On the Complexity of Hazard-free Circuits

CHRISTIAN IKENMEYER*†, Max Planck Institute for Software Systems, Germany
BALAGOPAL KOMARATH*, Saarland University, Germany
CHRISTOPH LENZEN*, Max Planck Institute for Informatics, Germany
VLADIMIR LYSIKOV*, Saarland University, Germany
ANDREY MOKHOV, Newcastle University, United Kingdom
KARTEEK SREENIVASAIAH‡, Indian Institute of Technology Hyderabad, India

The problem of constructing hazard-free Boolean circuits dates back to the 1940s and is an important problem in circuit design. Our main lower-bound result unconditionally shows the existence of functions whose circuit complexity is polynomially bounded while every hazard-free implementation is provably of exponential size. Previous lower bounds on the hazard-free complexity were only valid for depth 2 circuits. The same proof method yields that every subcubic implementation of Boolean matrix multiplication must have hazards.

These results follow from a crucial structural insight: Hazard-free complexity is a natural generalization of monotone complexity to all (not necessarily monotone) Boolean functions. Thus, we can apply known monotone complexity lower bounds to find lower bounds on the hazard-free complexity. We also lift these methods from the monotone setting to prove exponential hazard-free complexity lower bounds for non-monotone functions.

As our main upper-bound result, we show how to efficiently convert a Boolean circuit into a bounded-bit hazard-free circuit with only a polynomially large blow-up in the number of gates. Previously, the best known method yielded exponentially large circuits in the worst case, so our algorithm gives an exponential improvement.

As a side result, we establish the NP-completeness of several hazard detection problems.

CCS Concepts: • **Theory of computation → Circuit complexity**;

Additional Key Words and Phrases: Hazards, Boolean circuits, monotone circuits, computational complexity

*Saarland University, the Max Planck Institute for Informatics, and the Max Planck Institute for Software Systems are part of the Saarland Informatics Campus.

†This work was done while this author was at the Max Planck Institute for Informatics.

‡This work was done while this author was at Saarland University.

The results of this article have been presented at the 50th Annual ACM Symposium on the Theory of Computing (STOC 2018) as [22].

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant Agreement No. 716562). Andrey Mokhov's research was supported by EPSRC Grant POETS No. EP/N031768/1.

Authors' addresses: C. Ikenmeyer, Max Planck Institute for Software Systems, Campus E1 5, 66123, Saarbrücken, Germany; email: cikenmey@mpi-sws.org; B. Komarath and V. Lysikov, Saarland University, Campus E1 3, 66123, Saarbrücken, Germany; emails: {bkomarath, vlysikov}@cs.uni-saarland.de; C. Lenzen, Max Planck Institute for Informatics, Campus E1 4, 66123, Saarbrücken, Germany; email: clenzen@mpi-inf.mpg.de; A. Mokhov, Newcastle University, School of Engineering, Newcastle upon Tyne, NE1 7RU, United Kingdom; email: andrey.mokhov@ncl.ac.uk; K. Sreenivasaiah, Indian Institute of Technology Hyderabad, Kandi, Sangareddy, Hyderabad, Telangana State, 502285, India; email: karteek@iith.ac.in.

0004-5411/2019/08-ART25
https://doi.org/10.1145/3320123

ACM Reference format:
Christian Ikenmeyer, Balagopal Komarath, Christoph Lenzen, Vladimir Lysikov, Andrey Mokhov, and Karteek Sreenivasaiah. 2019. On the Complexity of Hazard-free Circuits. *J. ACM* 66, 4, Article 25 (August 2019), 20 pages.
https://doi.org/10.1145/3320123

1 INTRODUCTION

We study the problem of *hazards* in Boolean circuits. This problem naturally occurs in digital circuit design, specifically in the implementation of circuits in hardware (e.g., [9, 21]), but is also closely related to questions in logic (e.g., [25, 26, 29]) and cybersecurity [20, 46]. Objects are called differently in the different fields; for presentational simplicity, we use the parlance of hardware circuits throughout the article.

A Boolean circuit is a circuit that uses `and`-, `or`-, and `not`-gates, in the sense of [23, Section 1.2], where `and` and `or` have fan-in two. The standard approach to studying hardware implementations of Boolean circuits is to use the digital abstraction, in which voltages on wires and at gates are interpreted as either logical 0 or 1. More generally, this approach is suitable for any system in which there is a guarantee that the inputs to the circuit and the outputs of the gates of the circuit can be reliably interpreted in this way (i.e., be identified as the Boolean value matching the gate's truth table).

Kleene Logic and Hazards

Several independent works ([17, 50] and references therein) observed that Kleene's classical three-valued *strong logic of indeterminacy* K_3 [25, Section 64] captures the issues arising from non-digital inputs. The idea is simple and intuitive. The two-valued Boolean logic is extended by a third value $\mathfrak{u}$ representing any unknown, uncertain, undefined, transitioning, or otherwise non-binary value. We call both Boolean values *stable*, while $\mathfrak{u}$ is called *unstable*. The behavior of a Boolean gate is then extended as follows. Let $\mathbb{B} := \{0, 1\}$ and $\mathbb{T} := \{0, \mathfrak{u}, 1\}$. Given a string $x \in \mathbb{T}^k$, a *resolution* $y \in \mathbb{B}^k$ of x is defined as a string that is obtained by replacing each occurrence of $\mathfrak{u}$ in x by either 0 or 1. If a k-ary gate (with one output) is subjected to inputs $x \in \mathbb{T}^k$, then it outputs $b \in \mathbb{B}$ iff it outputs b for *all* resolutions $y \in \mathbb{B}^k$ of x; otherwise, it outputs $\mathfrak{u}$. In other words, the gate outputs a Boolean value b, if and only if its output does not actually depend on the unstable inputs. This results in the following extended specifications of `and`, `or`, and `not` gates:

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

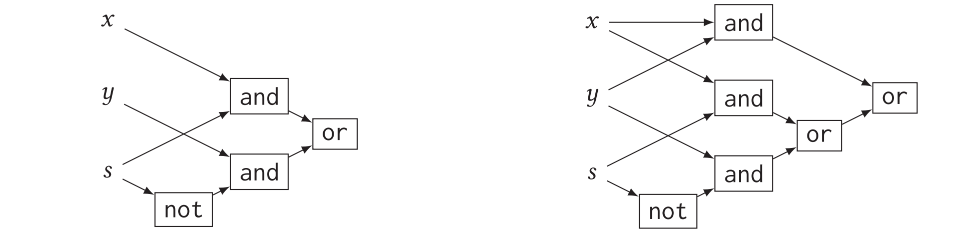
and	0	$\mathfrak{u}$	1
0	0	0	0
$\mathfrak{u}$	0	$\mathfrak{u}$	$\mathfrak{u}$
1	0	$\mathfrak{u}$	1

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

By induction over the circuit structure, a circuit C with n input gates now computes a function $C : \mathbb{T}^n \to \mathbb{T}$.

Unfortunately, in some cases, the circuit might behave in an undesirable way. Consider a multiplexer circuit (MUX), which for Boolean inputs $x, y, s \in \mathbb{B}$ outputs x if $s = 0$ and y if $s = 1$. A straightforward circuit implementation is shown in Figure 1(a). Despite the fact that $\mathtt{MUX}(1, 1, 0) = \mathtt{MUX}(1, 1, 1) = 1$, one can verify that in Figure 1(a), $\mathtt{MUX}(1, 1, \mathfrak{u}) = \mathfrak{u}$. Such behaviour is called a hazard:

Definition 1.1 (Hazard). We say that a circuit C on n inputs *has a hazard at* $x \in \mathbb{T}^n$ iff $C(x) = \mathfrak{u}$ and there is a Boolean value $b \in \mathbb{B}$ such that for all resolutions y of x we have $C(y) = b$. If C has no hazard, then it is called *hazard-free.*



$\mathtt{MUX}(x, y, s) := (x$ and $s)$ or $(y$ and not $s)$

(a) Multiplexer with hazard at $(x, y, s) = (1, 1, \mathfrak{u})$

$(x$ and $y)$ or $((x$ and $s)$ or $(y$ and not $s))$

(b) Hazard-free multiplexer

Fig. 1. Two circuits that implement the same Boolean multiplexer function. One has a hazard, and the other one is hazard-free.

The name *hazard-free* has different meanings in the literature. Our definition is taken from [11]. In Figure 1(b), we see a hazard-free circuit for the multiplexer function. Note that this circuit uses more gates than the one in Figure 1(a). The problem of detecting hazards and constructing circuits that are hazard-free started a large body of literature; see Section 2. The question whether hazards can be avoided in principle was settled by Huffman.

THEOREM 1.2 [21]. *Every Boolean function has a hazard-free circuit computing it.*

He immediately noted that avoiding hazards is potentially expensive [21, p. 54]:

> "In this example at least, the elimination of hazards required a substantial increase in the number of contacts."

Indeed, his result is derived using a clause construction based on the prime implicants of the considered function, which can be exponentially many; see, e.g., [10]. There has been no significant progress on the complexity of hazard-free circuits since Huffmann's work. Accordingly, the main question we study in this article is:

> What is the additional cost of making a circuit hazard-free?

Our Contribution

Unconditional Lower Bounds. Our first main result is that monotone circuit lower bounds directly yield lower bounds on hazard-free circuits. A circuit is *monotone* if it only uses and-gates and or-gates, but does not use any not-gates. For a Boolean function f, denote (i) by $L(f)$ its Boolean complexity, i.e., the size of a smallest circuit computing f, (ii) by $L_{\mathfrak{u}}(f)$ its *hazard-free complexity,* i.e., the size of a smallest hazard-free circuit computing f, and (iii), if f is monotone, then by $L_+(f)$ its monotone circuit complexity, i.e., the size of a smallest monotone circuit computing f. We show that $L_{\mathfrak{u}}$ properly extends L_+ to the domain of all Boolean functions.

THEOREM 1.3. *If f is monotone, then $L_{\mathfrak{u}}(f) = L_+(f)$.*

We consider this connection particularly striking, because hazard-free circuits are highly desirable in practical applications. Moreover, to our surprise the construction underlying Theorem 1.3 yields a circuit computing a new directional derivative that we call the *hazard derivative*[1] of the function at $x = 0$ in direction of y, which equals the function itself if it is monotone (and not

[1] Interestingly, this is closely related to, but *not* identical to, the Boolean directional derivative defined in, e.g., [12, Definition 3], which has applications in cryptography. To the best of our knowledge, the hazard derivative has not appeared in the literature so far.

constant 1). We consider this observation to be of independent interest, as it provides additional insight into the structure of hazard-free circuits.

We get the following (non-exhaustive) list of immediate corollaries that highlight the importance of Theorem 1.3.

Corollary 1.4 (Using Monotone Lower Bound from [40]). *Define the Boolean permanent function* $f_n : \mathbb{B}^{n^2} \to \mathbb{B}$ *as*

$$f(x_{11}, \ldots, x_{nn}) = \bigvee_{\sigma \in S_n} \bigwedge_{i=1}^{n} x_{i\sigma(i)}.$$

We have $L(f_n) = O(n^5)$ *and* $L_u(f_n) \geq 2^{\Omega(\log^2 n)}$.

Corollary 1.5 (Using Monotone Lower Bound from [45]). *There exists a family of functions* $f_n : \mathbb{B}^{n^2} \to \mathbb{B}$ *such that* $L(f_n) = \mathrm{poly}(n)$ *and* $L_u(f_n) \geq 2^{cn^{1/3-o(1)}}$ *for a constant* $c > 0$.

In particular, there is an exponential separation between L and L_u.

Corollaries 1.4 and 1.5 are immediate applications of Theorem 1.3. Using additional techniques, we can obtain separation results even for *non-monotone* functions!

Corollary 1.6. *Let* $\det_n : \mathbb{B}^{n^2} \to \mathbb{B}$ *be the determinant over the field with 2 elements, that is,*

$$\det_n(x_{11}, \ldots, x_{nn}) = \bigoplus_{\sigma \in S_n} \prod_{i=1}^{n} x_{i\sigma(i)}.$$

We have $L(\det_n) = \mathrm{poly}(n)$ *and* $L_u(\det_n) \geq 2^{\Omega(\log^2 n)}$.

These techniques also allow us to improve the gap between Boolean and hazard-free complexity relative to Corollary 1.5.

Corollary 1.7 (Using Monotone Lower Bound from [2]). *There exists a family of functions* $f_n : \mathbb{B}^N \to \mathbb{B}$ *such that* $L(f_n) = O(N)$ *but* $L_u(f_n) \geq 2^{cN^{1/4-o(1)}}$ *for some* $c > 0$*, where the number of input variables of* f_n *is* $N = 4^n + \lfloor \frac{2^{n/2}}{4\sqrt{n}} \rfloor$.

As a final example, we state a weaker but still substantial separation result for Boolean matrix multiplication.

Corollary 1.8 (Using Monotone Lower Bound from [31, 36], See Also the Earlier [38]). *Let* $f : \mathbb{B}^{n \times n} \times \mathbb{B}^{n \times n} \to \mathbb{B}^{n \times n}$ *be the Boolean matrix multiplication map, i.e.,* $f(X, Y) = Z$ *with* $z_{i,j} = \bigvee_{k=1}^{n} x_{i,k} \wedge y_{k,j}$. *Every circuit computing* f *with fewer than* $2n^3 - n^2$ *gates has a hazard. In particular, every circuit that implements Strassen's algorithm [42] (or any of its later improvements; see e.g., [27]) in a way that can be used for subcubic Boolean matrix multiplication has a hazard.*

Since our methods are based on relabeling circuits only, analogous translations can be performed for statements about other circuit complexity measures, for example, the separation result for the circuit depth from [39]. The previously best lower bounds on the size of hazard-free circuits are restricted to depth 2 circuits (with unbounded fan-in and not counting input negations), see Section 2.

For our lower bounds, we use the strong connection between hazard-free complexity and monotone complexity, but we want to point out a conceptional difference between monotone circuits and hazard-free circuits: monotone circuits are defined syntactically, via a restriction of the available gates, whereas being hazard-free is a semantic property, that is, it refers to the properties of the function computed by a circuit.

Parametrized Upper Bound. These hardness results imply that we cannot hope for a general construction of a small hazard-free circuit for f even if $L(f)$ is small. However, the task becomes easier when restricting to hazards with a limited number of unstable input bits.

Definition 1.9 (k-bit Hazard). For a natural number k, a circuit C on n inputs has a k*-bit* hazard at $x \in \mathbb{T}^n$, iff C has a hazard at x and $\mathfrak{u}$ appears at most k times in x.

Such a restriction on the number of unstable input bits has been considered in many papers (see, e.g., [20, 47, 50, 51]), but the state-of-the-art in terms of asymptotic complexity has not improved since Huffman's initial construction [21], which is of size exponential in n, see the discussion of [43, 44] in [15, Section "Speculative Computing"]. We present a construction with blow-up exponential in k, but polynomial in n. In particular, if k is constant and $L(f_n) \in \text{poly}(n)$, this is an exponential improvement.

Corollary 1.10. *Let C be a circuit with n inputs, $|C|$ gates and depth D. Then there is a circuit with at most $(\frac{ne}{k})^{2k}(|C| + 7)$ gates and depth $D + 2k(\lceil \log n \rceil + 2)$ that computes the same function and has no k-bit hazards.*

Further Results. We round off the presentation by a number of further results. First, to further support the claim that the theory of hazards in circuits is natural, we prove that it is independent of the set of gates (`and`, `or`, `not`), as long as the set of gates is functionally complete and contains a constant function; see Corollary A.3. Second, it appears unlikely that much more than logarithmically many unstable bits can be handled with only a polynomial circuit size blow-up.

Theorem 1.11. *Fix a monotonously weakly increasing sequence of natural numbers k_n with $\log n \leq k_n$ and set $j_n := k_n / \log n$. If Boolean circuits deciding j_n-CLIQUE on graphs with n vertices require a circuit size of at least $n^{\Omega(j_n)}$, then there exists a function $f_n : \mathbb{B}^{n^2+k_n} \rightarrow \mathbb{B}$ with $L(f_n) = \text{poly}(n)$ for which circuits without k_n-bit hazards require $2^{\Omega(k_n)}$ many gates to compute.*

In particular, if $k_n = \omega(\log n)$ is only slightly superlogarithmic, then Theorem 1.11 provides a function where the circuit size blow-up is superpolynomial if we insist on having no k_n-bit hazards. In this case j_n is slightly superconstant, which means that "Boolean circuits deciding j_n-CLIQUE require size at least $n^{\Omega(j_n)}$" is a consequence of a nonuniform version of the exponential time hypothesis (see [28]), i.e., smaller circuits would be a major algorithmic breakthrough.

We remark that, although it has not been done before, deriving conditional lower bounds such as Theorem 1.11 is rather straightforward. In contrast, Theorem 1.3 yields *unconditional* lower bounds.

Finally, determining whether a circuit has a hazard is NP-complete, even for 1-bit hazards (Theorem 6.5). This matches the fact that the best algorithms for these tasks have exponential running time [13]. Interestingly, this also means that if NP ≠ coNP, given a circuit there exists no polynomial-time verifiable certificate of size polynomial in the size of the circuit to prove that the circuit is hazard-free, or even free of 1-bit hazards.

2 RELATED WORK

Multi-valued logic is a very old topic and several three-valued logic definitions exist. In 1938 Kleene defined his strong logic of indeterminacy [24, p. 153], see also his later textbook [25, Section 64]. It can be readily defined by setting $\mathfrak{u} = \frac{1}{2}$, `not` $x := 1 - x$, x `and` $y := \min(x, y)$, and x `or` $y := \max(x, y)$, as it is commonly done in fuzzy logic [37, 41]. This happens to model the behavior of physical Boolean gates and can be used to formally define hazards. This was first realized by Goto in [18, p. 128], which is the first paper that contains a hazard-free implementation of the multiplexer; see [18, Figure 7·5]. The third truth value in circuits was mentioned one year earlier in [17]. As far as we know, this early Japanese work was unnoticed in the Western world at first. The first structural results on hazards appeared in a seminal paper by Huffman [21], who proved that every Boolean function has a hazard-free circuit. This is also the first paper that observes the apparent circuit size blow-up that occurs when insisting on a hazard-free implementation of a function. Huffman mainly focused on 1-bit hazards, but notes that his methods carry over to general hazards. Interestingly, our Corollary 1.10 shows that for 1-bit hazards the circuit size blow-up is polynomially bounded, while for general hazards, we get the strong separation of Corollary 1.5.

The importance of hazard-free circuits is already highlighted, for example, in the classical textbook [9]. Three-valued logic for circuits was introduced by Yoeli and Rinon in [50]. In 1965, Eichelberger published the influential paper [13], which shows how to use three-valued logic to detect hazards in exponential time. This paper also contains the first lower bound on hazard-free depth 2 circuits: A hazard-free and-or circuit with negations at the inputs must have at least as many gates as its function has prime implicants, which can be an exponentially large number; see, e.g., [10]. Later work on lower bounds was also only concerned with depth 2 circuits, for example, [35].

Mukaidono [32] was the first to formally define a partial order of definedness, see also [33, 34], where it is shown that a ternary function is computable by a circuit iff it is monotone under this partial order. In 1981, Marino [30] used a continuity argument to show (in a more general context) that specific ternary functions cannot be implemented; for example, there is no circuit that implements the detector function $f(\mathfrak{u}) = 1$, $f(0) = f(1) = 0$.

Nowadays the theory of three-valued logic and hazards can be found, for example, in [7]. A fairly recent survey on multi-valued logic and hazards is given in [5].

Recent work models clocked circuits [16]. Applying the standard technique of "unrolling" a clocked circuit into a combinational circuit, one sees that the computational power of clocked and unclocked circuits is the same. Moreover, lower and upper bounds translate between the models as expected; using r rounds of computation changes circuit size by a factor of at most r. However, [16] also models a special type of registers, masking registers, that have the property that if they output $\mathfrak{u}$ when being read in clock cycle r, they output a stable value in all subsequent rounds (until written to again). With these registers, each round of computation enables computing strictly more (ternary) functions. Interestingly, adding masking registers also breaks the relation between hazard-free and monotone complexity: [16] presents a transformation that trades a factor $O(k)$ blow-up in circuit size for eliminating k-bit hazards. In particular, choosing $k = n$, a linear blow-up suffices to construct a hazard-free circuit out of an arbitrary hazardous implementation of a Boolean function.

Seemingly unrelated, in 2009 a cybersecurity paper [46] was published that studies information flow on the Boolean gate level. The logic of the information flow happens to be Kleene's logic and thus results transfer in both directions. In particular (using different nomenclature), they design a circuit (see [46, Figure 2]) that computes the Boolean derivative, very similar to our construction in Proposition 4.8. In the 2012 follow-up paper [20], the construction of this circuit is monotone (see [20, Figure 1]), which is a key property that we use in our main structural correspondence result Theorem 1.3.

There is an abundance of monotone circuit lower bounds that all translate to hazard-free complexity lower bounds, for example, [1, 39, 40, 49] and references in [19] for general problems but also [48] and references therein for explicit problems, [31, 36, 38] for matrix multiplication, and [4] for the Boolean convolution map. This last reference also implies that any implementation of the Fast Fourier Transform to solve Boolean convolution must have hazards.

3 DEFINITIONS

We study functions $F : \mathbb{T}^n \to \mathbb{T}$ that can be implemented by circuits. The Boolean analogue is just the set of *all* Boolean functions. In our setting this is more subtle. First, if a circuit gets a Boolean input, then by the definition of the gates it also outputs a Boolean value. Thus, every function that is computed by circuits *preserves stable values*, i.e., yields a Boolean value on a Boolean input. Now, we equip $\mathbb{T}$ with a partial order $\preceq$ such that $\mathfrak{u}$ is the least element and 0 and 1 are incomparable elements greater than $\mathfrak{u}$; see [32]. We extend this order to $\mathbb{T}^n$ in the usual way. For tuples $x, y \in \mathbb{T}^n$ the statement $x \preceq y$ means that y is obtained from x by replacing some unstable values with stable ones. Since the gates and, or, not are monotone with respect to $\preceq$, every function F computed by a circuit must be monotone with respect to $\preceq$. It turns out that these two properties capture precisely what can be computed:

Proposition 3.1 [32, Theorem 3]. *A function $F : \mathbb{T}^n \to \mathbb{T}$ can be computed by a circuit iff F preserves stable values and is monotone with respect to $\preceq$.*

A function $F : \mathbb{T}^n \to \mathbb{T}$ that preserves stable values and is monotone with respect to $\preceq$ shall be called a *natural function*. A function $F : \mathbb{T}^n \to \mathbb{T}$ is called an *extension* of a Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ if the restriction $F|_{\mathbb{B}^n}$ coincides with f.

Observe that any natural extension F of a Boolean function f must satisfy the following. If y and y' are resolutions of x (in particular $x \preceq y$ and $x \preceq y'$) such that $F(y) \neq F(y')$, then it must hold that $F(y) = 0$ and $F(y') = 1$ (or vice versa), due to preservation of stable values. By $\preceq$-monotonicity, this necessitates that $F(x) = \mathfrak{u}$, the only value "smaller" than both 0 and 1. Thus, one cannot hope for a stable output of a circuit if x has two resolutions with different outputs. In contrast, if all resolutions of x produce the same output, we can require a stable output for x, i.e., that a circuit computing F is hazard-free.

Definition 3.2. For a Boolean function $f : \mathbb{B}^n \to \mathbb{B}$, define its *hazard-free extension* $\bar{f} : \mathbb{T}^n \to \mathbb{T}$ as follows:

$$\bar{f}(x) = \begin{cases} 0, & \text{if } f(y) = 0 \text{ for all resolutions } y \text{ of } x, \\ 1, & \text{if } f(y) = 1 \text{ for all resolutions } y \text{ of } x, \\ \mathfrak{u}, & \text{otherwise.} \end{cases}$$

Hazard-free extensions are natural functions and are exactly those functions that are computed by hazard-free circuits, as can be seen, for example, by Theorem 1.2. Equivalently, $\bar{f}$ is the unique extension of f that is monotone and maximal with respect to $\preceq$.

We remark that later on we will also use the usual order $\leq$ on $\mathbb{B}$ and $\mathbb{B}^n$. We stress that the term *monotone Boolean function* refers to functions $\mathbb{B}^n \to \mathbb{B}$ monotone with respect to $\leq$.

4 LOWER BOUNDS ON THE SIZE OF HAZARD-FREE CIRCUITS

In this section, we present a number of lower bound results. As a warm-up, we first prove the conditional lower bound Theorem 1.11, which is a direct consequence of the upcoming Proposition 4.1 and noting that $n^{\Omega(k_n/\log n)} = 2^{\Omega(k_n)}$. We then discuss the main result $L_{\mathfrak{u}}(f) = L_+(f)$ for monotone functions f and how Corollaries 1.4–1.8 follow from it.

Conditional Lower Bound

Proposition 4.1. *Fix a monotonously weakly increasing sequence of natural numbers j_n with $j_n \leq n$. There is a function $f_n : \mathbb{B}^{\binom{n}{2}+j_n \log n} \to \mathbb{B}$ with $L(f_n) = \text{poly}(n)$ and the following property: if f_n can be computed by circuits of size L_n that are free of $(j_n \log n)$-bit hazards, then there are Boolean circuits of size $2L_n$ that decide j_n-CLIQUE.*

Proof. The function f_n gets as input the adjacency matrix of a graph G on n vertices and a list ℓ of j_n vertex indices, each encoded in binary with $\log n$ many bits:

$$f_n(G,\ell) = \begin{cases} 1 & \text{if } \ell \text{ encodes a list of } j_n \text{ vertices that form a } j_n\text{-clique in } G, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, $L(f_n) = \text{poly}(n)$. Let C compute f_n and have no $(j_n \log n)$-hazards. By the definition of $(j_n \log n)$-hazards, it follows that $C(G, \mathfrak{u}^{j_n \log n}) \neq 0$ iff G contains a j_n-clique. From C, we construct a circuit C' that decides j_n-CLIQUE as follows. We double each gate and each wire. Additionally, after each doubled not-gate, we twist the two wires so that this not construction sends $(0,1)$ to $(0,1)$ instead of to $(1,0)$. Stable inputs to C are doubled, whereas the input $\mathfrak{u}$ is encoded as the Boolean pair $(0,1)$. It is easy to see that the resulting circuit simulates C. Our circuit C' should have $\binom{n}{2}$ inputs and should satisfy $C'(G) = 1$ iff $C(G, \mathfrak{u}^{j_n \log n}) \neq 0$; thus, we fix the $j_n \log n$ rightmost input pairs to constants $(0,1)$ to obtain C'. From the two output gates, we treat the right output gate as the output of C', while dismissing the left output gate. □

Monotone Circuits are Hazard-free

Our first step towards $L_{\mathfrak{u}}(f) = L_+(f)$ is to show that $L_{\mathfrak{u}}(f) \leq L_+(f)$.

Lemma 4.2. *Monotone circuits are hazard-free. In particular, for monotone Boolean functions f, we have $L_{\mathfrak{u}}(f) \leq L_+(f)$.*

Proof. We prove the claim by induction over the number of computation gates in the circuit. Trivially, a monotone circuit without computation gates is hazard-free, as it merely forwards some input to the output. For the induction step, let C be a monotone circuit computing a natural function $F : \mathbb{T}^n \to \mathbb{T}$ such that the gate computing the output of C receives as inputs the outputs of two hazard-free monotone subcircuits C_1 and C_2. We denote by F_1 and F_2 the natural functions computed by C_1 and C_2, respectively. The gate computing the output of C can be an and- or an or-gate, and we will treat both cases in parallel. Let $x \in \mathbb{T}^n$ be arbitrary with the property that $F(y) = 1$ for all resolutions y of x. Denote by y_0 the resolution of x in which all $\mathfrak{u}$'s are replaced by 0. The fact that $F(y_0) = 1$ implies that $F_1(y_0) = F_2(y_0) = 1$ ($F_1(y_0) = 1$ or $F_2(y_0) = 1$). Since the restrictions of F_1 and F_2 to $\mathbb{B}^n$ are monotone Boolean functions, this extends from y_0 to all resolutions y of x, because $y \geq y_0$ and thus $F(y) \geq F(y_0) = 1$. Since C_1 and C_2 are hazard-free by the induction hypothesis, we have $F_1(x) = F_2(x) = 1$ ($F_1(x) = 1$ or $F_2(x) = 1$). As basic gates are hazard-free, we conclude that $F(x) = 1$.

The case that $F(y) = 0$ for all resolutions y of some $x \in \mathbb{T}^n$ is analogous, where y_0 is replaced by y_1, the resolution of x in which all $\mathfrak{u}$'s are replaced by 1. □

The following sections show a much deeper relationship between monotone and hazard-free circuits. A key concept is the derivative, which we will discuss next.

Derivatives of Natural Functions

In this section, we introduce the hazard derivative, a key ingredient for showing $L_{\mathfrak{u}}(f) \geq L_+(f)$. Let $F : \mathbb{T}^n \to \mathbb{T}$ be a natural function and $x \in \mathbb{B}^n$ be a stable input. If $\tilde{x} \preceq x$, that is, if $\tilde{x}$ is obtained from x by replacing stable bits by $\mathfrak{u}$, then $F(\tilde{x}) \preceq F(x)$. This means that there are two possibilities for $F(\tilde{x})$ — either $F(\tilde{x}) = F(x)$ or $F(\tilde{x}) = \mathfrak{u}$.

We can encode in one Boolean function the information about how the value of F changes from $F(x)$ to $\mathfrak{u}$ when the bits of the input change from stable to unstable. It is reminiscent of the idea of the derivative in analysis or the Boolean derivative, which also show how the value of the function changes when the input changes. To make the connection more apparent, we introduce a notation for replacing stable bits by unstable ones: if $x, y \in \mathbb{B}^n$, then $x + \mathfrak{u}y$ denotes the tuple that is obtained from x by changing the values to $\mathfrak{u}$ in all positions in which y has a 1 and keeping the other values unchanged. Formally,

$$\tilde{x} = x + \mathfrak{u}y \quad \Leftrightarrow \quad \tilde{x}_i = \begin{cases} x_i, & \text{if } y_i = 0, \\ \mathfrak{u}, & \text{if } y_i = 1. \end{cases}$$

This notation is consistent with interpreting the addition and multiplication on $\mathbb{T}$ as the hazard-free extensions of the usual addition modulo 2 and multiplication on $\mathbb{B}$ (`xor` and `and`).

Any tuple $\tilde{x} \preceq x$ can be presented as $x + \mathfrak{u}y$ for some $y \in \mathbb{B}^n$. As we have seen, $F(x + \mathfrak{u}y)$ is either $F(x)$ or $\mathfrak{u}$. This condition can also be written as $F(x + \mathfrak{u}y) = F(x) + \mathfrak{u}\Delta$ for some $\Delta \in \mathbb{B}$.

Definition 4.3. Let $F : \mathbb{T}^n \to \mathbb{T}$ be a natural function. The *hazard derivative* (or just *derivative* for short) of F is the Boolean function $\mathrm{d}F : \mathbb{B}^n \times \mathbb{B}^n \to \mathbb{B}$ such that

$$F(x + \mathfrak{u}y) = F(x) + \mathfrak{u} \cdot \mathrm{d}F(x; y). \tag{4.1}$$

In other words,

$$\mathrm{d}F(x; y) = \begin{cases} 0, & \text{if } F(x + \mathfrak{u}y) = F(x), \\ 1, & \text{if } F(x + \mathfrak{u}y) = \mathfrak{u}. \end{cases}$$

For a Boolean function f, we use the shorthand notation $\mathrm{d}f := \mathrm{d}\overline{f}$.

Consider, for example, the disjunction `or`. The values of $(x_1 + \mathfrak{u}y_1)$`or`$(x_2 + \mathfrak{u}y_2)$ are as follows:

`or`	$0 + \mathfrak{u} \cdot 0$	$0 + \mathfrak{u} \cdot 1$	$1 + \mathfrak{u} \cdot 0$	$1 + \mathfrak{u} \cdot 1$
$0 + \mathfrak{u} \cdot 0$	0	$\mathfrak{u}$	1	$\mathfrak{u}$
$0 + \mathfrak{u} \cdot 1$	$\mathfrak{u}$	$\mathfrak{u}$	1	$\mathfrak{u}$
$1 + \mathfrak{u} \cdot 0$	1	1	1	1
$1 + \mathfrak{u} \cdot 1$	$\mathfrak{u}$	$\mathfrak{u}$	1	$\mathfrak{u}$

Thus,

$$\mathtt{dor}(x_1, x_2; y_1, y_2) = \neg x_1 y_2 \vee \neg x_2 y_1 \vee y_1 y_2. \tag{4.2a}$$

Similarly, we find

$$\mathtt{dnot}(x; y) = y, \tag{4.2b}$$

$$\mathtt{dand}(x_1, x_2; y_1, y_2) = x_1 y_2 \vee x_2 y_1 \vee y_1 y_2, \tag{4.2c}$$

$$\mathtt{dxor}(x_1, x_2; y_1, y_2) = y_1 \vee y_2. \tag{4.2d}$$

Caveat: Since natural functions F are exactly those ternary functions defined by circuits, we can obtain $\mathrm{d}F$ from the ternary evaluations of any circuit computing F. For Boolean functions f, it is more natural to think of $\mathrm{d}f$ as a property of the function f, because the correspondence to circuits is not as close: we can obtain $\mathrm{d}f$ from the ternary evaluations of any *hazard-free* circuit computing f on Boolean inputs.

In general, we can find the derivative of a Boolean function as follows:

LEMMA 4.4. *For* $f : \mathbb{B}^n \to \mathbb{B}$, *we have* $\mathrm{d}f(x; y) = \bigvee_{z \le y} [f(x) + f(x + z)]$. *In particular, if* $f(0) = 0$, *then* $\mathrm{d}f(0; y) = \bigvee_{z \le y} f(z)$.

PROOF. Resolutions of $x + \mathfrak{u}y$ coincide with x at positions where y has a 0 and have arbitrary stable bits at positions where y has a 1. Therefore, each resolution of $x + \mathfrak{u}y$ can be presented as $x + z$ for some z such that $z_i = 0$ whenever $y_i = 0$, that is, $z \le y$. Hence, the set of all resolutions of $x + \mathfrak{u}y$ is $S(x + \mathfrak{u}y) := \{x + z \mid z \le y\}$.

The derivative $\mathrm{d}f(x; y) = 1$ if and only if $\bar{f}(x + \mathfrak{u}y) = \mathfrak{u}$. By definition of hazard-freeness, this happens when f takes both values 0 and 1 on $S(x + \mathfrak{u}y)$, in other words, when the

$f(x+z) \neq f(x)$ for some $z \in S(x+\mathfrak{u}y)$. The disjunction $\bigvee_{z \leq y}[f(z)+f(x+z)]$ represents exactly this statement. □

As a corollary, we obtain a surprisingly close relation between monotone Boolean functions and their derivatives. For a natural function F and any fixed $x \in \mathbb{B}^n$, let $\mathrm{d}F(x;.)$ denote the Boolean function that maps $y \in \mathbb{B}^n$ to $\mathrm{d}F(x;y)$, and define the shorthand $\mathrm{d}f(x;.) := \mathrm{d}\overline{f}(x;.)$ for a Boolean function f.

Corollary 4.5. *Suppose that $f : \mathbb{B}^n \to \mathbb{B}$ is monotone with $f(0) = 0$. Then $\mathrm{d}f(0,.) = f$.*

Lemma 4.6. *For natural $F : \mathbb{T}^n \to \mathbb{T}$ and fixed $x \in \mathbb{B}^n$, $\mathrm{d}F(x;.)$ is a monotone Boolean function.*

Proof. Note that the expression $x + \mathfrak{u}y$ is antimonotone in y: if $y_1 \geq y_2$, i.e., y_1 is obtained from y_2 by replacing 0s with 1s, then $x + \mathfrak{u}y_1$ is obtained from $x + \mathfrak{u}y_2$ by replacing more stable bits of x with $\mathfrak{u}$, so $x + \mathfrak{u}y_1 \preceq x + \mathfrak{u}y_2$. Thus, if $y_1 \geq y_2$, then F being natural yields that

$$F(x) + \mathfrak{u}\,\mathrm{d}F(x;y_1) = F(x+\mathfrak{u}y_1) \preceq F(x+\mathfrak{u}y_2) = F(x) + \mathfrak{u}\,\mathrm{d}F(x;y_2),$$

so $\mathrm{d}F(x;y_1) \geq \mathrm{d}F(x;y_2)$. □

We can also define derivatives for vector functions $F : \mathbb{T}^n \to \mathbb{T}^m$, $F(x) = (F_1(x), \dots, F_m(x))$ with natural components $F_1, \dots, F_m$ as $\mathrm{d}F(x;y) = (\mathrm{d}F_1(x;y), \dots, \mathrm{d}F_m(x;y))$. Note that Equation (4.1) still holds and uniquely defines the derivative for vector functions.

The following statement is the analogue of the chain rule in analysis.

Lemma 4.7 (Chain Rule). *Let $F : \mathbb{T}^n \to \mathbb{T}^m$ and $G : \mathbb{T}^m \to \mathbb{T}^l$ be natural functions and $H(x) = G(F(x))$. Then*

$$\mathrm{d}H(x;y) = \mathrm{d}G(F(x);\mathrm{d}F(x;y)).$$

Proof. Use Equation (4.1).

$$\begin{aligned} H(x+\mathfrak{u}y) &= G(F(x+\mathfrak{u}y)) = G(F(x) + \mathfrak{u}\,\mathrm{d}F(x;y)) = G(F(x)) + \mathfrak{u}\,\mathrm{d}G(F(x);\mathrm{d}F(x;y)) \\ &= H(x) + \mathfrak{u}\,\mathrm{d}G(F(x);\mathrm{d}F(x;y)), \end{aligned}$$

and the claim follows with another application of Equation (4.1). □

Using Monotone Circuits to Compute Derivatives

In this section, we show how to efficiently compute derivatives by transforming circuits to monotone circuits. Our main tool is the chain rule (Lemma 4.7).

For a circuit C and a gate β of C, let C_β denote the natural function computed at the gate β. From a circuit C, we now construct a circuit C' by independently replacing each gate β on t inputs $\alpha_1, \dots, \alpha_t$ ($0 \leq t \leq 2$) by a subcircuit on $2t$ inputs $\alpha_1, \dots, \alpha_t, \alpha'_1, \dots, \alpha'_t$ and two output gates β, β' (the wiring between these subcircuits in C' is the same as the wiring between the gates in C, but in C' we have two parallel wires for each wire in C). The goal is that $C'_\beta(x,y) = C_\beta(x)$ and $C'_{\beta'}(x,y) = \mathrm{d}C_\beta(x;y)$ for Boolean inputs $x, y \in \mathbb{B}^n$, see the upcoming Proposition 4.8.

To construct C', we extend C with new gates. For each gate β in C, we add a new gate β'. If β is an input gate x_i, then β' is the input gate y_i. If β is a constant gate, then β' is the constant-0 gate.

The most interesting case is when β is a gate implementing a function $\varphi \in \{\mathtt{and}, \mathtt{or}, \mathtt{not}\}$ with incoming edges from gates $\alpha_1, \dots, \alpha_t$ (in our definition of the circuit, the arity t is 1 or 2, but the construction works without modification in the general case). In this case, we add to β a subcircuit that takes $\alpha_1, \dots, \alpha_t$ and their counterparts $\alpha'_1, \dots, \alpha'_t$ as inputs and β' as its output gate, which computes $C'_{\beta'}(x,y) = \mathrm{d}\varphi(C'_{\alpha_1}(x,y), \dots, C'_{\alpha_t}(x,y); C'_{\alpha'_1}(x,y), \dots, C'_{\alpha'_t}(x,y))$. For the sake of

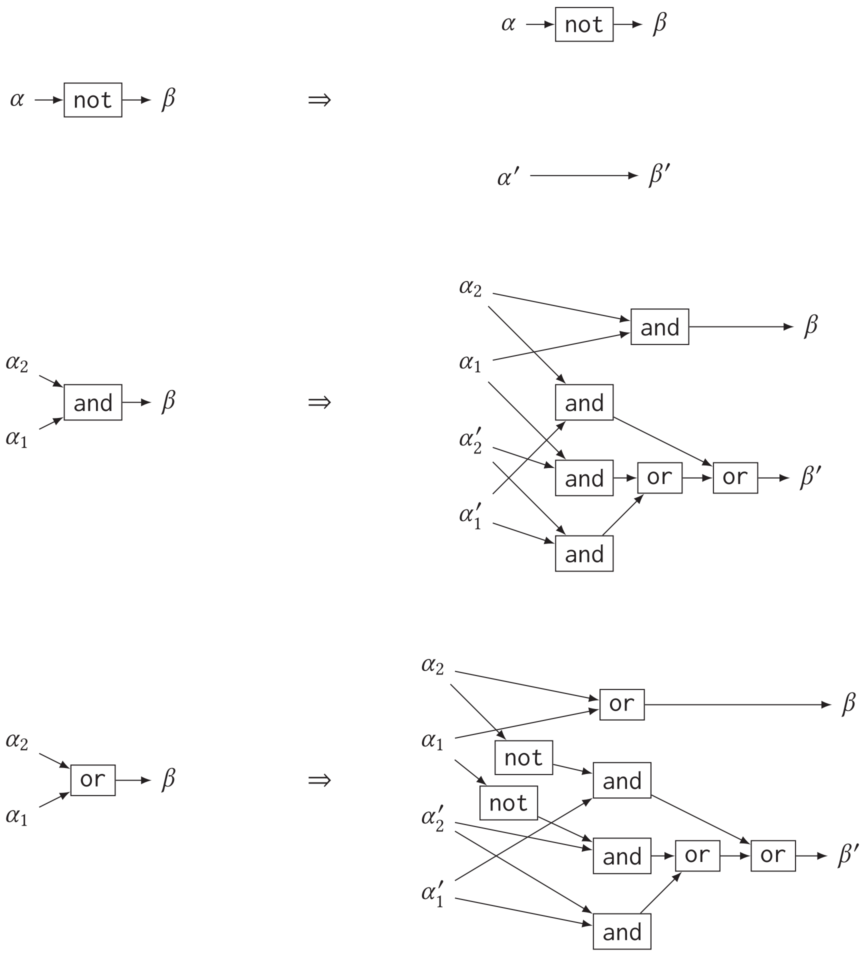


Fig. 2. Gates in C get replaced by subcircuits in the construction of C'.

concreteness, for the gate types not, and, or according to Equation (4.2) this construction is depicted in Figure 2.

Proposition 4.8. *$C'_\beta(x,y) = C_\beta(x)$ and $C'_{\beta'}(x,y) = \mathrm{d}C_\beta(x;y)$ for Boolean inputs $x, y \in \mathbb{B}^n$.*

Proof. Clearly $C'_\beta(x,y) = C_\beta(x)$. By induction on the structure of the circuit, we now prove that $C'_{\beta'}(x,y) = \mathrm{d}C_\beta(x;y)$. In the base case, if β is an input or constant gate, the claim is obvious. If β is a gate of type $\varphi \in \{\mathsf{and}, \mathsf{or}, \mathsf{not}\}$ with incoming edges from $\alpha_1, \ldots, \alpha_t$, then

$$C_\beta(x) = \varphi(C_{\alpha_1}(x), \ldots, C_{\alpha_t}(x)).$$

By the chain rule,

$$\mathrm{d}C_\beta(x;y) = \mathrm{d}\varphi(C_{\alpha_1}(x), \ldots, C_{\alpha_t}(x); \mathrm{d}C_{\alpha_1}(x;y), \ldots, \mathrm{d}C_{\alpha_t}(x;y)).$$

By the induction hypothesis, $(\alpha'_1, \ldots, \alpha'_t) = (\mathrm{d}C_{\alpha_1}(x;y), \ldots, \mathrm{d}C_{\alpha_t}(x;y))$; thus, the induction step succeeds by construction of $C'_{\beta'}$. □

Note that this construction can be seen as simulation of the behavior of the circuit C on the input $x + \mathfrak{u}y$: the value computed at the gate β on this input is $C_\beta(x) + \mathfrak{u}\,\mathrm{d}C_\beta(x;y)$, and in C' the gates β and β' compute the two parts of this expression separately.

The construction can be seen as an adaptation of the technique of automatic differentiation, an operator-wise transformation of a program computing a function to a program computing its derivative. The idea is simple and dates back to the 1950s [8]. A theoretically inclined reader may be more familiar with the theorem of Baur and Strassen on computing all derivatives of a multivariate rational function given by an algebraic circuit [3], which provides foundations of an alternative method—reverse mode automatic differentiation.

By fixing the first half of the input bits in C', we now establish the link to monotone complexity. In the following theorem the case $x = 0$ will be of particular interest.

Theorem 4.9. *For $f : \mathbb{B}^n \to \mathbb{B}$ and fixed $x \in \mathbb{B}^n$, it holds that $L_+(\mathrm{d}f(x,.)) \leq L_{\mathfrak{u}}(f)$.*

Proof. Let C be a hazard-free circuit for f of minimal size and let $x \in \mathbb{B}^n$ be fixed. We start by constructing the circuit C' from Proposition 4.8, and for each gate in C we remember the corresponding subcircuit in C'. For each subcircuit, we call the gates α_i the *primary inputs* and the α_i' the *secondary inputs*. From C', we now construct a monotone circuit C^x on n inputs that computes $\mathrm{d}f(x;.)$ as follows. We fix the leftmost n input bits $x \in \mathbb{B}^n$ in C'. This assigns a Boolean value $C'_\alpha(x) = C_\alpha(x)$ to each primary input α in each constructed subcircuit. Each constructed subcircuit's secondary output β' now computes some Boolean function in the secondary inputs α_i'. If the values at the secondary inputs are $u_1 = C'_{\alpha_1'}(x,y), \dots, u_t = C'_{\alpha_t'}(x,y)$, then the value at the secondary output is $\psi(u_1,\dots,u_t) = \mathrm{d}\varphi(C_{\alpha_1}(x),\dots,C_{\alpha_t}(x);u_1,\dots,u_t)$. Lemma 4.6 implies that ψ is monotone (which can alternatively be seen directly from Figure 2, where fixing all primary inputs makes all not gates superfluous). However, the only monotone functions on at most two input bits are the identity (on one input), and, or, and the constants. Thus, we can replace each subcircuit in C' by (at most) one monotone gate, yielding the desired monotone circuit C^x that has at most as many gates as C and outputs $\mathrm{d}f(x;.) = \mathrm{d}\bar{f}(x;.) = \mathrm{d}C(x;.) = C'(.)$, where the second equality holds because C is hazard-free. □

We now use this construction to prove Theorem 1.3.

Proof of Theorem 1.3. The claim is trivial for the constant 1 function. Note that this is the only case of a monotone function that has $f(0) \neq 0$. Hence, assume that f is monotone with $f(0) = 0$. By Lemma 4.2, we have that $L_{\mathfrak{u}}(f) \leq L_+(f)$. The other direction can be seen via $L_+(f) \overset{\text{Cor. 4.5}}{=} L_+(\mathrm{d}f(0,.)) \overset{\text{Thm. 4.9}}{\leq} L_{\mathfrak{u}}(f)$. □

Theorem 1.3 shows that the hazard-free complexity $L_{\mathfrak{u}}$ can be seen as an extension of monotone complexity L_+ to general Boolean functions. Thus, known results about the gap between general and monotone complexity transfer directly to hazard-free complexity.

Unconditional Lower Bounds

Corollaries 1.4, 1.5, and 1.8 are immediate applications of Theorem 1.3. Interestingly, however, we can also derive results on *non-monotone* functions, which is illustrated by Corollary 1.6.

Proof of Corollary 1.6. The fact that the determinant can be computed efficiently is well known.

Consider the derivative $\mathrm{d}\det_n(0;y) = \bigvee_{z \leq y} \det_n(z)$ (Lemma 4.4). If there exists a permutation $\pi \in S_n$ such that all $y_{i\pi(i)}$ are 1, then, replacing all the other entries with 0, we get a matrix $z \leq y$ with $\det_n(z) = 1$, and $\mathrm{d}\det_n(0;y) = 1$. If there is no such permutation, then all the summands in the definition of $\det_n(y)$ are 0, and this is also true for all matrices $z \leq y$. In this case, $\mathrm{d}\det_n(0; y) = 0$. Combining both cases, we get that $\mathrm{d}\det_n(0; .)$ equals the Boolean permanent function f_n from Corollary 1.4. The lower bound then follows from [40] and Theorem 4.9 (as in Corollary 1.4). □

We can combine this technique with the ideas from the proof of Theorem 1.11 to show even stronger separation results, exhibiting a family of functions for which the complexity of Boolean circuits is *linear,* yet the complexity of hazard-free circuits grows almost as fast as in Corollary 1.5.

Lemma 4.10. *Let $f : \mathbb{B}^n \to \mathbb{B}$ be a monotone Boolean function with $f(0) = 0$ and $g : \mathbb{B}^{n+m} \to \mathbb{B}$ be a function such that $f(x) = 1$ iff $g(x, y) = 1$ for some $y \in \mathbb{B}^m$. Then $L_+(f) \leq L_{\mathrm{u}}(g)$.*

Proof. Using Lemma 4.4, we obtain

$$\mathrm{d}g(0, 0; x, 1) = \bigvee_{(z,t) \leq (x,1)} g(z, t) = \bigvee_{z \leq x} \bigvee_{t} g(z, t) = \bigvee_{z \leq x} f(z) = f(z),$$

which means that the circuit for f can be obtained from the circuit for $\mathrm{d}g(0; .)$ by substituting 1 for some inputs. The statement then follows from Theorem 4.9. □

Proof of Corollary 1.7. We use the NP-complete family $\mathrm{POLY}(q, s)$ from the paper of Alon and Boppana [2]. Let $\mathrm{GF}(q)$ denote a finite field with q elements. We encode subsets $E \subset \mathrm{GF}(q)^2$ using q^2 Boolean variables in a straightforward way. The function $\mathrm{POLY}(q, s)$ maps $E \subset \mathrm{GF}(q)^2$ to 1 iff there exists a polynomial p of degree at most s over $\mathrm{GF}(q)$ such that $(a, p(a)) \in E$ for every $a \in \mathrm{GF}(q)$.

Alon and Boppana proved that for $s \leq \frac{1}{2}\sqrt{\frac{q}{\ln q}}$ the monotone complexity of this function is at least q^{cs} for some constant c. For simplicity, we choose $q = 2^n$ and $s = \lfloor \frac{1}{4}\sqrt{\frac{q}{\log q}} \rfloor = \lfloor \frac{2^{n/2}}{4\sqrt{n}} \rfloor$. In this case, $L_+(\mathrm{POLY}(q, s)) \geq 2^{cq^{1/2}\sqrt{\log q}}$.

We define f_n as the verifier for this instance of POLY. The function f_n takes $q^2 + sq = O(q^2)$ variables. The first q^2 inputs encode a subset $E \subset \mathrm{GF}(q)^2$, and the second sn inputs encode coefficients of the polynomial p of degree at most s over $\mathrm{GF}(q)$, each coefficient using n bits. The value $f_n(E, p) = 1$ iff $(a, p(a)) \in E$ for all $a \in \mathrm{GF}(q)$. To implement the function f_n, for each element $a \in \mathrm{GF}(q)$, we compute the value $p(a)$ using finite field arithmetic. Each such computation requires $O(sn^2)$ gates. Then, we use $p(a)$ as a selector in a multiplexer to compute the value indicating whether $(a, p(a))$ is contained in E, choosing it from all the bits of the input E corresponding to pairs of form (a, b). This multiplexer requires additional $O(q)$ gates for each element $a \in \mathrm{GF}(q)$. The result is the conjunction of the computed values for all $a \in \mathrm{GF}(q)$. The total size of the circuit $O(q^2 + qsn^2 + q)$ is linear in the size of the input.

The lower bound on the hazard-free complexity follows from the Alon-Boppana lower bound and Lemma 4.10. □

5 CONSTRUCTING *K*-BIT HAZARD-FREE CIRCUITS

In this section, we prove Corollary 1.10. We slightly improve on [22].

For a collection T of subsets of $[n]$, denote by $L_T(f)$ the minimum size of a circuit whose outputs coincide with $\bar{f}$ whenever the set of input positions with unstable bits is a subset of a set in the collection T. Thus, $\leq$-monotonicity of natural functions implies that $L(f) = L_\emptyset(f) \leq L_T(f) \leq L_{\{[n]\}}(f) = L_{\mathrm{u}}(f)$. Excluding k-bit hazards therefore means that we consider $T = \binom{[n]}{k}$, i.e., T contains all subsets of $[n]$ with exactly k elements. The minimum circuit depth $D_T(f)$ is defined analogously.

As the base case of our construction, we construct circuits handling only fixed positions for the (up to) k unstable bits, i.e., $T = \{S\}$ for some $S \in \binom{[n]}{k}$. This is straightforward with an approach very similar to speculative computing [43, 44].

We take 2^k copies of a circuit computing f. In the ith copy ($0 \leq i < 2^k$), we fix the inputs in S to the binary representation of i. Now, we use a hazard-free multiplexer to select one of these 2^k outputs, where the original input bits from S are used as the select bits. A hazard-free k-bit multiplexer of size $O(2^k)$ can be derived from the 1-bit construction given in Figure 1(b).

Lemma 5.1. *A k-bit multiplexer $\mathtt{MUX}_k$ receives inputs $x \in \mathbb{B}^{2^k}$ and $s \in \mathbb{B}^k$. It interprets s a number from $[2^k]$ and outputs x_s. There is a hazard-free circuit for $\mathtt{MUX}_k$ of size $6(2^k - 1)$ and depth $4k$.*

Proof. A hazard-free $\mathtt{MUX}_1$ of size 6 and depth 4 is given in Figure 1(b); its correctness is verified by a simple case analysis. From a hazard-free $\mathtt{MUX}_k$ and the hazard-free $\mathtt{MUX}_1$, we construct a hazard-free $\mathtt{MUX}_{k+1}$ circuit C as follows:

$$\begin{aligned}\mathtt{MUX}_{k+1}(x_1, \ldots, x_{2^{k+1}}; s_1, \ldots, s_{k+1}) = \mathtt{MUX}_1(\ & \mathtt{MUX}_k(x_1, \ldots, x_{2^k}; s_1, \ldots, s_k), \\ & \mathtt{MUX}_k(x_{2^k+1}, \ldots, x_{2^{k+1}}; s_1, \ldots, s_k); \quad s_{k+1}).\end{aligned}$$

One can readily verify that the resulting Boolean function is $\mathtt{MUX}_k$, and it has the desired circuit size and depth by construction. To show that this circuit for $\mathtt{MUX}_{k+1}$ is hazard-free, we make a case distinction.

If s_{k+1} is stable, w.l.o.g. $s_{k+1} = 0$, then C outputs $\mathtt{MUX}_k(x_1, \ldots, x_{2^k}; s_1, \ldots, s_k)$, since $\mathtt{MUX}_1$ is hazard-free. Thus, if $\mathtt{MUX}_{k+1}$ has a hazard at $(x_1, \ldots, x_{2^{k+1}}; s_1, \ldots, s_k, 0)$, then $\mathtt{MUX}_k$ has a hazard at $(x_1, \ldots, x_{2^k}; s_1, \ldots, s_k)$. But by the induction hypothesis, $\mathtt{MUX}_k$ is hazard-free.

Now, we consider the case $s_{k+1} = \mathfrak{u}$. For the sake of contradiction, assume that $\mathtt{MUX}_{k+1}$ has a hazard at $(x_1, \ldots, x_{2^{k+1}}; s_1, \ldots, s_k, \mathfrak{u})$. Then all resolutions $(x'_1, \ldots, x'_{2^{k+1}}; s'_1, \ldots, s'_k, s'_{k+1}) \in \mathbb{B}^{2^{k+1}+k+1}$ of $(x_1, \ldots, x_{2^{k+1}}; s_1, \ldots, s_k, \mathfrak{u})$ yield $\mathtt{MUX}_{k+1}(x'_1, \ldots, x'_{2^{k+1}}; s'_1, \ldots, s'_k, s'_{k+1}) = b$ for the same $b \in \mathbb{B}$. By construction of C this implies that

$$\mathtt{MUX}_k(x'_1, \ldots, x'_{2^k}; s'_1, \ldots, s'_k) = b = \mathtt{MUX}_k(x'_{2^k+1}, \ldots, x'_{2^{k+1}}; s'_1, \ldots, s'_k)\,.$$

By the induction hypothesis, $\mathtt{MUX}_k$ is hazard-free. Thus, we can conclude that

$$\mathtt{MUX}_k(x_1, \ldots, x_{2^k}; s_1, \ldots, s_k) = b = \mathtt{MUX}_k(x_{2^k+1}, \ldots, x_{2^{k+1}}; s_1, \ldots, s_k)\,.$$

This implies $\mathtt{MUX}_{k+1}(x_1, \ldots, x_{2^{k+1}}; s_1, \ldots, s_k, \mathfrak{u}) = b$, because $\mathtt{MUX}_1$ is hazard-free. This is a contradiction to $\mathtt{MUX}_{k+1}$ having a hazard at $(x_1, \ldots, x_{2^{k+1}}; s_1, \ldots, s_k, \mathfrak{u})$.

Putting both cases together, we conclude that $\mathtt{MUX}_{k+1}$ is hazard-free. □

Lemma 5.2. *Let $f : \mathbb{B}^n \to \mathbb{B}$ and $S \subseteq [n]$ with $|S| = k$. Then $L_{\{S\}}(f) < 2^k(L(f) + 6)$ and $D_{\{S\}}(f) \leq D(f) + 4k$.*

Proof. For every assignment $\vec{a} \in \mathbb{B}^{|S|}$, compute $g_{\vec{a}} = f(x|_{S \leftarrow \vec{a}})$, where $x|_{S \leftarrow \vec{a}}$ is the bit string obtained by replacing in x the bits at the positions S by the bit vector $\vec{a}$. We feed the results and the actual input bits from indices in S into the hazard-free k-bit $\mathtt{MUX}$ from Lemma 5.1 such that for stable values the correct output is determined. The correctness of the construction is now immediate from the fact that the $\mathtt{MUX}$ is hazard-free.

Concerning the size bound, for each $\vec{a} \in \mathbb{B}^{|S|}$, we have $L(g_{\vec{a}}) \leq L(f)$. Using the size bound for the $\mathtt{MUX}$ from Lemma 5.1, the construction thus has size smaller than $2^k(L(f) + 6)$. Similarly, we combine $D(g_{\vec{a}}) \leq D(f)$ with the depth of the $\mathtt{MUX}$ to obtain the bound $D_{\{S\}}(f) \leq D(f) + 4k$. □

Using this construction as the base case, we increase the number of sets (i.e., possible positions of the k unstable bits) our circuits can handle.

Theorem 5.3. *Let* $T = \binom{[n]}{k}$. *Then,*

$$L_T(f) \leq \left(\frac{ne}{k}\right)^{2k} (L(f) + 7) \text{ and } D_T(f) \leq D(f) + 2k\,(\lceil \log n \rceil + 2).$$

Proof. Put an ordering on T and let T_i be the ith element in T, $1 \leq i \leq |T|$. Denote by C_{ij}, $1 \leq i, j \leq |T|$, a circuit whose outputs coincide with $\bar{f}$ whenever all unstable bits are from $T_i \cup T_j$. Set $a_i := \mathsf{and}(C_{i1}, \ldots, C_{i|T|})$ (where a hazard-free and with fan-in $|T|$ is implemented by a binary tree of fan-in 2 ands of minimum depth). We claim that $o := \mathsf{or}(a_1, \ldots, a_{|T|})$ (again a hazard-free version implemented by a tree) coincides with $\bar{f}$ whenever there are at most k unstable bits.

To show the claim, assume that $x \in \mathbb{T}^n$ is stable except at indices from some $T_i \in \binom{[n]}{k}$. Assume first that $\bar{f}(x) = 1$. Then $a_i = \mathsf{and}(1, \ldots, 1) = 1$. This implies $o = 1$, because the $|T|$-bit or is hazard-free and one of its inputs is a 1. Next, suppose that $\bar{f}(x) = 0$. Then, for each $i' \leq |T|$, $C_{i'i}(x) = 0$. Hence, $a_{i'} = 0$, because the $|T|$-bit and is hazard-free and one of its inputs is a 0. It follows that $o = \mathsf{or}(0, \ldots, 0) = 0$. The case that $\bar{f}(x) = \mathfrak{u}$ is trivial; hence, the claim holds.

The above circuit contains the circuits C_{ij} and additionally $|T|^2 - 1$ many gates (a binary tree of ands and ors). By Lemma 5.2, each C_{ij} can be implemented with size $2^{2k}(L(f) + 6)$, as $|T_i \cup T_j| \leq 2k$. Moreover, using exactly all subsets of size $2k$, we use at most $\binom{n}{2k} \leq (\frac{en}{2k})^{2k}$ different such circuits. This results in a gate complexity of at most

$$\left(\frac{en}{k}\right)^{2k} (L(f) + 6) + \binom{n}{k}^2 - 1 < \left(\frac{en}{k}\right)^{2k} (L(f) + 7).$$

The depth of the circuit is $D(f) + 4k$ from the C_{ij} plus the depth of the trees, which is $\lceil \log(|T| - 1) \rceil \leq k \lceil \log n \rceil$. □

Corollary 1.10 simply rephrases the theorem without the terminology introduced in this section.

6 COMPLEXITY OF HAZARD DETECTION

In this section, we show that detecting hazards and detecting 1-bit hazards are both NP-complete problems, see Theorem 6.5 below. The arguments are a bit subtle, and thus we introduce several auxiliary hazard detection problems.

Definition 6.1. We say that a circuit C with n inputs has a *fixed hazard at position* $i \in [n]$ if C has a 1-bit hazard at a tuple $x \in \mathbb{T}^n$ with $x_i = \mathfrak{u}$.

We fix some reasonable binary encoding of circuits and define the following languages:

- $\mathsf{FixedHazard} = \{\langle C, i \rangle \mid C \text{ has a fixed hazard at position } i\}$
- $\mathsf{OneBitHazard} = \{C \mid C \text{ has a 1-bit hazard}\}$
- $\mathsf{Hazard} = \{C \mid C \text{ has a hazard}\}$

A circuit C is called *satisfiable* if there is a Boolean input for which C outputs 1. Otherwise, C is called *unsatisfiable.* We define a promise problem UnsatFixedHazard: Given an unsatisfiable circuit C and $i \in [n]$, accept if C has a fixed hazard at position i.

Lemma 6.2. UnsatFixedHazard *is* NP-*hard.*

Proof. We reduce from circuit satisfiability as follows: To decide if a circuit C on n inputs is satisfiable, construct a circuit $C' = C \wedge (x_{n+1} \wedge \neg x_{n+1})$ where x_{n+1} is a new variable. Note that C' is unsatisfiable by construction. We claim that C is satisfiable if and only if C' has a fixed hazard at position $n + 1$.

"⇒": Let a be an assignment that satisfies C. Then C' evaluates to $\mathfrak{u}$ on input $(a, \mathfrak{u})$ and hence has a fixed hazard at position $n + 1$.

"⇐": If C is unsatisfiable, then $C'(a, y) = 0$ for all $a \in \mathbb{B}^n$, $y \in \mathbb{T}$, and hence does not have a fixed hazard at position $n + 1$. □

Lemma 6.3. *The languages* FixedHazard*,* OneBitHazard*, and* Hazard *are* NP*-hard.*

Proof. Since UnsatFixedHazard is NP-hard, the more general problem FixedHazard is also NP-hard.

We show that deciding the languages OneBitHazard and Hazard is at least as hard as solving UnsatFixedHazard. Let $C(x_1, \dots, x_n)$ be an unsatisfiable circuit. Construct the circuit $C' = C(x_1, \dots, x_n) \oplus x_2 \oplus \cdots \oplus x_n$. We claim that C' has a hazard if and only if C has a fixed hazard at position x_1.

"⇒": Suppose C' has a hazard. Note that since C computes the constant 0 function, C' computes $x_2 \oplus \cdots \oplus x_n$. If any of the input variables $x_2, \dots, x_n$ has value $\mathfrak{u}$, then C' correctly outputs $\mathfrak{u}$. Thus, C' can have a hazard only on inputs $a \in \mathbb{T}^n$ that have exactly one $\mathfrak{u}$ occurring in the input position 1. In this case $C(a) = \mathfrak{u}$, because otherwise $C'(a)$ would be a Boolean value. Hence, C has a fixed hazard at position 1.

"⇐": If C has a fixed hazard at position 1, then by definition, C outputs $\mathfrak{u}$ when $x_1 = \mathfrak{u}$ while all other inputs are stable. In this case, C' also outputs $\mathfrak{u}$ on this input. This is a hazard, since the Boolean function computed by C' does not depend on x_1.

Thus, Hazard is NP-hard.

Note that in the first part of this proof, we actually proved that for the circuit C' all hazards are 1-bit hazards. So, the language OneBitHazard is also NP-hard. □

Lemma 6.4. *The languages* FixedHazard*,* OneBitHazard*, and* Hazard *are in* NP*.*

Proof. For FixedHazard and OneBitHazard, we can take the input on which the circuit has a hazard as a witness. The verifier then has to check that the circuit actually outputs $\mathfrak{u}$ on this input and that the outputs on the two stable inputs obtained by replacing $\mathfrak{u}$ by 0 and 1 match.

For Hazard, the verifier cannot check the definition directly, since the number of resolutions can be exponential. However, if a circuit C has a hazard, then there exists an input $x \in \mathbb{T}^n$ with $C(x) = \mathfrak{u}$ such that on the inputs $x^{(0)}$ and $x^{(1)}$ that are obtained from x by replacing the *leftmost* $\mathfrak{u}$ by 0 and 1, respectively, the circuit C outputs the same stable value b. This can be seen as follows. Let $H_C \subset \mathbb{T}^n$ be the set of all inputs on which C has a hazard. Any element x that is maximal in H_C with respect to $\preceq$ satisfies the requirement: since x is a hazard, $C(x) = \mathfrak{u}$ and the output of C on all resolutions of x is the same stable value b. Thus, the output of C on all resolutions of $x^{(0)}$ and of $x^{(1)}$ is b. Since x is maximal, both $x^{(0)}$ and $x^{(1)}$ do not lie in H_C, which implies $C(x^{(0)}) = C(x^{(1)}) = b$. Such x with $C(x) = \mathfrak{u}$ and $C(x^{(0)}) = C(x^{(1)}) = b$ can be used as a witness for Hazard. On the other hand, if there exists an input $x \in \mathbb{T}^n$ that satisfies the above condition, then the circuit C has a hazard at x. □

From Lemmas 6.3 and 6.4, we conclude:

Theorem 6.5. *The languages* FixedHazard*,* OneBitHazard*, and* Hazard *are* NP*-complete.*

7 FUTURE DIRECTIONS

Hazard-free complexity is an interesting subject that arises in practice when constructing physical circuits. Theorem 1.3 sends the strong message that hazard-free complexity is of interest even *without* its application in mind, because it is a natural generalization of monotone complexity.

Section 5 hints toward the fact that there is a rich intermediate landscape of k-hazard free circuits to be analyzed for different ranges of k. This can potentially be very illuminating for our understanding of the nature and limits of efficient computation in general.

Given the lower bound corollaries to Theorem 1.3 and the circuit construction in Corollary 1.10, one dangling open question is the fixed parameter tractability of k-hazard-free circuits: Does there exist a function φ such that for all sequences of Boolean functions $f_n : \mathbb{B}^n \to \mathbb{B}$ with Boolean complexity $L(f_n)$ there exist k-hazard-free circuits of size $\varphi(k) \cdot \text{poly}(n, L(f_n))$?

A further direction of interest is to understand the power of masking registers [16], both in terms of computational power and efficiency. It is neither known precisely which functions can be computed by clocked circuits within r rounds, nor is it clear whether a factor $\Omega(k)$ overhead for eliminating hazards with masking registers is necessary.

APPENDIX

A CIRCUITS WITH DIFFERENT BASIC GATES

In the main part of this article, we used circuits with and-, or-, and not-gates, because this is one of the standard models in circuit complexity. But, we have also already seen that the circuit transformations we use for proving lower bounds rely only on the general construction of the derivative and can be performed on circuits with arbitrary gates, not just and-, or-, and not-gates. In this Appendix, we show that any other functionally complete set (in the sense of, e.g., [14]) can be used to give an equivalent theory of hazard-free complexity and natural functions, see the upcoming Corollary A.3. *A priori* it is not obvious that every function can be implemented by a hazard-free circuit over some set of gates, even if the set of gates is functionally complete in the Boolean sense. We prove that everything works properly if we allow constant input gates. This subtlety is unavoidable, since any nontrivial natural function outputs $\mathfrak{u}$ if all inputs are $\mathfrak{u}$, so any circuit without constant gates also has this property. Therefore, the constant function is not computable by hazard-free circuits without the use of constant gates.

Reference [6, Theorem 2] shows that every natural function can be implemented over the set of functions $\Phi = \{\texttt{and}, \texttt{or}, \texttt{not}, 1\}$. Using the fact that a hazard-free implementation of or can be achieved via the standard De Morgan implementation $x \texttt{ or } y = \texttt{not}((\texttt{not } x) \texttt{ and } (\texttt{not } y))$, it follows that

$$\text{every natural function can be implemented over the set of functions } \{\texttt{and}, \texttt{not}, 1\}. \tag{A.1}$$

A Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ is called *linear* if there exist $a_0, \ldots, a_n \in \mathbb{B}$ with $f(x_1, \ldots, x_n) = a_0 \oplus a_1 x_1 \oplus \cdots \oplus a_n x_n$. Otherwise, f is called *nonlinear*. The composition of linear functions is linear, but not all Boolean functions are linear. Thus, every functionally complete set must contain a nonlinear function.

Variants of the following lemma are often used as a part of proof of Post's theorem characterizing functionally complete systems.

Lemma A.1. *Let $f : \mathbb{B}^n \to \mathbb{B}$ be a nonlinear Boolean function. Then, $n \geq 2$. Moreover, by substituting constants for some input variables of f, we can obtain a function of 2 variables of the form $(x_1 \oplus c_1)(x_2 \oplus c_2) \oplus c_0$.*

Proof. Using the fact that over the field $\mathbb{F}_2$ with two elements, we have $x_i \texttt{ and } x_j = x_i \cdot x_j$ and $\texttt{not } x_i = x_i \oplus 1$, we can represent f as a polynomial over $\mathbb{F}_2$. Using that $(x_i)^k = x_i$ for $k \geq 1$, we can represent f in its algebraic normal form,

$$f(x_1, \ldots, x_n) = \bigoplus_{I \subset [n]} a_I \prod_{i \in I} x_i,$$

where each $a_I \in \mathbb{B}$. We call I a *monomial* and call $|I|$ its *degree*. Since f is nonlinear, there is at least one monomial of degree at least 2 with nonzero coefficient a_I. Thus, we proved $n \geq 2$. Among monomials of degree at least 2, choose one monomial of minimal degree and set all the variables not contained in this monomial to 0. Without loss of generality, the chosen monomial is $x_1 \cdots x_t$, $t \geq 2$. The resulting function has the form

$$x_1 \ldots x_t \oplus a_1 x_1 \oplus \cdots \oplus a_t x_t \oplus a_0.$$

Setting all variables except x_1 and x_2 to 1, we obtain $x_1 x_2 \oplus a_1 x_1 \oplus a_2 x_2 \oplus a'_0$, or $(x_1 \oplus c_1)(x_2 \oplus c_2) \oplus c_0$ where $c_1 = a_2$, $c_2 = a_1$ and $c_0 = a'_0 \oplus a_1 a_2$. □

Theorem A.2. *Let Φ be a set of natural functions such that their restrictions to $\mathbb{B}$ form a functionally complete set. Suppose Φ contains an extension of a nonlinear Boolean function that is free of* 1*-bit hazards. Then every natural function can be computed by a circuit over Φ using the constant* 1.

Proof. In the light of (A.1), it is enough to show that hazard-free circuits for `not` and `and` can be implemented over Φ. The statement is trivial for the negation: since `not` has only one natural extension, any circuit that computes it is automatically hazard-free. Using `not`, we can obtain the constant 0 from the constant 1.

By Lemma A.1, we obtain from the 1-hazard-free nonlinear function contained in Φ a function of the form $(x_1 \oplus c_1)(x_2 \oplus c_2) \oplus c_0$ by substituting constants 0 and 1 into this nonlinear function. Constant substitution does not introduce hazards. Since `not` $x = x \oplus 1$, we can transform the circuit C computing $(x_1 \oplus c_1)(x_2 \oplus c_2) \oplus c_0$ to a circuit C' computing $x_1 x_2$ by placing `not` on input x_i if $c_i = 1$ and on the output if $c_0 = 1$. In other words, $C'(x_1, x_2) = C(x_1 \oplus c_1, x_2 \oplus c_2) \oplus c_0$.

Let us check that C' is hazard-free. The circuit C' is computing the conjunction and thus can have hazards only on two inputs: $(0, \mathfrak{u})$ and $(\mathfrak{u}, 0)$. If $C'(0, \mathfrak{u}) = \mathfrak{u}$, then $C(c_1, \mathfrak{u}) = \mathfrak{u}$. This is a 1-bit hazard, since $(c_1 \oplus c_1)(x \oplus c_2) \oplus c_0 = c_0$ for all $x \in \mathbb{B}$. The other case is analogous. □

Corollary A.3. *Given a functionally complete set of Boolean functions, let Φ be the set of their hazard-free extensions. Every natural function can be computed by a circuit over Φ using the constant 1.*

Proof. Since a functionally complete set cannot only consist of linear functions, at least one function must be nonlinear. A hazard-free function in particular does not have a 1-hazard. Thus, Theorem A.2 applies. □

ACKNOWLEDGMENTS

We thank Arseniy Alekseyev, Karl Bringmann, Ulan Degenbaev, Stefan Friedrichs, and Matthias Függer for helpful discussions. Moreover, we thank Attila Kinali for his help with Japanese references.

REFERENCES

[1] Miklos Ajtai and Yuri Gurevich. 1987. Monotone versus positive. *J. ACM* 34, 4 (Oct. 1987), 1004–1015. DOI: https://doi.org/10.1145/31846.31852

[2] Noga Alon and Ravi B. Boppana. 1987. The monotone circuit complexity of Boolean functions. *Combinatorica* 7, 1 (1987), 1–22. DOI: https://doi.org/10.1007/BF02579196

[3] Walter Baur and Volker Strassen. 1983. The complexity of partial derivatives. *Theor. Comput. Sci.* 22 (1983), 317–330. DOI: https://doi.org/10.1016/0304-3975(83)90110-X

[4] Norbert Blum. 1985. An $\Omega(n^{4/3})$ lower bound on the monotone network complexity of the nth degree convolution. *Theoretical Computer Science* 36, Supplement C (1985), 59–69. DOI: https://doi.org/10.1016/0304-3975(85)90030-1

[5] J. Brzozowski, Z. Esik, and Y. Iland. 2001. Algebras for hazard detection. In *Proceedings of the 31st International Symposium on Multiple-Valued Logic.*

[6] J. A. Brzozowski. 1999. Some applications of ternary algebras. *Publicationes Mathematicae (Debrecen)* 54, Supplement (1999), 583–599.

[7] Janusz A. Brzozowski and Carl-Johan H. Seger. 1995. *Asynchronous Circuits*. Springer, New York.

[8] H. Martin Bücker and George F. Corliss. 2006. A bibliography of automatic differentiation. In *Automatic Differentiation: Applications, Theory, and Implementations*, Martin Bücker, George Corliss, Uwe Naumann, Paul Hovland, and Boyana Norris (Eds.). Springer-Verlag, Berlin, 321–322.

[9] Samuel H. Caldwell. 1958. *Switching Circuits and Logical Design*. John Wiley & Sons.

[10] Ashok K. Chandra and George Markowsky. 1978. On the number of prime implicants. *Discrete Math.* 24, 1 (1978), 7–11. DOI: https://doi.org/10.1016/0012-365X(78)90168-1

[11] Marc Davio, Jean-Pierre Deschamps, and André Thaysé. 1978. *Discrete and Switching Functions*. McGraw-Hill.

[12] A. Martín del Rey, G. Rodríguez Sánchez, and A. de la Villa Cuenca. 2012. On the Boolean partial derivatives and their composition. *Appl. Math. Lett.* 25, 4 (2012), 739–744. DOI: https://doi.org/10.1016/j.aml.2011.10.013

[13] E. B. Eichelberger. 1965. Hazard detection in combinational and sequential switching circuits. *IBM J. Res. Dev.* 9, 2 (March 1965), 90–99. DOI: https://doi.org/10.1147/rd.92.0090

[14] Herbert B. Enderton. 2001. *A Mathematical Introduction to Logic* (2nd ed.). Academic Press.

[15] Stephan Friedrichs. 2017. *Metastability-containing circuits, parallel distance problems, and terrain guarding*. Ph.D. Dissertation. Universität des Saarlandes, Postfach 151141, 66041 Saarbrücken. http://scidok.sulb.uni-saarland.de/volltexte/2017/6966

[16] Stephan Friedrichs, Matthias Függer, and Christoph Lenzen. 2018. Metastability-containing circuits. *IEEE Trans. Comput.* (2018). Retrieved from https://arxiv.org/abs/1606.06570.

[17] M. Goto. 1948. Application of three-valued logic to construct the theory of relay networks (in Japanese) 三値論理学の継電器回路網理論への應用. *Proceedings of the Joint Meeting IEE, IECE, and I. of Illum. E. of Japan,* 電気三学 会東京支部連合大会講演要旨. 昭和 *22 · 23*年 (1948), 31–32.

[18] M. Goto. 1949. Application of logical mathematics to the theory of relay networks (in Japanese). *J. Inst. Elec. Eng. Japan* 64, 726 (1949), 125–130.

[19] Michelangelo Grigni and Michael Sipser. 1992. Monotone complexity. In *Proceedings of the London Mathematical Society Symposium on Boolean Function Complexity*. Cambridge University Press, New York, NY, 57–75. http://dl.acm.org/citation.cfm?id=167687.167706

[20] W. Hu, J. Oberg, A. Irturk, M. Tiwari, T. Sherwood, D. Mu, and R. Kastner. 2012. On the complexity of generating gate level information flow tracking logic. *IEEE Trans. Info. Forensics Secur.* 7, 3 (June 2012), 1067–1080. DOI: https://doi.org/10.1109/TIFS.2012.2189105

[21] David A. Huffman. 1957. The design and use of hazard-free switching networks. *J. ACM* 4, 1 (Jan. 1957), 47–62. DOI: https://doi.org/10.1145/320856.320866

[22] Christian Ikenmeyer, Balagopal Komarath, Christoph Lenzen, Vladimir Lysikov, Andrey Mokhov, and Karteek Sreenivasaiah. 2018. On the complexity of hazard-free circuits. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC'18)*. ACM, New York, NY, 878–889. DOI: https://doi.org/10.1145/3188745.3188912

[23] Stasys Jukna. 2012. *Boolean Function Complexity—Advances and Frontiers*. Algorithms and combinatorics, Vol. 27. Springer. DOI: https://doi.org/10.1007/978-3-642-24508-4

[24] Stephen Cole Kleene. 1938. On notation for ordinal numbers. *J. Symbol. Logic* 3, 4 (1938), 150–155. http://www.jstor.org/stable/2267778

[25] Stephen Cole Kleene. 1952. *Introduction to Metamathematics*. North Holland.

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

[27] François Le Gall. 2014. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC'14)*. ACM, New York, NY, 296–303. DOI: https://doi.org/10.1145/2608628.2608664

[28] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. 2011. Lower bounds based on the exponential time hypothesis. *Bull. EATCS* 105 (2011), 41–71.

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

[30] Leonard R. Marino. 1981. General theory of metastable operation. *IEEE Trans. Comput.* 30, 2 (1981), 107–115.

[31] K. Mehlhorn and Z. Galil. 1976. Monotone switching circuits and boolean matrix product. *Computing* 16, 1 (01 Mar. 1976), 99–111. DOI: https://doi.org/10.1007/BF02241983

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

[33] M. Mukaidono. 1983. Advanced results on application of fuzzy switching functions to hazard detection. In *Advances in Fuzzy Sets, Possibility Theory and Applications*, P. P. Wong (Ed.). Plenum Publishing, 335–349.

[34] M. Mukaidono. 1983. Regular ternary logic functions—Ternary logic functions suitable for treating ambiguity. In *Proceedings of the 13th International Symposium on Multiple-Valued Logic*. IEEE Computer Society Press, 286–291.

[35] S. M. Nowick and D. L. Dill. 1992. Exact two-level minimization of hazard-free logic with multiple-input changes. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 626–630. DOI: https://doi.org/10.1109/ICCAD.1992.279301

[36] Michael S. Paterson. 1975. Complexity of monotone networks for Boolean matrix product. *Theoret. Comput. Sci.* 1, 1 (1975), 13–20. DOI: https://doi.org/10.1016/0304-3975(75)90009-2

[37] Pedro Ponce-Cruz and Fernando D. Ramírez-Figueroa. 1979. *Intelligent Control Systems with LabVIEW*. Springer-Verlag, London.

[38] Vaughan R. Pratt. 1974. The power of negative thinking in multiplying boolean matrices. In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing (STOC'74).* ACM, New York, NY, 80–83. DOI: https://doi.org/10.1145/800119.803887

[39] Ran Raz and Avi Wigderson. 1992. Monotone circuits for matching require linear depth. *J. ACM* 39, 3 (1992), 736–744. DOI: https://doi.org/10.1145/146637.146684

[40] Alexander A. Razborov. 1985. Lower bounds on monotone complexity of the logical permanent. *Math. Notes* 37, 6 (1985), 485–493. DOI: https://doi.org/10.1007/BF01157687

[41] Raul Rojas. 1996. *Neural Networks—A Systematic Introduction.* Springer-Verlag, Berlin.

[42] Volker Strassen. 1969. Gaussian elimination is not optimal. *Numer. Math.* 13 (1969), 354–356.

[43] G. Tarawneh and A. Yakovlev. 2012. An RTL method for hiding clock domain crossing latency. In *Proceedings of the 19th IEEE International Conference on Electronics, Circuits, and Systems (ICECS'12).* 540–543. DOI: https://doi.org/10.1109/ICECS.2012.6463557

[44] G. Tarawneh, A. Yakovlev, and T. Mak. 2014. Eliminating synchronization latency using sequenced latching. *IEEE Trans. Very Large Scale Integr. Syst.* 22, 2 (Feb. 2014), 408–419. DOI: https://doi.org/10.1109/TVLSI.2013.2243177

[45] É. Tardos. 1988. The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica* 8, 1 (Mar. 1988), 141–142. DOI: https://doi.org/10.1007/BF02122563

[46] Mohit Tiwari, Hassan M. G. Wassel, Bita Mazloom, Shashidhar Mysore, Frederic T. Chong, and Timothy Sherwood. 2009. Complete information flow tracking from the gates up. *SIGARCH Comput. Archit. News* 37, 1 (Mar. 2009), 109–120. DOI: https://doi.org/10.1145/2528521.1508258

[47] S. H. Unger. 1995. Hazards, critical races, and metastability. *IEEE Trans. Comput.* 44, 6 (June 1995), 754–768. DOI: https://doi.org/10.1109/12.391185

[48] Ingo Wegener. 1982. Boolean functions whose monotone complexity is of size $n^2 / \log n$. *Lect. Notes Comput. Sci.* 21 (Nov. 1982), 213–224.

[49] A. C. Yao. 1989. Circuits and local computation. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC'89).* ACM, New York, NY, 186–196. DOI: https://doi.org/10.1145/73007.73025

[50] Michael Yoeli and Shlomo Rinon. 1964. Application of ternary algebra to the study of static hazards. *J. ACM* 11, 1 (Jan 1964), 84–97. DOI: https://doi.org/10.1145/321203.321214

[51] Y. Zisapel, M. Krieger, and J. Kella. 1979. Detection of hazards in combinational switching circuits. *IEEE Trans. Comput.* C-28, 1 (Jan. 1979), 52–56. DOI: https://doi.org/10.1109/TC.1979.1675222

Received July 2018; revised January 2019; accepted March 2019