Sensitivity and Query Complexity under Uncertainty

Deepu Benson
Institute of Science and Technology, Chinmaya Vishwa Vidyapeeth, Ernakulam, India

Balagopal Komarath
Department of Computer Science and Engineering, IIT Gandhinagar, Gujarat, India

Nikhil Mande
Department of Computer Science, University of Liverpool, UK

Sai Soumya Nalli
Microsoft Research, Bangalore, India

Jayalal Sarma
Department of Computer Science and Engineering, IIT Madras, Chennai, India

Karteek Sreenivasaiah
Department of Computer Science, University of Liverpool, UK

Abstract

In this paper, we study the query complexity of Boolean functions in the presence of uncertainty, motivated by parallel computation with an unlimited number of processors where inputs are allowed to be unknown. We allow each query to produce three results: zero, one, or unknown. The output could also be: zero, one, or unknown, with the constraint that we should output "unknown" only when we cannot determine the answer from the revealed input bits. Such an extension of a Boolean function is called its *hazard-free* extension.

- We prove an analogue of Huang's celebrated sensitivity theorem [Annals of Mathematics, 2019] in our model of query complexity with uncertainty.
- We show that the deterministic query complexity of the hazard-free extension of a Boolean function is at most quadratic in its randomized query complexity and quartic in its quantum query complexity, improving upon the best-known bounds in the Boolean world.
- We exhibit an exponential gap between the smallest depth (size) of decision trees computing a Boolean function, and those computing its hazard-free extension.
- We present general methods to convert decision trees for Boolean functions to those for their hazard-free counterparts, and show optimality of this construction. We also parameterize this result by the maximum number of unknown values in the input.
- We show lower bounds on size complexity of decision trees for hazard-free extensions of Boolean functions in terms of the number of prime implicants and prime implicates of the underlying Boolean function.

2012 ACM Subject Classification Theory of computation → Oracles and decision trees

Keywords and phrases CREW-PRAM, query complexity, decision trees, sensitivity, hazard-free extensions

Related Version This paper (see [4] for the full version) merges and extends results from [18] and [5].
Full Version: `https://arxiv.org/abs/2507.00148`

Funding *Deepu Benson*: Funded by SERB SRG Project SRG/2021/001339

1 Introduction

A single-tape Turing Machine needs at least n steps in the worst case to compute any n-bit function that depends on all of its inputs. One way to achieve faster computation is to

use multiple processors in parallel. Parallel computation is modeled using *Parallel Random Access Machines* (PRAM), originally defined by Fortune and Wyllie [10], in which there are multiple processors and multiple memory cells shared among all processors. In a single time step, each processor can read a fixed number of memory cells, execute one instruction, and write to a fixed number of memory cells. In the real world, access to shared memory has to be mediated using some mechanism to avoid read-write conflicts. A popular mechanism to achieve this is called *Concurrent-Read Exclusive-Write* (CREW) in which concurrent reads of the same memory cell are allowed, but each cell can only be written to by at most one processor in a single time step. An algorithm that violates this restriction is invalid.

A fundamental problem in such a model of computation is to determine the number of processors and the amount of time required for computing Boolean functions. A Boolean function $f : \{0,1\}^n \mapsto \{0,1\}$ is pre-determined[1] and the input bits are presented in shared memory locations. The processors have to write the output bit $f(x)$ to a designated shared memory location. For example, consider the Boolean disjunction (logical OR) of all input bits which outputs 1 if and only if there is at least one 1 among the inputs. There is a simple divide-and-conquer algorithm to compute the OR of n input bits in $O(\log n)$ time using n processors that exploits the fact that OR is associative and distributive. This is essentially a CREW-PRAM algorithm to compute the OR of n bits in $O(\log(n))$-time. Note that each of the processors in the above algorithm computes a trivial function at each step namely the OR of two bits. Can we do better if we are allowed to use more complex computations at each step?

Cook and Dwork [7], and Cook, Dwork, and Reischuk [8] answer this question by showing that any CREW-PRAM algorithm that computes the logical OR of n-bits needs $\Omega(\log(n))$-time, irrespective of the functions computed by the processors at each step. The reason their lower bound is independent of the functions allowed at each processor is because their lower bound really applies to the number *accesses* made to the shared memory. If we only care about analyzing the number of memory accesses by an algorithm running on an all-powerful processor, a neat way to think of the computation at each processor is to model it as a two-player interactive game: a *querier* who is all-powerful and an *oracle*. They want to compute a Boolean function $f : \{0,1\}^n \to \{0,1\}$ known beforehand, on an input $x \in \{0,1\}^n$. However, the input x is only known to the oracle. The only interaction allowed is where the querier asks a query $i \in [n]$, and the oracle can reply with x_i. The *query complexity* of f is defined as the the maximum number of queries required to find the answer, where the maximum is taken over all x. The querier's aim is to minimize the number of queries required to determine the value of the function in the worst-case. This is exactly the computation model studied in an exciting area of computer science simply called 'query complexity'.

The technique used in [7] and [8] involves defining a measure that is equivalent to what is now commonly known as *sensitivity* of a Boolean function, denoted $\mathsf{s}(f)$. More precisely, they show that the time needed by CREW-PRAM algorithms, irrespective of the instruction set, to compute a Boolean function is asymptotically lower bounded by the logarithm of the sensitivity of the function.

The question whether a function can be computed in time at most the logarithm of the function's sensitivity, thereby characterizing the CREW-PRAM time complexity in terms of sensitivity, remained open. Nisan [20] introduced a generalization of sensitivity called *block-sensitivity*, denoted $\mathsf{bs}(f)$, which is lower bounded by sensitivity, and proved that CREW-PRAM time complexity is asymptotically the same as the logarithm of the block

[1] The number of input bits is fixed, making this model non-uniform.

$\wedge$	0	u	1
0	0	0	0
u	0	u	u
1	0	u	1

$\vee$	0	u	1
0	0	u	1
u	u	u	1
1	1	1	1

$\neg$	0	u	1
	1	u	0

Table 1 Kleene's three valued logic 'K3'

sensitivity of a function. Nisan further related *decision tree depth complexity*, denoted $\mathsf{D}(f)$ to block-sensitivity providing another characterization for CREW-PRAM time complexity as the logarithm of decision tree depth. However, the block-sensitivity of a function can be potentially much higher than its sensitivity, as shown by Rubinstein [22] with an explicit function that witnesses a quadratic gap between these two measures. After research spanning almost three decades which saw extensive study of relationships between the above parameters and various other parameters (see, for example, [1] and the references therein), such as certificate complexity (denoted $\mathsf{cc}(f)$) and degree (denoted $\mathsf{deg}(f)$), Huang [13], in a breakthrough result, proved that sensitivity and block-sensitivity are polynomially related. This is the celebrated sensitivity theorem:

▶ **Theorem 1** (Sensitivity theorem for Boolean functions [13]). *For all Boolean functions* $f : \{0,1\}^n \to \{0,1\}$, $\mathsf{bs}(f) = O(\mathsf{s}(f)^4)$.

By earlier results [20, 21, 3], Huang's result implies that the measures of sensitivity, block sensitivity, certificate complexity, degree, approximate degree, deterministic/randomized/quantum query complexity are all polynomially equivalent for Boolean functions. This strong connection makes the study of query complexity essentially equivalent to studying the time complexity of CREW-PRAMs. Thus henceforth, we shall predominantly use terminology from query complexity, and also express our results in the context of query complexity.

In this work, we initiate a systematic study to understand the effect of allowing *uncertainty* among the inputs in the setting of CREW-PRAM. Allowing uncertainty in this setting is easier to understand in the equivalent query model: When an input bit is queried by the querier, the oracle can reply with "uncertain". If it is possible for the function value to be determined in the presence of such uncertainties, then we would like the querier to output such a value. In what follows, we make the setting more formal.

To model uncertainty we use a classic three-valued logic, namely Kleene's strong logic of indeterminacy [17], usually denoted *K3*. The logic K3 has three truth values 0, 1, and u. The behaviour of the third value u with respect to the basic Boolean primitives — conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$) — are given in Table 1.

The logic K3 has been used in several other contexts where there is a need to represent and work with *unknowns* and hence has found several important wide-ranging applications in computer science. For instance, in relational database theory, SQL implements K3 and uses u to represent a NULL value [19]. Perhaps the oldest use of K3 is in modeling *hazards* that occur in real-world combinational circuits. Recently there have been a series of results studying constructions and complexity of hazard-free circuits [14, 15, 16]. Here u was used to represent *metastability*, or an *unstable* voltage, that can resolve to 0 or 1 at a later time.

One way to interpret the basic binary operations $\vee$, $\wedge$, and $\neg$ in K3 is as follows: for a bit $b \in \{0,1\}$, if the value of the function is guaranteed to be b regardless of all $\{0,1\}$-settings to the u-variables, then the output is b. Otherwise, the output is u. This interpretation generalizes in a nice way to n-variate Boolean functions. In literature, this extension is typically called the *hazard-free* extension of f (see, for instance, [14]), and is an important

concept studied in circuits and switching network theory since the 1950s. The interested reader can refer to [14], and the references therein, for history and applications of this particular way of extending f to K3. We define this extension formally below.

For a string $x \in \{0, \mathsf{u}, 1\}^n$, define the *resolutions* of x as follows:

$$\mathsf{Res}(x) := \{y \in \{0,1\}^n : y_i = x_i \ \forall i \in [n] \text{ with } x_i \in \{0,1\}\}.$$

That is, $\mathsf{Res}(x)$ denotes the set of all strings in $\{0,1\}^n$ that are consistent with the $\{0,1\}$-valued bits of x. The hazard-free extension of a Boolean function is defined as follows:

▶ **Definition 2** (Hazard-free Extensions). *For a Boolean function $f : \{0,1\}^n \to \{0,1\}$, we define its hazard-free extension $\widetilde{f} : \{0, \mathsf{u}, 1\}^n \to \{0, \mathsf{u}, 1\}$ as follows. For an input $y \in \{0, \mathsf{u}, 1\}^n$,*

$$\widetilde{f}(y) := \begin{cases} b & \text{if } f(y) = b \text{ for all } y \in \mathsf{Res}(x) \\ u & \text{otherwise} \end{cases}$$

To understand the motivation behind this definition, consider the *instability partial order* defined on $\{0, \mathsf{u}, 1\}$ by the relations $\mathsf{u} \leq 0$ and $\mathsf{u} \leq 1$. The elements 0 and 1 are incomparable. This partial order captures the intuition that u is less certain than 0 and 1. This partial order can be extended naturally to elements of $\{0, \mathsf{u}, 1\}^n$ as $x \leq y$ iff $x_i \leq y_i$ for all $1 \leq i \leq n$. A function $f' : \{0, \mathsf{u}, 1\}^n \to \{0, \mathsf{u}, 1\}$ is *natural* if for all $x, y \in \{0, \mathsf{u}, 1\}^n$ such that $x \leq y$, we have $f'(x) \leq f'(y)$ and $f'(z) \in \{0,1\}$ when $z \in \{0,1\}^n$. Intuitively, this property says that the function cannot produce a less certain output on a more certain input and if there is no uncertainty in the input, there should be no uncertainty in the output. A natural function f' extends a Boolean function f if $f'(x) = f(x)$ for all $x \in \{0,1\}^n$. There could be many natural functions that extend a Boolean function. Consider two natural functions f' and f'' that extend f. We say $f' \leq f''$ if $f'(x) \leq f''(x)$ for all $x \in \{0, \mathsf{u}, 1\}^n$. This says that the output of f'' is at least as certain as the output of f'. An alternative definition for the hazard-free extension of a function f is as follows: it is the unique function $\widetilde{f}$ such that $f' \leq \widetilde{f}$ for all natural functions f' that extends f. That is, the hazard-free extension of a Boolean function is the best we could hope to compute in the presence of uncertainties in the inputs to f.

We note here that even though we use the term "hazard-free", there is a fundamental difference between our model of computation and the ones studied in results such as [14, 15, 16]. For hazard-free circuits, the value u represents an unstable voltage, and the gates in a circuit are fundamentally unable to detect it. That is, there is no circuit that can output 1 when input is u and 0 otherwise. However, in our setting, the value u is simply another symbol just like 0 or 1. So we can indeed detect/read a u value. The restriction that we have to compute the hazard-free extension is a semantic one in this paper, whereas Boolean circuits can only compute natural functions. In other words, we are interested in query complexity of hazard-free extensions of Boolean functions per se, and we have no notion of metastability in our computation model.

There is a rich body of work on the query complexity of Boolean functions that has established best-possible polynomial relationships among various models such as deterministic, randomized, and quantum models of query complexity, and their interplay with analytical measures such as block sensitivity and certificate complexity (see, for example, [1] and the references therein). We study these relationships in the presence of uncertainty, in particular for hazard-free extensions of Boolean functions. A main goal is to characterize query complexity (equivalently CREW-PRAM time complexity) of hazard-free extensions of Boolean functions using these parameters.

1.1 Our Results

In this subsection, we discuss the results presented in this paper. The organization of the paper follows the structure of this subsection.

1.1.1 Sensitivity Theorem in the Presence of Uncertainty

We prove the sensitivity theorem for Boolean functions in the presence of uncertainties. We define analogues of query complexity, sensitivity, block sensitivity, and certificate complexity called u-query complexity (denoted $\mathsf{D_u}(f)$), u-sensitivity (denoted $\mathsf{s_u}(f)$), u-block sensitivity (denoted $\mathsf{bs_u}(f)$), and u-certificate complexity (denoted $\mathsf{cc_u}(f)$), respectively. We show that analogues of Cook, Dwork, and Reischuk's [8] lower bound and Nisan's [20] upper bound hold in the presence of uncertainties by adapting their proofs suitably (See the full version [4] for details). Therefore, we can now focus on proving that these parameters are polynomially equivalent in the presence of uncertainties. Huang's proof of the sensitivity theorem for Boolean functions crucially uses the parameter called the degree of a Boolean function. It is unclear how to define an analogue of degree in our setting. However, it turns out that a more classical parameter, the maximum of prime implicant size and prime implicate size, suffices.

Our proof of the sensitivity theorem in the presence of uncertainty is much simpler and more straightforward than the proof of the sensitivity theorem in the classical setting. It raises the question of whether we can find a simpler proof of the classical sensitivity theorem by generalizing our proof to handle an arbitrary upper bound on the number of uncertain values in the input. Note that the classical setting assumes that this number is 0 and we prove the sensitivity theorem by assuming that this number is n, the number of inputs.

Recall that a *literal* is an input variable or its negation. An *implicant* (*implicate*) of a Boolean function f is a subset S of all literals such that f is 1 (is 0) on any input that has all literals in S set to 1 (set to 0 respectively). A *prime implicant* (*prime implicate*) is an implicant (implicate) such that no proper subset of it is an implicant (implicate respectively), i.e., the implicant (implicate) is minimal (w.r.t. set inclusion). The *size* of a prime implicant or prime implicate is the size of the set. Prime implicants and prime implicates of a Boolean function are widely studied in electronic circuit design and Boolean function analysis.

▶ **Theorem 3** (Sensitivity theorem for hazard-free extensions of Boolean functions)**.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function and let k_1 and k_2 be the sizes of a largest prime implicant and prime implicate of f. Then, the parameters $\mathsf{s_u}(f)$, $\mathsf{bs_u}(f)$, $\mathsf{D_u}(f)$, $\mathsf{cc_u}(f)$, and $\max\{k_1, k_2\}$ are linearly equivalent.*

We note here that while Huang [13] showed $\mathsf{bs}(f) = O(s(f)^4)$ for all Boolean f, our result shows that, in the presence of uncertainty block sensitivity and sensitivity are in fact linearly related to each other.

1.1.2 Relationships

Clearly, the query complexity of the hazard-free extension of a Boolean function f cannot be smaller than that of f itself. Can the query complexity of the hazard-free extension of a Boolean function be much more than the query complexity of the function itself? For monotone functions, we show that the answer is no. In fact, this also holds for randomized query complexity (denoted by $\mathsf{R}(f)$ and $\mathsf{R_u}(f)$) and quantum query complexity (denoted by $\mathsf{Q}(f)$ and $\mathsf{Q_u}(f)$).

▶ **Lemma 4.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a monotone Boolean function. Then we have*

$$\mathsf{D_u}(f) = \Theta(\mathsf{D}(f)) \text{ and } \mathsf{R_u}(f) = \Theta(\mathsf{R}(f)) \text{ and } \mathsf{Q_u}(f) = \Theta(\mathsf{Q}(f)).$$

A natural question to ask is whether the model we are considering is non-degenerate. In other words, do all hazard-free extensions of Boolean functions have large query complexity? Using Lemma 4, we can show that there are functions that are easy to compute even in the presence of uncertainties. The following monotone variant of the MUX function (defined below) by Wegener [24] is sufficient.

▶ **Definition 5** ([24])**.** *For an even integer $n > 0$, define mMUX_n, as follows: The function mMUX_n is defined on $n+\binom{n}{n/2}$ (which is $\Theta(2^n/\sqrt{n})$) variables, where the latter $\binom{n}{n/2}$ variables are indexed by all n-bit strings of Hamming weight exactly $n/2$. For $(x, y) \in \{0,1\}^{n+\binom{n}{n/2}}$, define*

$$\mathsf{mMUX}_n(x,y) = \begin{cases} 0 & |x| < n/2 \\ 1 & |x| > n/2 \\ y_x & \text{otherwise.} \end{cases}$$

It is easy to see that this function is monotone and has a query complexity of $n + 1$. We also exhibit a non-monotone, non-degenerate n-variate function such that its hazard-free extension has $O(\log n)$ query complexity (see full version [4] for the details).

For general functions, we show that uncertainty can blow-up query complexity exponentially. The Boolean function $\mathsf{MUX}_n : \{0,1\}^{n+2^n} \to \{0,1\}$ defined by

$$\mathsf{MUX}_n(s_0, s_1, \ldots, s_{n-1}, (x_{(b_0,\ldots,b_{n-1})})_{b_i \in \{0,1\}}) := x_{(s_0,\ldots,s_{n-1})}$$

is a function on $n + 2^n$ inputs that depends on all its inputs and has query complexity of $n + 1$. The inputs s_i are called the *selector* bits and the inputs x_j are called *data* bits. It is easy to show that any function that depends on all its N input bits must have at least logarithmic (in N) query complexity. Therefore, MUX_n is one of the easiest functions to compute in the query complexity model. We prove that its hazard-free extension is one of the hardest functions in the query complexity model.

▶ **Theorem 6.**

$$\mathsf{D}_u(\mathsf{MUX}_n) = 2^n + n \text{ and } \mathsf{R_u}(\mathsf{MUX}_n) = \Theta(2^n) \text{ and } \mathsf{Q_u}(\mathsf{MUX}_n) = \Theta(2^{n/2})$$

We also show the following relationships between deterministic, randomized, and quantum query complexities of hazard-free extensions of Boolean functions.

▶ **Theorem 7.** *For $f : \{0,1\}^n \to \{0,1\}$, we have*

$$\mathsf{D_u}(f) = O(\mathsf{R_u}(f)^2) \text{ and } \mathsf{D_u}(f) = O(\mathsf{Q_u}(f)^4).$$

We remark here that the deterministic-randomized relationship above is *better* than the best-known cubic relationship in the Boolean world, while the quartic deterministic-quantum separation above matches the best-known separation in the Boolean world (see [1]). The key reason we are able to obtain better relationships is because we show that sensitivity, block sensitivity, and certificate complexity are all linearly related in the presence of uncertainty (Theorem 3). This is not the case in the classical Boolean setting. Regarding best possible separations: while a linear relationship between deterministic and randomized query complexities

in the presence of uncertainty remains open, a quadratic deterministic-quantum separation follows from Theorem 6 (or from the OR function, which has maximal deterministic query complexity, but Grover's search algorithm [12] offers a quadratic quantum speedup).

It is natural to model query algorithms using *decision trees*. A decision tree represents the strategy of the querier using a tree structure. The internal nodes of the tree are labeled by input variables and has two outgoing edges labeled 0 and 1. Computation starts at the root and queries the variable labeling the current node. Then, the outgoing edge labeled by the answer is taken to reach the next node. The leaves of the tree are labeled with 0 or 1 and represent the answer determined by the querier. A decision tree is said to compute a Boolean function f if the querier can correctly answer the value of f for every possible input by following the decision tree from root to a leaf. The *depth* of the decision tree represents the worst-case time. The depth of a smallest depth decision tree that computes f is called the *decision tree depth complexity* of f, and it is the same as *deterministic query complexity* of f.

1.1.3 Decision Tree Size

Decision trees have played an important role in machine learning. The size of a decision tree is an important measure as large decision trees often suffer from over-fitting. It has been long-known that functions that admit small-size decision trees are efficiently PAC learnable [9]. Moreover, the class of decision trees of large size is not efficiently PAC-learnable as their VC-dimension is directly proportional to their size. Various ideas to prune decision trees and reduce their size while maintaining reasonable empirical error (error on the training set) are used in practice. For a detailed treatment of the role of decision tree size in learning, the interested reader may refer to [23, Chapter 18]. We denote the decision tree size of a function f by $\mathsf{size}(f)$ and the decision tree size of its hazard-free extension by $\mathsf{size}_{\mathsf{u}}(f)$. We show that for the MUX function despite the exponential blow-up in depth from Theorem 6, the size blow-up is only polynomial.

▶ **Theorem 8.** $2 \cdot 4^n \leq \mathsf{size}_{\mathsf{u}}(\mathsf{MUX}_n) \leq 4^{n+1} - 3^n$.

In contrast, we show that there are functions for which the size blow-up is exponential. In particular, the AND_n function has linear-size Boolean decision trees. However, its hazard-free extension needs exponential-size decision trees.

▶ **Theorem 9.** $\mathsf{size}_{\mathsf{u}}(\mathsf{AND}_n) = 2^{n+1} - 1$.

We also show that there are hazard-free extensions of Boolean functions that require trees of size $\Omega\left(\binom{n}{n/3}\binom{2n/3}{n/3}\right)$ (see full version [4] for the details). Notice that a ternary tree of depth n can have at most 3^n leaves. This lower bound is only smaller than this worst-case by a polynomial factor.

We also show how to construct decision trees for hazard-free extensions of Boolean functions from a decision tree for the underlying Boolean function.

▶ **Theorem 10.** *For any Boolean function f, we have* $2\mathsf{size}(f) - 1 \leq \mathsf{size}_{\mathsf{u}}(f) \leq 2^{\mathsf{size}(f)} - 1$.

The tightness of the first inequality is witnessed by the PARITY_n function and that of the second inequality is witnessed by the AND_n function.

We also show that, in the case of hazard-free extensions too, sensitivity plays an important role in the function learning problem. The problem is as follows: We are provided a few input-output pairs and a guarantee that the function is from some family. The goal is to learn

the function from as few samples as possible. It is known that a function with sensitivity s is completely specified by its values on a Hamming ball of radius $2s$ [11]. We prove an analogue for hazard-free extensions of Boolean functions. We refer the reader to the full version [4] for details.

▶ **Theorem 11.** *A hazard-free extension f that has $\mathsf{s}_u(f) \leq s$ is specified by its values on any Hamming ball of radius $4s$ in $\{0, \mathsf{u}, 1\}^n$.*

1.1.4 Limited Uncertainty

We study computation in the presence of only a limited amount of uncertainty by introducing a parameter k that limits the number of bits for which the oracle can respond u. For metastability-containing electronic circuits, it is known that assuming only limited metastability allows constructing circuits that are significantly smaller [14]. We show a similar effect on decision tree size and query complexity when uncertainty is limited. See the full version [4] for details.

▶ **Theorem 12.** *Let T be a Boolean decision tree of size s and depth d for f. Then, there exists a decision tree of size at most $s^{2^{k+1}-1}$ and depth at most $2^k \cdot d$ for $\widetilde{f}$ provided that the input is guaranteed to have at most k positions with value u.*

For settings in which k is a small constant, observe that the decision tree size is polynomial in the size of the Boolean decision tree. If k is considered a parameter, observe that the depth remains fixed-parameter tractable in the language of parameterized complexity theory.

2 Preliminaries

▶ **Definition 13** (u-**query complexity**). *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function, and let $\widetilde{f}$ be its hazard-free extension. A* deterministic decision tree*, also called a* deterministic query algorithm*, computing $\widetilde{f}$, is a ternary tree T whose leaf nodes are labeled by elements of $\{0, \mathsf{u}, 1\}$, each internal node is labeled by a variable x_i where $i \in [n]$ and has three outgoing edges, labeled 0, 1, and u. On an input $x \in \{0, \mathsf{u}, 1\}^n$, the tree's computation proceeds from the root down to a leaf as follows: from a node labeled x_i, we take the outgoing edge labeled by value of x_i until we reach a leaf. The label of the leaf is the output of the tree $T(x)$.*

We say that T computes $\widetilde{f}$ if $T(x) = \widetilde{f}(x)$ for all $x \in \{0, \mathsf{u}, 1\}^n$. The deterministic u-query complexity of f, denoted $\mathsf{D}_\mathsf{u}(f)$, is defined as

$$\mathsf{D}_\mathsf{u}(f) := \min_T \operatorname{depth}(T),$$

where the minimization is over all deterministic decision trees T that compute $\widetilde{f}$.

▶ **Definition 14** (u-**sensitivity**). *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function, and let $\widetilde{f}$ be its hazard-free extension. For an $x \in \{0, \mathsf{u}, 1\}^n$, we define the u-sensitivity of f at x as:*

$$\mathsf{s}_\mathsf{u}(f, x) = |\{i \mid \exists y \in \{0, \mathsf{u}, 1\}^n \text{ s.t. } \widetilde{f}(y) \neq \widetilde{f}(x) \text{ and } y_j \neq x_j \text{ at only } j = i\}|$$

The elements i of the set are called the u-sensitive bits of x for f*. The* u-sensitivity of f*, denoted $\mathsf{s}_\mathsf{u}(f)$, is $\max_{x \in \{0,\mathsf{u},1\}^n} \mathsf{s}_\mathsf{u}(f, x)$.*

▶ **Definition 15** (u-**block sensitivity**). *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function and let $\widetilde{f}$ be its hazard-free extension. For $x \in \{0, \mathsf{u}, 1\}^n$, the u-block sensitivity of f at x is defined as maximum k such that there are disjoint subsets $B_1, B_2, \ldots B_k \subseteq [n]$ and for each $i \in [k]$,*

there is a y such that $\widetilde{f}(y) \neq \widetilde{f}(x)$ and y differs from x at exactly the positions in B_i. Each B_i is called a u*-sensitive block of f on input x. The* u*-block sensitivity of f, denoted $\mathsf{bs}_{\mathsf{u}}(f)$, is then defined as the maximum* u*-block sensitivity of f taken over all x.*

For $b \in \{0,1\}$, we use $\mathsf{bs}_{\mathsf{u},b}(f)$ to denote the maximum u-block sensitivity of f at x, over all $x \in f^{-1}(b)$. For a string $x \in \{0,\mathsf{u},1\}^n$ and a set $B \subseteq [n]$ that is a sensitive block of $\widetilde{f}$ at x, we abuse notation and use x^B to denote an arbitrary but fixed string $y \in \{0,\mathsf{u},1\}^n$ that satisfies $y_{[n]\setminus B} = x_{[n]\setminus B}$ and $\widetilde{f}(y) \neq \widetilde{f}(x)$.

We now formally define certificate complexity for hazard-free extensions of Boolean functions. We first define a *partial assignment* as follows: By a partial assignment on n bits, we mean a string $p \in \{0,1,\mathsf{u},*\}^n$, representing partial knowledge of a string in $\{0,\mathsf{u},1\}^n$, where the $*$-entries are yet to be determined. We say a string $y \in \{0,\mathsf{u},1\}^n$ is *consistent* with a partial assignment $p \in \{0,1,\mathsf{u},*\}^n$ if $y_i = p_i$ for all $i \in [n]$ with $p_i \neq *$.

▶ **Definition 16** (**u-certificate complexity**). *Let $f : \{0,1\}^n \to \{0,1\}$ and $x \in \{0,\mathsf{u},1\}^n$. A partial assignment $p \in \{0,1,\mathsf{u},*\}^n$ is called a* certificate *for $\widetilde{f}$ at x if*

- *x is consistent with p, and*
- *$\widetilde{f}(y) = \widetilde{f}(x)$ for all y consistent with p.*

The size of this certificate is $|p| := |\{i \in [n] : p_i \neq *\}|$. The domain *of p is said to be $\{i \in [n] : p_i \neq *\}$. The* certificate complexity *of $\widetilde{f}$ at $x \in \{0,\mathsf{u},1\}^n$, denoted $\mathsf{cc}_{\mathsf{u}}(f,x)$, is the minimum size of a certificate p for $\widetilde{f}$ at x. The* u-certificate complexity *of $\widetilde{f}$, denoted $\mathsf{cc}_{\mathsf{u}}(f)$, is the maximum value of $\mathsf{cc}_{\mathsf{u}}(f,x)$ over all x.*

In other words, a certificate for $\widetilde{f}$ at x is a set of variables of x that if revealed, guarantees the output of all consistent strings with the revealed variables to be equal to $\widetilde{f}(x)$.

▶ **Definition 17** (**Randomized u-query complexity**). *A randomized decision tree is a distribution over deterministic decision trees. We say a randomized decision tree computes $\widetilde{f}$ with error $1/3$ if for all $x \in \{0,\mathsf{u},1\}^n$, the probability that it outputs $\widetilde{f}(x)$ is at least $2/3$. The depth of a randomized decision tree is the maximum depth of a deterministic decision tree in its support. Define the randomized* u*-query complexity of f as follows.*

$$\mathsf{R}_{\mathsf{u}}(f) := \min_{T} \operatorname{depth}(T),$$

where the minimization is over all randomized decision trees T that compute $\widetilde{f}$ to error at most $1/3$.

We refer the reader to [6] for basics of quantum query complexity.

▶ **Definition 18** (**Quantum u-query Complexity**). *A quantum query algorithm $\mathcal{A}$ for $\widetilde{f}$ begins in a fixed initial state $|\psi_0\rangle$ in a finite-dimensional Hilbert space, applies a sequence of unitaries $U_0, O_x, U_1, O_x, \ldots, U_T$, and performs a measurement. Here, the initial state $|\psi_0\rangle$ and the unitaries $U_0, U_1, \ldots, U_T$ are independent of the input. The unitary O_x represents the "query" operation, and does the following for each basis state: it maps $|i\rangle|b\rangle|w\rangle$ to $|i\rangle|b + x_i \bmod 3\rangle|w\rangle$ for all $i \in [n]$ (here $x_i = \mathsf{u}$ is interpreted as $x_i = 2$, and the last register represents workspace that is not affected by the application of a query oracle).*

The algorithm then performs a 3-outcome measurement on a designated output qutrit and outputs the observed value.

We say that $\mathcal{A}$ is a bounded-error quantum query algorithm computing $\widetilde{f}$ if for all $x \in \{0,\mathsf{u},1\}^n$ the probability that $\widetilde{f}(x)$ is output is at least $2/3$. The (bounded-error) quantum u-query complexity *of $\widetilde{f}$, denoted by $\mathsf{Q}_{\mathsf{u}}(f)$, is the least number of queries required for a quantum query algorithm to compute $\widetilde{f}$ with error probability at most $1/3$.*

3 Sensitivity Theorem in the Presence of Uncertainty

We first show that sensitivity, block sensitivity, certificate complexity, and size of the largest prime implicant/prime implicate are all asymptotically the same. This is a more formal and more precise restatement of Theorem 3.

▶ **Theorem 19.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. Let k_1 be the size of a largest prime implicant of f, and k_2 be the size of a largest prime implicate of f. Then, we have:*

$$\max\{k_1, k_2\} \leq \mathsf{s_u}(f) \leq \mathsf{bs_u}(f) \leq \mathsf{cc_u}(f) \leq k_1 + k_2 - 1.$$

Proof. The inequalities $\mathsf{s_u}(f) \leq \mathsf{bs_u}(f) \leq \mathsf{cc_u}(f)$ follow from definitions as for their Boolean counterparts (see, for example, [6, Proposition 1]).

To show the first inequality, we crucially use our three-valued domain. Let P be a prime implicant of f. Define the input $y_P \in \{0, \mathsf{u}, 1\}^n$ to be 1 in those positions where P contains the corresponding positive literal, 0 where P has the corresponding negated literal, and u everywhere else. We claim that each index in $\{i \in [n] : x_i \in P \text{ or } \neg x_i \in P\}$ is sensitive for $\widetilde{f}$ at y_P. To see this, first observe that $\widetilde{f}(y_P) = 1$ since P being an implicant means f is 1 on all resolutions of y_P. Let $x_i \in P$ be a positive literal. Then, observe that if setting the i'th bit of y_P to a 0 does not change the value of $\widetilde{f}$, then changing the i'th bit to u would also not change the value of $\widetilde{f}$. This means $P \setminus \{x_i\}$ would also be an implicant of f, contradicting the fact that P was a *prime* implicant. The case when x_i appears as a negated literal in P is similar. Thus we have $\mathsf{s_u}(f) \geq k_1$. A similar argument can be made for prime implicates as well, showing that $\mathsf{s_u}(f) \geq k_2$. This proves the first inequality in the theorem.

To prove the last inequality, let $x \in \{0, \mathsf{u}, 1\}^n$ be any input to $\widetilde{f}$. Observe that if $\widetilde{f}(x) = 0$, then the prover can pick an implicate (of size at most k_2) that has all literals set to 0 in x, and reveal those values to the verifier. If $\widetilde{f}(x) = 1$, then the prover reveals an implicant (of size at most k_1) that is 1. If $\widetilde{f}(x) = \mathsf{u}$, then there must exist inputs $x^0, x^1 \in \mathsf{Res}(x)$ such that $f(x^0) = 0$ and $f(x^1) = 1$. Since both x^0 and x^1 are resolutions of x, it must hold that for all $i \in [n]$ where $x_i \in \{0,1\}$, $x^0_i = x^1_i = x_i$. i.e., x^0 and x^1 differ from x only in positions where x is u. Hence, a prime implicant that is 1 in x^1 must contain a u in x. Similarly, a prime implicate that is 0 in x^0 must contain a u in x. Thus, there exists a prime implicant and a prime implicate of f both of which contain a literal assigned u in x. The prover reveals the bits in such a prime implicant and prime implicate. Since every prime implicant and every prime implicate have at least one common variable, the prover reveals only $k_1 + k_2 - 1$ values. Why should this convince the verifier? Since x^0, x^1, and x coincide on positions where x has 0 or 1, it is possible to set the u in the revealed prime implicant to values that make the function output 1. Similarly, it is possible to make the function output 0 by filling in the values to the u positions in the prime implicate revealed. Thus, the verifier can check that there are indeed two valid resolutions that give different outputs. ◀

▶ Remark 20. Notice that Theorem 19 shows that in our setting, the parameters sensitivity, block sensitivity, and certificate complexity are equivalent (up to a multiplicative constant) to the largest prime implicants/prime implicates. In the Boolean world, for certificate complexity we get a tighter characterization in terms of CNF/DNF width, which is analogous to prime implicant/prime implicant size here. On the other hand, in the Boolean world, there is a quadratic separation between sensitivity and block sensitivity [22].

We now proceed to show that similar to their Boolean counterparts, the deterministic u-query complexity is polynomially upper bounded by u-sensitivity and deterministic, randomized, and quantum u-query complexities are all polynomially related to each other. We do this in two parts:

(i) $\mathsf{D}_{\mathsf{u}}(f) \leq \mathsf{cc}_{\mathsf{u}}(f) \cdot \mathsf{bs}_{\mathsf{u}}(f)$.
(ii) $\mathsf{bs}_{\mathsf{u}}(f) = O(\mathsf{R}_{\mathsf{u}}(f))$, $\mathsf{bs}_{\mathsf{u}}(f) = O(\mathsf{Q}_{\mathsf{u}}(f)^2)$.

We start by showing $\mathsf{D}_{\mathsf{u}}(f) = O(\mathsf{cc}_{\mathsf{u}}(f) \cdot \mathsf{bs}_{\mathsf{u}}(f))$. Algorithm 1 is a deterministic query algorithm that achieves this bound, as shown in Theorem 24. Following this, we derive (ii) in Lemma 27. The final relationships among the three u-query complexities is presented in Theorem 28.

Algorithm 1 u-query algorithm

Given: Known $f : \{0,1\}^n \to \{0,1\}$; Query access to an unknown $x \in \{0,\mathsf{u},1\}^n$.
Goal: Output $\widetilde{f}(x)$
Initialize partial assignment $x^* \leftarrow *^n$
for $i \leftarrow 1$ to $\max(\mathsf{bs}_{\mathsf{u},0}(\widetilde{f}), \mathsf{bs}_{\mathsf{u},1}(\widetilde{f}))$ **do**
    $c \leftarrow$ a minimum u-certificate of $\widetilde{f}$ at an arbitrary $x \in \widetilde{f}^{-1}(\mathsf{u})$ consistent with x^*
    $C \leftarrow$ domain of c
    Query all variables in C
    Update x^* with the answers from the oracle.
    if x^* is a u-certificate of $\widetilde{f}$ **then**
        Output u
    if x^* is a 0-certificate of $\widetilde{f}$ **then**
        Output 0
    if x^* is a 1-certificate of $\widetilde{f}$ **then**
        Output 1
Output u

We will need the following observations to prove correctness of the algorithm:

▶ **Observation 21.** *Let x^* be a partial assignment that does not contain a b-certificate of $\widetilde{f}$ for any $b \in \{0,\mathsf{u},1\}$. Then there exists a $z \in \widetilde{f}^{-1}(\mathsf{u})$ that is consistent with x^*.*

Proof. Since x^* does not already contain a b-certificate of $\widetilde{f}$ for any $b \in \{0,\mathsf{u},1\}$, this means that for each $b \in \{0,\mathsf{u},1\}$, there exists a setting to the $*$-variables in the partial assignment to yield a b-input to $\widetilde{f}$. Specifically, it must be the case that there exists some assignment to the undetermined $*$'s that sets an implicant of f to 1, and some assignment that sets an implicate of f to 0. Define the string z to be the same as x^* except with $*$'s replaced with u. It can be observed that $\widetilde{f}(z) = \mathsf{u}$. ◀

▶ **Observation 22.** *Let $f : \{0,1\}^n \to \{0,1\}$, $b \in \{0,1\}$ and $x \in \widetilde{f}^{-1}(b)$. Let c be a minimal certificate of $\widetilde{f}$ at x, and let its domain be C. Then for all $i \in C$, we have $c_i \neq \mathsf{u}$.*

Proof. Assume towards a contradiction an index $i \in [n]$ in a minimal certificate c for $\widetilde{f}$ at x with $c_i = \mathsf{u}$. By the definition of a certificate, and $\widetilde{f}$, the partial assignment c' obtained by removing i from the domain of c is such that all strings $x \in \{0,\mathsf{u},1\}^n$ consistent with c' satisfy $\widetilde{f}(x) = b$. This shows that c' is also a certificate for $\widetilde{f}$ at x, contradicting the minimality of c. ◀

▶ **Lemma 23.** *If Algorithm 1 reaches Line 15, then every 1-input of $\widetilde{f}$, and every 0-input of $\widetilde{f}$ is inconsistent with the partial assignment x^* (at Line 15).*

Proof. Let k denote $\max(\mathsf{bs}_{\mathsf{u},0}(\widetilde{f}), \mathsf{bs}_{\mathsf{u},1}(\widetilde{f}))$, the number of iterations of the **for** loop on Line 4. Assume the algorithm reached Line 15, and let x' be the partial assignment x^* constructed by the algorithm when it reaches Line 15.

Suppose, for the sake of contradiction, there exists an input $y \in \widetilde{f}^{-1}(1)$ that is consistent with x'. Since the algorithm reached Line 15, it must be the case that the partial assignment x' constructed does not contain a b-certificate for any $b \in \{0, \mathsf{u}, 1\}$ because otherwise one of the **if** conditions between Lines 9 and 11 would have terminated the algorithm. Then using Observation 21, there must also exist a u-input consistent with x'. Hence every time Line 5 was executed, there was indeed an $x \in \widetilde{f}^{-1}(\mathsf{u})$ consistent with x^*. Suppose the u-certificates used during the run of the **for** loop on Line 4 were $c_1, \ldots, c_k$, and their respective domains were $C_1, \ldots, C_k$.

The fact that neither of the **if** conditions fired means that every time the oracle was queried, the replies differed from the u-certificate being queried in at least one index each time. Let $B_i \subseteq C_i$ be the set of positions in C_i where x' differs from c_i. By the observations above, each B_i is non-empty. Observe that since c_{i+1} is chosen to be consistent with x^*, it must be the case that c_{i+1} and x^* agree on all positions in C_i. Hence B_{i+1} is disjoint from B_i. With the same reasoning, we can conclude that B_{i+1} is disjoint from every B_j where $j \leq i$.

Observe that for each $i \in [k]$, there is a setting to the bits in B_i such that x' becomes a u-input – simply take the setting of these bits from c_i, which is a u-certificate. More formally, for each $i \in [k]$, there exists a string $\alpha_i \in \{0,1\}^{|B_i|}$ such that $f(x'|_{B_i \leftarrow \alpha_i}) = \mathsf{u}$.

Since y is consistent with x', it agrees with x' on all positions where $x' \neq *$. This means the previous observation holds for y too. That is, for each $i \in [k]$, there exists strings $\alpha_i \in \{0,1\}^{|B_i|}$ such that $f(y|_{B_i \leftarrow \alpha_i}) = \mathsf{u}$. Recall that $y \in \widetilde{f}^{-1}(1)$, and hence the sets $B_1, \ldots, B_k$ form a collection of disjoint sensitive blocks for $\widetilde{f}$ at y. Further, since the algorithm has not found a 1-certificate (or 0-certificate) yet, it must be the case that there is some u-input consistent with x' by Observation 21. This means there is yet another disjoint block B_{k+1} that is sensitive for $\widetilde{f}$ at y. But this is a contradiction since the maximum, over all 1-inputs of $\widetilde{f}$, number of disjoint sensitive blocks is $\mathsf{bs}_{\mathsf{u},1}(\widetilde{f}) = k < k+1$.

A nearly identical proof can be used to show that every 0-input is inconsistent with x'. ◀

▶ **Theorem 24.** *Algorithm 1 correctly computes $\widetilde{f}$, and makes at most $O(\mathsf{cc}_{\mathsf{u}}(f)\mathsf{bs}_{\mathsf{u}}(f))$ queries. Thus $\mathsf{D}_{\mathsf{u}}(f) = O(\mathsf{cc}_{\mathsf{u}}(f) \cdot \mathsf{bs}_{\mathsf{u}}(f))$.*

Proof. If the algorithm outputs a value in $\{0,1\}$, then it must have passed the corresponding **if** condition (either in Line 11, or in Line 13 and is trivially correct. If the algorithm outputs u, then either the **if** condition on Line 9 must have passed, or Line 15 must have been reached. In the former case, the correctness of the algorithm is trivial. In the latter case, from Claim 23, we conclude that every 0-input and every 1-input of $\widetilde{f}$ must be inconsistent with the partial assignment x^* constructed by the algorithm when it arrives at Line 15. This means that every $x \in \{0, \mathsf{u}, 1\}^n$ that is consistent with x^* (in particular, the unknown input x) must satisfy $\widetilde{f}(x) = \mathsf{u}$, which concludes the proof of correctness.

The **for** loop runs for $\max(\mathsf{bs}_{\mathsf{u},0}(f), \mathsf{bs}_{\mathsf{u},1}(f)) = O(\mathsf{bs}_{\mathsf{u}}(f))$ many iterations, and at most $\mathsf{cc}_{\mathsf{u}}(f)$ many bits are queried in each iteration. ◀

We can now conclude that deterministic, randomized, and quantum u-query complexities are all polynomially related. We use Yao's minimax principle [25] and the adversary method for lower bounds on quantum query complexity due to Ambainis [2]. We state them below for convenience.

▶ **Lemma 25** (Yao's minimax principle)**.** *For finite sets D, E and a function $f : D \to E$, we have $\mathsf{R}(f) \geq k$ if and only if there exists a distribution $\mu : D \to [0,1]$ such that $\mathsf{D}_{\mu}(f) \geq k$.*

Here, $\mathsf{D}_\mu(f)$ is the minimum depth of a deterministic decision tree that computes f to error at most $1/3$ when inputs are drawn from the distribution μ.

▶ **Theorem 26** ([2, Theorem 5.1])**.** *Let D, E be finite sets, let n be a positive integer, and let $f : D^n \to E$ be a function. Let X, Y be two sets of inputs such that $f(x) \neq f(y)$ for all $(x, y) \in X \times Y$. Let $R \subseteq X \times Y$ be such that*

- *For every $x \in X$, there are at least m different $y \in Y$ with $(x, y) \in R$,*
- *for every $y \in Y$, there are at least m' different $x \in X$ with $(x, y) \in R$,*
- *for every $x \in X$ and $i \in [n]$, there are at most ℓ different $y \in Y$ such that $(x, y) \in R$ and $x_i \neq y_i$,*
- *for every $y \in Y$ and $i \in [n]$, there are at most ℓ' different $x \in X$ such that $(x, y) \in R$ and $x_i \neq y_i$.*

Then $\mathsf{Q}(f) = \Omega(\sqrt{\frac{mm'}{\ell\ell'}})$.

We lower bound randomized and quantum u-query complexities polynomially by u-block sensitivity.

▶ **Lemma 27.** *Let $f : \{0,1\}^n \to \{0,1\}$. Then,*

$$\mathsf{R}_\mathsf{u}(f) = \Omega(\mathsf{bs}_\mathsf{u}(f)), \qquad \mathsf{Q}_\mathsf{u}(f) = \Omega(\sqrt{\mathsf{bs}_\mathsf{u}(f)}).$$

Proof. We use Lemma 25 for the randomized lower bound. Let $x \in \{0, \mathsf{u}, 1\}^n$ be such that $\mathsf{bs}_\mathsf{u}(f) = \mathsf{bs}_\mathsf{u}(\widetilde{f}, x) = k$, with corresponding sensitive blocks $B_1, \ldots, B_k$. Define a distribution μ on $\{0, \mathsf{u}, 1\}^n$ as follows:

$$\mu(x) = 1/2,$$
$$\mu(x^{B_i}) = 1/2k \text{ for all } i \in [k].$$

Towards a contradiction, let T be a deterministic decision tree of cost less than $k/10$ that computes $\widetilde{f}$ to error at most $1/3$ under the input distribution μ. Let L_x denote the leaf of T reached by the input x. There are now two cases:

- If the output at L_x is not equal to $\widetilde{f}(x)$, then T errs on x, which contributes to an error of $1/2$ under μ, which is a contradiction.
- If the output at L_x equals $\widetilde{f}(x)$, since the number of queries on this path is less than $k/10$, there must exist at least $9k/10$ many blocks B_i such that no variable of x^{B_i} is read on input x^{B_i} (observe that in this case, x^{B_i} also reaches the leaf L_x). Since $f(x^{B_i}) \neq f(x)$, this means T makes an error on each of these B_i's, contributing to a total error of at least $9k/10 \cdot 1/2k = 0.45$ under μ, which is a contradiction.

This concludes the randomized lower bound.

For the quantum lower bound we use Theorem 26. Define $X = \{x\}, Y = \left\{x^{B_i} : i \in [k]\right\}$, and $R = X \times Y$. From Theorem 26 we have $m = k, m' = 1, \ell = 1$ (since each index appears in at most 1 block, as each block is disjoint) and $\ell' = 1$. Theorem 26 then implies $\mathsf{Q}_\mathsf{u}(f) = \Omega(\sqrt{k}) = \Omega(\sqrt{\mathsf{bs}_\mathsf{u}(f)})$. ◀

▶ **Theorem 28.** *For $f : \{0,1\}^n \to \{0,1\}$, we have*

$$\mathsf{D}_\mathsf{u}(f) = O(\mathsf{R}_\mathsf{u}(f)^2), \qquad \mathsf{D}_\mathsf{u}(f) = O(\mathsf{Q}_\mathsf{u}(f)^4).$$

Proof. Theorem 24 and Theorem 19 imply

$$\mathsf{D}_\mathsf{u}(f) = O(\mathsf{cc}_\mathsf{u}(f) \cdot \mathsf{bs}_\mathsf{u}(f)) = O(\mathsf{bs}_\mathsf{u}(f)^2) = O(\mathsf{R}_\mathsf{u}(f)^2),$$

where the final bound is from Lemma 27. Substituting the second bound from Lemma 27 in the last equality above yields $\mathsf{D}_\mathsf{u}(f) = O(\mathsf{Q}_\mathsf{u}(f)^4)$. ◀

4 Relationships

We start by proving that for monotone functions, the presence of uncertainty does not make computation significantly harder. Similar relationships are known for monotone combinational circuits w.r.t. containing metastability [14].

Proof of Lemma 4. Any query algorithm for $\widetilde{f}$ also computes f with at most the same cost, and hence $\mathsf{D}(f) \leq \mathsf{D}_{\mathsf{u}}(f)$. Similarly we have $\mathsf{R}(f) \leq \mathsf{R}_{\mathsf{u}}(f)$ and $\mathsf{Q}(f) \leq \mathsf{Q}_{\mathsf{u}}(f)$.

We start with a best (deterministic/randomized/quantum) query algorithm A for f, and an oracle holding an input $x \in \{0, \mathsf{u}, 1\}^n$ for $\widetilde{f}$. Now, for $b \in \{0, 1\}$, we define A_b to be the same algorithm as A, but whenever A queries the j^{th} bit of its input, it performs the following operation instead:

1. Query the j'th bit of x, denote the outcome by x_j.
2. If $x_j \in \{0, 1\}$, return x_j.
3. If $x_j = \mathsf{u}$, return b.

In the case of quantum query complexity, note that this operation can indeed be implemented quantumly making 2 queries to O_x. The initial query performs the instructions described above, and the second query uncomputes the values from the tuple we don't need for the remaining part of the computation. Note here that $O_x^3 = I$ by definition, and thus $O_x^2 = O_x^{-1}$, which is what we need to implement for the uncomputation operations.

Let $S_{\mathsf{u}} = \{i \mid x_i = \mathsf{u}\}$ be the positions that have u in x. Recall that $y^0 := x|_{S_u \leftarrow \vec{0}}$ (and $y^1 := x|_{S_u \leftarrow \vec{1}}$) is the input that has all the u in x replaced by 0s (by 1s respectively). Run A_0 and A_1 (possibly repeated constantly many times each to boost correctness probability) to determine the values of $f(y^0)$ and $f(y^1)$ with high probability. If $f(y^0) = 1$, then we output 1. Else if $f(y^1) = 0$, we output 0. Else we have $f(y^0) = 0$ and $f(y^1) = 1$, and we output u.

Correctness: First observe that for $b \in \{0, 1\}$, all answers to queries in the algorithm A_b are consistent with the input y^b. Next observe that in the poset (equivalently 'subcube') formed by the resolutions of x, the inputs y^0 and y^1 form the bottom and top elements respectively. Since f is monotone, if $f(y^0) = 1$, we can conclude that f is 1 on all resolutions of x. Similarly when $f(y^1) = 0$, it must be the case that f is 0 on all inputs in the poset. The remaining case is when $f(y^0) = 0$ and $f(y^1) = 1$. In this case, the inputs y^0 and y^1 are themselves resolutions of x with different evaluations of f, and hence the algorithm correctly outputs u.

By standard boosting of success probability using a Chernoff bound by repeating A_0 (A_1) constantly many times and taking a majority vote of the answers, we can ensure that the correctness probability of A_0 (A_1) is large enough, say at least 0.9. Thus, the algorithm described above has correctness probability at least $0.9^2 = 0.81$, and its cost is at most a constant times the cost of A. ◀

It is easy to prove using Theorem 19 that the u-query complexity of MUX_n is exponentially larger than its query complexity. We claim that $\mathsf{s}_{\mathsf{u}}(\mathsf{MUX}_n) \geq 2^n$. Consider the input where all selector bits are u and all data bits are 1. Flipping any data bit to zero changes the output from 1 to u. In the following theorem, we prove a stronger statement. A function is said to be *evasive* if its query complexity is the maximum possible. We first prove Theorem 6, i.e., MUX_n is evasive for all n. In fact, we prove a more general theorem. We consider decision trees for $\widetilde{\mathsf{MUX}_n}$ that are guaranteed to produce the correct output when the number of unstable values in the input is at most k, for an arbitrary $k \in [0, 2^n + n]$. We defer the proof of the following theorem to the full version [4].

▶ **Theorem 29.** *Let $k \in [0, 2^n + n]$. Any optimal-depth decision tree that correctly computes the hazard-free extension of* MUX_n *on inputs with at most k unstable values has depth exactly* $\min\{2^k + n, 2^n + n\}$.

5 Decision Tree Size and Limited Uncertainty

We show that despite requiring exponentially more depth in the presence of uncertainty, we can compute the function MUX_n using a decision tree that is only quadratically larger in size than the Boolean one (Theorem 8).

However, there are functions that require exponentially larger decision tree size in the presence of uncertainty such as the AND function (Theorem 9). We defer the proofs of these statements to the full version [4]. The size lower bound technique used here for both the functions involves constructing a set of inputs that must all lead to different leaves and hence any decision tree that computes the hazard-free extension correctly requires size at least as large as the size of this input set. The upper bound is shown by an explicit construction.

We present a general construction of decision trees of hazard-free extensions of functions from a decision tree of the underlying Boolean function (Theorem 10) in the full version [4] of this paper. The main idea is to construct the u subtree of the root node in the new decision tree from the 0 and 1 subtrees that have been constructed recursively. The u subtree construction can be viewed as a product construction where we replace each leaf in a copy of the 0 subtree with a modified copy of the 1 subtree. This product construction works because for every input that reaches the u subtree, the output value can determined by the output values when that bit is 0 and when it is 1, which is exactly what the 0 and 1 subtrees compute.

We also prove in Theorem 12 that the complexity increases only gradually along with the increase in the amount of uncertainty in the inputs. More specifically, we prove that if the inputs are guaranteed to have at most k bits with value u, without any guarantee on where they occur, the exponential blow-up in query complexity is contained to the parameter k (See full version [4] for a proof). This proof also makes use of the product construction mentioned above.

6 Discussion and Open Problems

In this paper we initiated a study of query complexity of Boolean functions in the presence of uncertainty, by considering a natural generalization of Kleene's strong logic of indeterminacy on three variables.

We showed that an analogue of the celebrated sensitivity theorem [13] holds in the presence of uncertainty too. While Huang showed a fourth-power relationship between sensitivity and block sensitivity in the Boolean world, we were able to show these measures are linearly related in the presence of uncertainty. The proof of sensitivity theorem in our setting is considerably different and easier from the proof of sensitivity theorem in the Boolean world. We can parameterize u-sensitivity and u-query complexity by restricting our attention to inputs that have at most k unstable bits. The setting $k = 0$ gives us the Boolean sensitivity theorem and the setting $k = n$, our sensitivity theorem. Can we unify these two proofs using this parameterization? That is, is there a single proof for the sensitivity theorem that works for all k?

We showed using u-analogues of block sensitivity, sensitivity, and certificate complexity that for all Boolean functions f, its deterministic, randomized, and quantum u-query

complexities are polynomially related to each other. An interesting research direction would be to determine the tightest possible separations between all of these measures. It is interesting to note that our quadratic relationship between deterministic and randomized u-query complexity improves upon the best-known cubic relationship in the usual query models. Moreover, our quartic deterministic-quantum relationship matches the best-known relationship in the Boolean world [1]. More generally, it would be interesting to see best known relationships between combinatorial measures of Boolean functions in this model, and see how they compare to the usual query model (see, for instance, [1, Table 1]). A linear relationship between deterministic and randomized query complexities in the presence of uncertainty remains open, but a quadratic deterministic-quantum separation follows from Theorem 6 or from the OR function (via Grover's search algorithm [12]).

While we studied an important extension of Boolean functions to a specific three-valued logic that has been extensively studied in various contexts, an interesting future direction is to consider query complexities of Boolean functions on other interesting logics. Our definition of $\widetilde{f}$ dictates that $\widetilde{f}(x) = b \in \{0,1\}$ iff $f(y) = b$ for all $y \in \mathsf{Res}(x)$, and $\widetilde{f}(x) = \mathsf{u}$ otherwise. A natural variant is to define a 0-1 valued function that outputs $b \in \{0,1\}$ iff majority of $f(y)$ equals b over all $y \in \mathsf{Res}(x)$. It is not hard to show that the complexity of this variant of MUX_n is bounded from below by the usual query complexity of Majority on 2^n variables, which is $\Omega(2^n)$ in the deterministic, randomized, and quantum query models.

References

1 Scott Aaronson, Shalev Ben-David, Robin Kothari, Shravas Rao, and Avishay Tal. Degree vs. approximate degree and quantum implications of huang's sensitivity theorem. In *Proceedings of 53rd Annual Symposium on Theory of Computing (STOC 2021)*, pages 1330–1342, 2021. doi:10.1145/3406325.3451047.

2 Andris Ambainis. Quantum Lower Bounds by Quantum Arguments. *J. Comput. Syst. Sci.*, 64(4):750–767, 2002. doi:10.1006/JCSS.2002.1826.

3 Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum Lower Bounds by Polynomials. *J. ACM*, 48(4):778–797, 2001. doi:10.1145/502090.502097.

4 Deepu Benson, Balagopal Komarath, Nikhil Mande, Sai Soumya Nalli, Jayalal Sarma, and Karteek Sreenivasaiah. Sensitivity and Query Complexity under Uncertainty, 2025. URL: https://arxiv.org/abs/2507.00148, arXiv:2507.00148.

5 Deepu Benson, Balagopal Komarath, Jayalal Sarma, and Nalli Sai Soumya. Hazard-free decision trees. *CoRR*, abs/2501.00831, 2025. doi:10.48550/ARXIV.2501.00831.

6 Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theor. Comput. Sci.*, 288(1):21–43, 2002. doi:10.1016/S0304-3975(01)00144-X.

7 Stephen Cook and Cynthia Dwork. Bounds on the time for parallel RAM's to compute simple functions. In *Proceedings of the 14th Annual Symposium on Theory of Computing (STOC 1982)*, page 231–233, 1982. doi:10.1145/800070.802196.

8 Stephen Cook, Cynthia Dwork, and Rüdiger Reischuk. Upper and Lower Time Bounds for Parallel Random Access Machines without Simultaneous Writes. *SIAM Journal on Computing*, 15(1):87–97, 1986. doi:10.1137/0215006.

9 Andrzej Ehrenfeucht and David Haussler. Learning Decision Trees from Random Examples. *Inf. Comput.*, 82(3):231–246, 1989. doi:10.1016/0890-5401(89)90001-1.

10 Steven Fortune and James Wyllie. Parallelism in Random Access Machines. In *Proceedings of the 10th Annual Symposium on Theory of Computing (STOC 1978)*, page 114–118, 1978. doi:10.1145/800133.804339.

11 Parikshit Gopalan, Noam Nisan, Rocco A. Servedio, Kunal Talwar, and Avi Wigderson. Smooth Boolean Functions Are Easy: Efficient Algorithms for Low-Sensitivity Functions. In

Proceedings of the Conference on Innovations in Theoretical Computer Science (ITCS 2016), page 59–70, 2016.

12 Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual Symposium on Theory of Computing (STOC 1996)*, pages 212–219, 1996.

13 Hao Huang. Induced subgraphs of hypercubes and a proof of the Sensitivity Conjecture. *Annals of Mathematics*, 190(3):949 – 955, 2019. doi:10.4007/annals.2019.190.3.6.

14 Christian Ikenmeyer, Balagopal Komarath, Christoph Lenzen, Vladimir Lysikov, Andrey Mokhov, and Karteek Sreenivasaiah. On the Complexity of Hazard-free Circuits. *Journal of the ACM (JACM)*, 66(4):1–20, 2019.

15 Christian Ikenmeyer, Balagopal Komarath, and Nitin Saurabh. Karchmer-Wigderson Games for Hazard-Free Computation. In *Proceedins of 14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*, volume 251, pages 74:1–74:25, 2023. doi:10.4230/LIPIcs.ITCS.2023.74.

16 Stasys Jukna. Notes on Hazard-Free Circuits. *SIAM J. Discret. Math.*, 35(2):770–787, 2021. doi:10.1137/20M1355240.

17 Stephen Cole Kleene. *Introduction to Metamathematics*. P. Noordhoff N.V., Groningen, 1952.

18 Nikhil S. Mande and Karteek Sreenivasaiah. Query Complexity with Unknowns. *CoRR*, abs/2412.06395, 2024. doi:10.48550/ARXIV.2412.06395.

19 Ron van der Meyden. Logical Approaches to Incomplete Information: A Survey. In *Logics for Databases and Information Systems*, pages 307–356. Kluwer, 1998.

20 Noam Nisan. CREW PRAMs and Decision Trees. *SIAM J. Comput.*, 20(6):999–1007, 1991. doi:10.1137/0220062.

21 Noam Nisan and Mario Szegedy. On the Degree of Boolean Functions as Real Polynomials. *Comput. Complex.*, 4:301–313, 1994. doi:10.1007/BF01263419.

22 David Rubinstein. Sensitivity vs. Block Sensitivity of Boolean functions. *Combinatorica*, 15(2):297–299, 1995.

23 Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning - From Theory to Algorithms*. Cambridge University Press, 2014.

24 Ingo Wegener. The Critical Complexity of All (Monotone) Boolean Functions and Monotone Graph Properties. *Inf. Control.*, 67(1-3):212–222, 1985. doi:10.1016/S0019-9958(85)80036-X.

25 Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, pages 222–227. IEEE Computer Society, 1977.