h_0 and h_1 belong to $\mathcal{H}_{m-1}$; so, h can be computed from h_0 and h_1 with five additional gates (see Figure 1, left). There are $2^{2^m} - 2^{2^{m-1}}$ functions in $\mathcal{H}_m \setminus \mathcal{H}_{m-1}$. So, all functions in $\mathcal{H}_m$ can be computed by using at most $5(2^{2^m} - 2^{2^{m-1}}) + 5\cdot 2^{2^{m-1}} = 5\cdot 2^{2^m}$ gates.

By identifying the input gates $f_b \in \mathcal{H}_m$ of the circuit F_{n-m} with the corresponding output gates of H_m, we obtain a hazard-free circuit F for f. That is, the circuit F is obtained from the circuit F_{n-m} by further applying the hazard-freeness preserving recursion (7.1) (see Figure 1, right): we only do not repeat the construction for the same input subfunctions f_b. By Proposition 4, the obtained circuit F is hazard-free. The number of gates in F is at most five times $2^{2^m} + 2^{n-m}$. For $m = \log_2(n - \log_2 n)$,

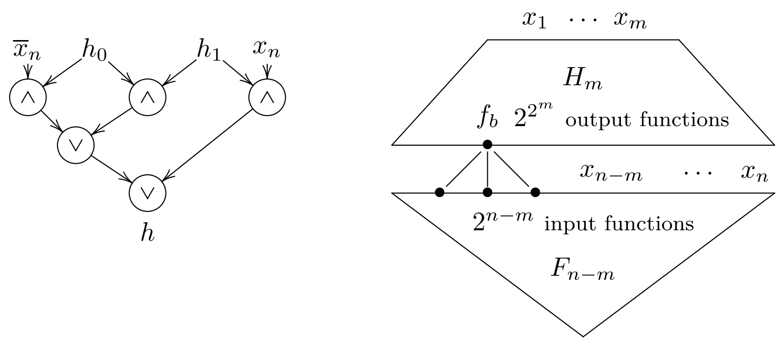

FIG. 1. *On the left is a fragment of a circuit for the consensus recursion* (7.1)*, and on the right is a schematic form of a hazard-free circuit which results after combining this recursion with the Shannon–Muller construction.*

we have $2^{2^m} = 2^n/n$ and $2^{n-m} = 2^n/(n - \log_2 n) = c \cdot 2^n/n$, where $c = 1 + \frac{\log_2 n}{n - \log_2 n}$. So, $L_{\mathtt{u}}(n) = \mathcal{O}(2^n/n)$ follows by taking m to be a nearest to $\log_2(n - \log_2 n)$ integer. Since $L_{\mathtt{u}}(n) \geq L(n) = \Omega(2^n/n)$, the order of magnitude of the Shannon function for hazard-free circuits is already known: $L_{\mathtt{u}}(n) = \Theta(2^n/n)$. It remains, however, open whether $L_{\mathtt{u}}(n) \sim 2^n/n$ holds.

Acknowledgments. I am thankful to Nitin Saurabh and Igor Sergeev for enlightening discussions, as well as to the referees for very useful suggestions to improve the presentation.

REFERENCES

[1] N. ALON AND R. BOPPANA, *The monotone circuit complexity of Boolean functions*, Combinatorica, 7 (1987), pp. 1–22, https://doi.org/10.1007/BF02579196.
[2] W. BAUR AND V. STRASSEN, *The complexity of partial derivatives*, Theoret. Comput. Sci., 22 (1983), pp. 317–330, https://doi.org/10.1016/0304-3975(83)90110-X.
[3] S. J. BERKOWITZ, *On computing the determinant in small parallel time using a small number of processors*, Inform. Process. Lett., 18 (1984), pp. 147–150, https://doi.org/10.1016/0020-0190(84)90018-8.
[4] A. BLAKE, *Canonical Expressions in Boolean Algebras*, Ph.D. thesis, Department of Mathematics, University of Chicago, 1937.
[5] J. G. BREDESON AND P. T. HULINA, *Elimination of static and dynamic hazards for multiple input changes in combinational switching circuits*, Inform. Control, 20 (1972), pp. 114–124, https://doi.org/10.1016/s0019-9958(72)90299-9.
[6] S. H. CADWELL, *Switching Circuits and Logical Design*, John Wiley & Sons, New York, 1958.
[7] Y. CRAMA AND P. L. HAMMER, EDS., *Boolean Functions: Theory, Algorithms, and Applications*, Encyclopedia Math. Appl. 142, Cambridge University Press, Cambridge, UK, 2011.
[8] E. EICHELBERGER, *Hazard detection in combinational and sequential switching circuits*, IBM J. Res. Develop., 9 (1965), pp. 90–99, https://doi.org/10.1147/rd.92.0090.
[9] M. GRÖTSCHEL, L. LOVÁSZ, AND A. SCHRIJVER, *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica, 1 (1981), pp. 169–197, https://doi.org/10.1007/bf02579273.
[10] J. HOPCROFT AND R. KARP, *An $n^{5/2}$ algorithm for maximum matching in bipartite graphs*, SIAM J. Comput., 2 (1973), pp. 225–231, https://doi.org/10.1137/0202019.
[11] 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. Inform. Forensics Secur., 7 (2012), pp. 1067–1080, https://doi.org/10.1109/tifs.2012.2189105.
[12] D. A. HUFFMAN, *The design and use of hazard-free switching networks*, J. ACM, 4 (1957), pp. 47–62, https://doi.org/10.1145/320856.320866.

[13] C. Ikenmeyer, B. Komarath, C. Lenzen, V. Lysikov, A. Mokhov, and K. Sreenivasaiah, *On the complexity of hazard-free circuits*, J. ACM, 66 (2019), 25, https://doi.org/10.1145/3320123.

[14] S. C. Kleene, *Introduction to Metamathematics*, North-Holland, Amsterdam, 1952.

[15] N. B. Komarath and N. Saurabh, *On the complexity of detecting hazards*, Inform. Process. Lett., 162 (2020), 105980, https://doi.org/10.1016/j.ipl.2020.105980.

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

[17] H. W. Kuhn, *The Hungarian method for the assignment problem*, Naval Res. Logist. Quart., 2 (1955), pp. 83–97, https://doi.org/10.1002/nav.3800020109.

[18] G. Malinowski, *Kleene logic and inference*, Bull. Section Logic, 43 (2014), pp. 42–52.

[19] L. L. Máté, S. Das, and H. Y. H. Chuang, *A logic hazard detection and elimination method*, Inform. Control, 26 (1975), pp. 351–368, https://doi.org/10.1016/S0019-9958(74)80004-5.

[20] M. Mukaidono, *The B-ternary logic and its applications to the detection of hazards in combinational switching circuits*, in Proceedings of the 8th International Symposium on Multiple-Valued Logic, 1978, pp. 269–275.

[21] D. E. Muller, *Complexity in electronic switching circuits*, IRE Trans. Comput., EC-5 (1959), pp. 15–19, https://doi.org/10.1109/tec.1956.5219786.

[22] W. V. Quine, *A way to simplify truth functions*, Amer. Math. Monthly, 62 (1955), pp. 627–631, https://doi.org/10.1080/00029890.1955.11988710.

[23] R. Raz and A. Wigderson, *Monotone circuits for matching require linear depth*, J. ACM, 39 (1992), pp. 736–744, https://doi.org/10.1145/146637.146684.

[24] A. A. Razborov, *Lower bounds on monotone complexity of the logical permanent*, Math. Notes Acad. Sci. USSR, 37 (1985), pp. 485–493, https://doi.org/10.1007/bf01157687.

[25] E. W. Samson and B. E. Mills, *Circuit Minimization: Algebra and Algorithms for New Boolean Canonical Expressions*, Tech. report TR 54-21, Air Force Cambridge Research Center, 1954.

[26] N. Saurabh, *private communication*, 2020.

[27] C. E. Shannon, *The synthesis of two-terminal switching circuits*, Bell Syst. Technical J., 28 (1949), pp. 59–98, https://doi.org/10.1002/j.1538-7305.1949.tb03624.x.

[28] E. Tardos, *The gap between monotone and non-monotone circuit complexity is exponential*, Combinatorica, 7 (1987), pp. 141–142, https://doi.org/10.1007/bf02122563.

[29] M. Tiwari, H. M. G. Wassel, B. Mazloom, S. Mysore, F. Chong, and T. Sherwood, *Complete information flow tracking from the gates up*, SIGARCH Comput. Archit. News, 31 (2009), pp. 109–120, https://doi.org/10.1145/2528521.1508258.

[30] I. Wegener, *The Complexity of Boolean Functions*, Wiley-Teubner, New York, 1987.

[31] M. Yoeli and S. Rinon, *Application of ternary algebra to the study of static hazards*, J. ACM, 11 (1964), pp. 84–97, https://doi.org/10.1145/321203.321214.