Circuits Resilient to Short-Circuit Errors

Klim Efremenko
Ben-Gurion University
Beersheba, Israel

Bernhard Haeupler
ETHZ
Zurich, Switzerland
Carnegie Mellon University
Pittsburgh, USA

Yael Tauman Kalai
Microsoft Research & MIT
Cambridge, USA

Pritish Kamath
Google Research
Mountain View, USA

Gillat Kol
Princeton University
Princeton, USA

Nicolas Resch
Centrum Wiskunde en Informatica
(CWI)
Amsterdam, Netherlands

Raghuvansh R. Saxena
Microsoft Research
Cambridge, USA

ABSTRACT

Given a Boolean circuit C, we wish to convert it to a circuit C' that computes the same function as C even if some of its gates suffer from *adversarial short circuit errors*, i.e., their output is replaced by the value of one of their inputs. Can we design such a resilient circuit C' whose size is roughly comparable to that of C? Prior work gave a positive answer for the special case where C is a formula.

We study the general case and show that *any* Boolean circuit C of size s can be converted to a new circuit C' of quasi-polynomial size $s^{O(\log s)}$ that computes the same function as C even if a $1/51$ fraction of the gates on any root-to-leaf path in C' are short circuited. Moreover, if the original circuit C is a formula, the resilient circuit C' is of near-linear size $s^{1+\epsilon}$. The construction of our resilient circuits utilizes the connection between circuits and DAG-*like communication protocols*, originally introduced in the context of proof complexity.

CCS CONCEPTS

• **Theory of computation → Communication complexity**.

KEYWORDS

Error Resilient Computation, Short Circuit Errors, Circuit Complexity.

ACM Reference Format:
Klim Efremenko, Bernhard Haeupler, Yael Tauman Kalai, Pritish Kamath, Gillat Kol, Nicolas Resch, and Raghuvansh R. Saxena. 2022. Circuits Resilient to Short-Circuit Errors. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC '22), June 20–24, 2022, Rome, Italy.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3519935.3520007

STOC '22, June 20–24, 2022, Rome, Italy

ACM ISBN 978-1-4503-9264-8/22/06…$15.00
https://doi.org/10.1145/3519935.3520007

1 INTRODUCTION

The study of *reliable computation* over unreliable components has both theoretical and practical significance and it is one of the oldest topics considered by theoretical computer science, dating back to an influential series of lectures by von Neumann in 1952 [30]. In the study of *noise resilient circuits*, we wish to convert a given circuit C to a circuit C' that computes the same function as C even if some of the gates of C' are faulty. Furthermore, we wish to do so with a *small overhead* in the size and depth, meaning that the size and depth of C' should be "close" to these of C.

In this paper we design fault tolerant Boolean circuits[1] with respect to *adversarial short-circuit errors*, an error model that was introduced by Kleitman, Leighton, and Ma [20]. First observe that if the adversary is allowed to corrupt gates arbitrarily, no circuit is resilient to even a single error, as the adversary can simply flip the result of the output gate. To prevent this, [20] allow the adversary to replace a gate by an arbitrary function g, as long as it satisfies $g(0,0) = 0$ and $g(1,1) = 1$.[2] This error model is practically motivated (see Section 1.3.1), and is equivalent to an error model where the value of a gate is replaced by the value of one of its children (the wire to the other child is "cut out"). We follow [18] and consider a strong noise model where an adversary can corrupt ("short circuits") at most a *constant fraction* of the gates on all root-to-leaf paths. In addition, we assume that the adversary is omniscient and has full information of the entire circuit. It can also choose a worst-case input and select which gates to corrupt and what to corrupt them based on this input.

Prior works exploring this model were either only able to handle a sub-constant error rate [20] (see Section 1.3) or considered a restricted family of circuits. Specifically, [3, 18] showed that any Boolean *formula*[3] can be converted to a noise resilient one with only a polynomial blowup in size. We note that since any circuit can be expanded out to an (exponentially larger) formula, their results can be used to convert any circuit to a noise resilient one,

[1] We assume fan-in 2 AND/OR gates and negations only at the leaves (i.e., the inputs to the circuit are $x_1, \ldots, x_n$ and $\bar{x}_1, \ldots, \bar{x}_n$).
[2] In particular, the adversary can replace any AND gate in the circuit with an OR gate and vice versa.
[3] Formulas are tree-shaped circuits (the fan-out of every gate is at most 1).

but will require an exponential blowup in size. Whether or not general circuits can be made noise resilient with a more modest overhead in size was left as an open problem (see [3, 18] and the excellent survey [14]).

1.1 Our Results

We answer this longstanding open problem in the positive, showing that any circuit can be made noise resilient with only a quasi-polynomial blowup in size.

Theorem 1.1 (Main result, informal). *Let C be a circuit with size s and depth d. There exists a circuit C' with size $s^{O(\log d)}$ and depth $O(d)$ that computes the same function as C even when a $\frac{1}{51}$-fraction of its gates on every root-to-leaf path are adversarially short circuited.*

We believe that a quasi-polynomial blowup in the circuit size is necessary when converting some circuits into noise resilient ones (see Section 2.3). In particular, we conjecture that, unlike in the case of formulas [3, 18], a polynomial overhead will not suffice. Unfortunately, proving so, even in an existential manner, may be currently out-of-reach as it would imply $\mathrm{P/poly} \not\subseteq \mathrm{NC}^1$; since $\mathrm{P/poly} \subseteq \mathrm{NC}^1$ implies that every circuit of polynomial size has an equivalent formula of polynomial size[4], and thus, due to [3, 18], also has an error-resilient formula of polynomial size.

In addition to the bound in Theorem 1.1, we are also able to bound the size of the obtained resilient circuit C' by $\mathrm{poly}(d)$ times the number of root-to-leaf paths in C. This allows us to show that if C is a formula, then C' has near-linear size $s^{1+\epsilon}$: Any formula can be converted to an equivalent balanced formula of near-linear size [1], and the number of root-to-leaf paths in any formula is at most its size.

Theorem 1.2. *Let C be a circuit of depth d and let p be the number root-to-leaf paths in C. There exists a circuit C' with size $p \cdot \mathrm{poly}(d)$ and depth $O(d)$ that computes the same function as C even when a $\frac{1}{51}$-fraction of its gates on every root-to-leaf path are adversarially short circuited.*

Corollary 1.3. *Let $\epsilon > 0$ and let C be a* formula *with size s. There exists a circuit C' with size $s^{1+\epsilon}$ that computes the same function as C even when a $\frac{1}{51}$-fraction of its gates on every root-to-leaf path are adversarially short circuited.*

We note that while Corollary 1.3 significantly improves the $\mathrm{poly}(s)$ blowup obtained by [3, 18], it is incomparable to their results as our noise-resilient circuit C' is not guaranteed to be a formula.

1.2 Resilient Circuits and Interactive Coding

Computation and communication are closely linked. Indeed, to construct their resilient formulas, [18] design corresponding robust communication protocols: Given a Boolean formula (or even a circuit) C of size s and depth d, they

(1) Apply the Karchmer-Wigderson transformation [19] to convert C to a communication protocol Π of length d for a related communication problem.
(2) Convert Π to a noise resilient protocol Π' of length $O(d)$ using tools from the field of *interactive coding* [26–28].
(3) Convert Π' to a noise resilient formula C' of depth $O(d)$ by proving a noisy version of the Karchmer-Wigderson theorem.

Since the size of C' is bounded by $2^{O(d)}$, when C is a formula we can balance it and get a $\mathrm{poly}(s)$-size resilient circuit. However, for general circuits, $2^{O(d)}$ may be exponential in s.

To circumvent this potential blowup in size, we use a generalization of the Karchmer-Wigderson theorem that shows an equivalence between circuit *size* and the size of a DAG*-like communication protocol* with a strong correctness guarantee, that we touch on below [25, 29]. Very roughly, a DAG-protocol can be viewed as a two-party pebble game over a (rooted) directed acyclic graph, where each non-leaf node is owned by one of the parties. When playing the game, the "pebble" starts at the root of this graph and is moved along the edges, where in each step the party who owns the vertex with the pebble moves it to one of its children. DAG-protocols were originally introduced in the context of proof complexity and they simplify to standard communication protocols when the underlying DAGs are trees.

To construct resilient circuits, we show how to convert DAG-protocols to noise resilient DAG-protocols that must operate correctly even if an adversary controls some of the nodes in the DAG and when the pebble lands on these nodes, the adversary chooses the child to progress to.

We next list some of the reasons that make the design of noise resilient DAG-protocols significantly more challenging than the design of standard noise resilient protocols:

Limited memory. In our game, the parties only know their input and the current location of the pebble. In particular, they may not know the path that led the pebble from the root to the current node, as in a DAG there may be multiple paths that lead to the same node. This can be interpreted as the parties not having sufficient memory to store the full transcript. Interactive coding schemes typically rely on the fact that the parties know the transcript. E.g., they often implement a "rewind-if-error" strategy, where the parties try to detect if an error occurred by comparing (hashes of) their transcripts, and then "rewind" to a point in the execution of the protocol before the error occurred, which is also determined using the transcripts.

Unreliable memory. In the setting of interactive coding, the communication between the parties is error-prone. However, each party is allowed to use its local memory (which may be limited, see Section 1.3.3), and this memory is always assumed to be reliable (not affected by noise). In contrast, in our noisy DAG-protocol model, the entire "memory" of the parties is given by the location of the pebble. Since the adversary can, in certain cases, move the pebble, it can tamper with the little memory the parties have.

Strong correctness guarantee. Perhaps the most challenging problem we encounter is the fact that the equivalence between circuits and DAG-protocols due to [25, 29] only holds provided that the DAG-protocols satisfy a very strong correctness guarantee, that we call *rectangular correctness*: In the case of standard communication protocols, the leaves of the protocol tree are labeled by potential outputs and the label of every leaf must be a correct solution for

[4] Recall that a function $f : \{0,1\}^n \to \{0,1\}$ has a formula of $\mathrm{poly}(n)$ size if and only if it has a circuit of $O(\log n)$ depth.

every input pair that reaches it. For DAG-protocols, we will sometimes require the label of a leaf to also be a correct solution for input pairs that do not reach it.

For a detailed overview of our efforts, see Section 2.

1.3 Related Work

1.3.1 Resilient Circuits. As is typically the case when modeling noise-resiliency (e.g., error correcting codes), the noise affecting the circuit can be modeled as either *stochastic* or as *adversarial*.

Stochastic noise. Von Neumann [30] studied the stochastic noise model, where the noise flips the value of each gate in the circuit independently with some small fixed probability. Von Neumann's model was studied by a long sequence of work, including [5–11, 16, 22, 23]. For example, in this model, it is known that a circuit of size s can be converted to a noise resilient circuit of size $O(s \log s)$, and that a function with sensitivity s' requires a resilient circuits of size $\Omega(s' \log s')$ [5, 10, 11, 22, 30].

Adversarial noise. The short-circuit fault model we adopt in this paper, where faulty gates output the value of one of their children, was introduced by [20]. As explained above, it is a simple error model that still allows for positive results in the adversarial setting. It is also motivated by applications - [20] note that "stuck-at" and "power-ground" failures resulting from short-circuits or broken connections are more common than other types of errors. As for results, [20] show that for any number of errors k, a circuit C of size s can be converted into a circuit C' of size $O(k \cdot s + k^{\log_2 3})$ that computes the same function as C provided that at most k of its gates are adversarially short-circuited. They also prove lower bounds on the size of resilient circuits and consider short-circuiting faults in the stochastic setting.

The task of making formulas resilient to short-circuit faults was considered by [3, 18]. In their model, an all-knowing adversary can short-circuit a constant fraction of the gates on all root-to-leaf paths. Observe that if the adversary is allowed to corrupt even a single root-to-leaf path in its entirety, it can force the output to equal the value of one of the input leaves by short circuiting the gates on the path leading to this leaf.

The main result of [18] is that a formula C of size s and depth d can be converted to a formula C' of size $\text{poly}(s)$ and depth $O(d)$ that computes the same function as C as long as at most $1/10 - \epsilon$ fraction of the gates on every root-to-leaf path are corrupted. Furthermore, the transformation of C to C' runs in $\text{poly}(s)$ time. The work of [3] shows that the maximum noise resilience of formulas is $1/5$: They give a polynomial size resilient circuit C' that can withstand $1/5 - \epsilon$ fraction of errors on every root-to-leaf path. In addition, they show that no circuit C' with sub-exponential blowup is resilient to $1/5$ fraction of errors on every root-to-leaf path.

The work of [12] studies a different adversarial model, where the adversary may corrupt the output of a small constant fraction of the gates at each layer of the circuit in an arbitrary way. By exploiting interesting connections between their model and the model of probabilistically checkable proofs, [12] were able to show that every symmetric function has a small resilient circuit. However, the obtained circuit is only guaranteed to compute, what they call, a "loose version" of the function, and may err on many inputs.

1.3.2 DAG-Protocols. Razborov [25] introduced a model of *PLS communication protocols*, and showed that it captures circuit *size*, generalizing the equivalence between the standard communication protocols and circuit/formula *depth* due to [19]. This connection was used by Krajíček [21], who introduced the technique of *monotone feasible interpolation*, which became a popular method for proving lower bounds on the refutation size in propositional proof systems such as Resolution, and Cutting Planes [2, 17], by reducing to monotone circuit lower bounds. The notion of PLS communication protocols was simplified by Pudlak [24] and Sokolov [29] to the notion of DAG-*like communication protocols*. Subsequently, a "converse" to monotone feasible interpolation was established in [13] to prove new lower bounds on monotone circuits by lifting lower bounds on Resolution refutations.

1.3.3 Interactive Coding. In the field of *interactive coding*, initiated by a seminal paper of Schulman [26], we wish to convert a given protocol Π that was designed to work over a noiseless channel, to a protocol Π' that works over a noisy channel. Various aspects of interactive codes (e.g., computational efficiency, interactive channel capacity, noise tolerance, list decoding, different channel types) were considered in recent years. See [14] for a survey.

Interactive coding with small memory. Motivated by the problem of constructing resilient circuits, [15] (which is an unpublished manuscript by a subset of the current authors and is an earlier version of this work) initiated the study of interactive codes that incur a small overhead in memory[5]. Building on [15], the work of [4] gives an interactive coding scheme that is resilient to a constant fraction of adversarial errors and only incurs an $O(\log d)$ overhead in the memory, where d is the length of Π. However, unlike our setting, the scheme of [4] assumes an *oblivious* adversary who makes all its decisions in advance (i.e., independently of the random choices of the interactive coding scheme or the communication history). Moreover, as explained above, to get our result, aside from dealing with small memory, we also need to deal with memory corruptions and rectangular correctness (see Section 1.2).

1.4 Open Problems

We next suggest several concrete directions for future work:

Transformation time. While the size of the resilient circuit C' we construct in Theorem 1.1 is bounded by $s^{O(\log d)}$, we don't know a similar bound on the running time of the transformation converting C to C'. Can such transformation run in polynomial time in the size of C', like in [3, 18]?

Lower bounds under assumptions. As discussed in Section 1.1, proving unconditional super-polynomial lower bounds on the size of resilient circuits will imply strong circuit lower bounds. However, such bounds may be within reach under assumptions. Is it possible to show that a quasi-polynomial overhead like in Theorem 1.1 is necessary in some "sufficiently rich" oracle model?

[5] A version of this manuscript can be found at https://arxiv.org/abs/1805.06872v1.

Maximum tolerance. The work of [3] shows that the maximum noise resilience of formulas under short-circuiting error is 1/5 (see Section 1.3). What is the maximum noise resilience of general circuits?

2 OUR TECHNIQUES

The work closest to the current one is that of [18], where an analogue of Theorem 1.1 holding only for Boolean formulas was shown. The work [18] does this in 3 steps:

(1) **Formulas → Protocols.** As their first step, [18] invoke the Karchmer-Wigderson transformation [19] that defines, for any Boolean function f, a related communication search problem[6] KW_f with the property that a Boolean formula computing f and of depth d is essentially equivalent to a communication protocol solving the search problem KW_f using d rounds. Moreover, having short circuit errors in the formula corresponds to running the protocol over a channel with corruption noise and perfect feedback[7]. We shall henceforth call these channels feedback channels for simplicity.
(2) **Protocols → Error Resilient Protocols.** As mentioned above, protocols that can run successfully even on feedback channels are equivalent to formulas that can work even when some gates are short circuited. In this step, [18] take the protocol for KW_f from Item 1 and convert it to a protocol that can run successfully even on feedback channels. This can be achieved using the by now standard tools from interactive coding.
(3) **Error Resilient Protocols → Error Resilient Formulas.** Now that we have a protocol for KW_f that can run successfully even on feedback channels, we can again use the Karchmer-Wigderson equivalence to devise from it a Boolean formula computing f that is resilient to short circuit errors. We note that this error-resilient version of the Karchmer-Wigderson transformation was also one of the results in [18].

We follow a similar high level blueprint. For the first step, we can no longer rely on the Karchmer-Wigderson transformation as it only works for Boolean formulas, and the trivial way to extend it to circuits by first expanding the circuit to a formula requires an exponential blowup in size, which Theorem 1.1 cannot afford. We get around this by using a generalization presented (separately) by Razborov and Sokolov [25, 29]. These show that a Boolean *circuit* of depth d and size s is equivalent to a DAG-like communication protocol with depth d and size s and a strong correctness guarantee, that we call *rectangular correctness*.

The aforementioned result is the analogue of Item 1 in our blueprint. We next describe the analogues for Items 2 and 3. The analogue of Item 3 is a result (see Theorem 4.4) that shows that the above equivalence due to Razborov and Sokolov also extends to the error resilient setting, and if we can show that there exist error resilient DAG-protocols that satisfy rectangular correctness, then they can be used to construct a circuit resilient to short circuit errors. This part of our proof conceptually follows [18] but is slightly more involved as it deals with more general objects.

The technical bulk of our proof goes in showing an analogue of Item 2. As mentioned above, this requires showing that a DAG-protocol (something more general than a protocol) satisfies rectangular correctness (something stronger than standard correctness) even in the presence of feedback errors. We describe our approach for this next.

2.1 DAG-Protocols

The most straightforward way to think of a DAG-protocol is to view it as a two-party pebble game over a (rooted) directed acyclic graph. Each leaf of the graph is labeled by an output while each non-leaf node is owned by either Alice or Bob, that in addition, have private inputs x and y respectively. A "pebble" starts at the root of this graph and is moved along the edges as follows: If currently the pebble is at a node v that is owned by Alice, then Alice uses her input x to select one of the out-edges of v for the pebble to take. The pebble then follows this edge and moves to the vertex that it leads to and the process continues. Ultimately, the pebble will reach a leaf and the output of the game will be the output of the leaf that the pebble reaches.

DAG-protocols generalize communication protocols. We now describe why the pebble game described above generalizes a communication protocol. For this, consider a communication protocol with some alphabet Σ and assume without loss of generality that the protocol is alternating with Alice speaking first and the output of the protocol is just its transcript. We argue that this communication protocol can be equivalently seen as a pebble game whose graph is just a complete $|\Sigma|$-ary tree with the even layers (including the root) owned by Alice and the odd layers owned by Bob. Furthermore, the output corresponding to a leaf is simply the unique path from the root to this leaf.

Now, if Alice's input requires her to send a symbol $\sigma \in \Sigma$ in the first round, then she can make the pebble follow the edge corresponding to σ in the pebble game. Similarly, if Bob wants to send $\sigma' \in \Sigma$ based on his input and the fact that he received σ, then he can direct the pebble to the edge corresponding to σ'. Proceeding this way, the pebble will just follow a path corresponding to the transcript of the protocol implying that the output of the pebble game will match the output of the protocol.

DAG-protocols and feedback errors. We also need to define the error model for DAG-protocols that corresponds to worst-case short circuit errors in the Boolean circuit. In this model, there is an adversary that is all-knowing and all-powerful: It knows the graph underlying the pebble game, the inputs of the parties, and controls some of the nodes in the graph. If the pebble ever lands on a node controlled by the adversary, then the edge it will take next will be determined by the adversary regardless of the inputs of the parties. We note that even though the parties do not control where the pebble goes, they do see where the adversary sent it.

In the case of a communication protocol, when the graph underlying the pebble game is a tree, such errors indeed correspond to feedback errors, as both the parties see where the pebble went

[6] Specifically, suppose that the function f maps the set $\{0,1\}^n$ to $\{0,1\}$. Then, in the search problem KW_f, Alice's input is an element $x \in \{0,1\}^n$ such that $f(x) = 1$ and Bob's input is an element $y \in \{0,1\}^n$ such that $f(y) = 0$. Their goal is to find a coordinate $i \in [n]$ satisfying $x_i \neq y_i$.

[7] In this model, some of the symbols sent during the protocol may be corrupted by an adversary, but the sender of each symbol gets to know via 'feedback' whether the symbol was received correctly or was corrupted, and in the latter case, also gets to know what it was corrupted to.

or equivalently, the sending party knows what symbol was received by the receiving party. This correspondence may lead one to think that tools from interactive coding that are used to make communication protocols error resilient can be extended to also make DAG-protocols error resilient. However, several problems arise during this extension.

The fundamental reason these problems arise is that when the underlying graph is not a tree, the current location of the pebble does not determine the path of the pebble from the root to that location. Essentially all interactive schemes crucially rely on the parties having the knowledge of this path or equivalently, remembering the sequence of symbols received. Indeed, if the parties know the symbols they received, they can use it along with their input to determine the sequence of symbols sent and check if any of the symbols sent by them was not received correctly by the other party. If there exists such a symbol, the parties can conclude that an error occurred and try to fix it.

Not being able to detect errors is a major problem. However, even if the parties can somehow tell whether or not an error happened, the fact that there are multiple (in fact, up to exponentially many) paths from the root to the current node would mean that the parties have multiple locations (on different paths) that they can rewind to. A subset of these locations that the parties can rewind to will be consistent with Alice's input while another subset will be consistent with Bob's input. Finding an element in the intersection that is not too far from the current node (as otherwise a small number of errors cause many rewinds) may require a lot of communication.

2.2 Rectangular Correctness

Not only do we need to work with DAG-protocols, a generalization of communication protocols, we actually need to show that DAG-protocols are rectangular correct, a notion much stronger than standard correctness, in order to eventually get error resilient circuits. We shall omit a precise definition of rectangular correctness in this sketch and will only provide a brief intuition (that is admittedly much weaker than the actual definition in Definition 3.4) by comparing it to the "standard" notion of correctness. For standard correctness, one requires that for all possible inputs x and y to Alice and Bob, the leaf of the graph reached by the inputs x and y is labelled with an output that is correct for x and y. Rectangular correctness is stronger, and requires that for all inputs x, x' to Alice and y, y' to Bob, any leaf that is reached when the inputs are x, y' and also reached when the inputs are x', y is labelled with an output that is correct for x and y (and also for x' and y', by symmetry), *even when* it may not be the leaf reached when the inputs are x and y.

Rectangular correctness – the bane. Note that if one restricts attention to the case $x = x'$ and $y = y'$, then the notion of rectangular correctness reduces to the notion of correctness, implying that it is indeed a stronger notion of correctness. It is in fact a very strong notion of correctness, and even without errors, converting a protocol that is correct (under the standard notion) to one that is also rectangular correct may require an exponential blowup. For example, note that for all functions f on n bits, the game KW_f can be solved (with standard correctness) by a pebble game of size linear in n, as all it requires is finding a coordinate where Alice's and Bob's input bits differ. A pebble game can first check if the first bit of Alice and Bob are the same, then check if the second bits are the same, and so on, solving the Karchmer-Wigderson game in $O(n)$ steps. However, if any such game KW_f can be solved with rectangular correctness by a pebble game of size less than an exponential, then the Razborov and Sokolov equivalence to circuits would imply that there is a circuit of size less than an exponential for any function f, a contradiction.

Rectangular correctness – the boon. On the other hand, the fact that circuits are equivalent to pebble games with rectangular correctness also helps us, as it ensures that the error-free pebble games that we start with have the strong rectangular correctness guarantee. To see this, let x and y be inputs for Alice and Bob and let $v_{x,y}$ be the leaf reached by the inputs x and y. Roughly speaking, rectangular correctness ensures that there are many leaves $\neq v_{x,y}$ in the tree that are labeled with outputs that are correct for x and y. This means that our error resilient version of this pebble game does not necessarily have to reach the leaf $v_{x,y}$, and it suffices for it to reach any of the leaves that are labeled with the same output. Focusing on this weaker requirement makes our task much easier, as explained next.

2.3 Building Error Resilient DAG-Protocols for Karchmer-Wigderson Games

Recall that our goal is to build error resilient DAG-protocols that are rectangular correct for all Karchmer-Wigderson games. For now, we drop the bane of rectangular correctness and focus only on getting standard correctness. We mention that even this would crucially rely on the boon of rectangular correctness for the error-free pebble game. Later, we shall adapt our techniques to also overcome the bane of rectangular correctness.

We wish to take inspiration from the interactive coding tools used in [18]. At an (extremely) high level, interactive coding schemes rely on two operations: (1) Detecting errors inserted by the adversary fast, and (2) rewinding appropriately in order to fix those errors once they are detected. As mentioned in Section 2.1, both these operations are hard to perform for general pebble games, however the boon of rectangular correctness gives us a way around.

Detecting errors fast. As mentioned in Section 2.2, instead of the stronger task of detecting errors, we shall use the definition of rectangular correctness to focus only on the weaker task of *detecting errors that can lead to an incorrect output.* For this, recall that the intuitive notion of rectangular correctness (provided in Section 2.2) has the nice property of being "independently verifiable" by Alice and Bob: Let x and y be the inputs of Alice and Bob and v be a leaf in the graph. Alice can go over all possible inputs y' for Bob and check if there exists one such y' that together with x, takes them to the leaf v. Similarly, Bob can go over all possible inputs x' for Alice and check if there is an x' that together with y will take them to v. If both the checks succeed, then rectangular correctness ensures that leaf v has the correct output for x and y. Crucially, this verification does not require any communication!

Even though this intuition for rectangular correctness is strictly weaker than the formal definition (see Definition 3.4) which proves a stronger guarantee for all (even non-leaf) nodes in the graph, it

does capture the fact that Alice and Bob can independently check whether any given node in the graph is correct for their inputs or not. If either of the checks fail, then the respective party knows that an error occurred and they need to rewind. Otherwise, if both the checks pass, there may still be errors but they will not lead the parties to an incorrect output, and they can continue with the rest of the pebble game.

The rewind mechanism. The next step after knowing whether or not to rewind is to determine where to rewind to. Here, we draw inspiration from the "meeting points" approach for interactive coding and adapt it to the setting of DAG-protocols. Let s and d be the size (*i.e.*, the number of vertices) and depth respectively of the DAG-protocol without errors and assume without loss of generality that the underlying graph is layered. In the meeting points approach, the error resilient DAG-protocol simulates the error-free DAG-protocol move by move, remembering, in addition to the current location v of the pebble in the error-free DAG-protocol, a set of $O(\log d)$ nodes in the error-free DAG-protocol on the path from the root to v[8].

The set of meeting points remembered are chosen to be exponentially spaced. That is, if currently the pebble is at depth d, then (roughly), there will be one meeting point at depths $\mathsf{d}-1, \mathsf{d}-2, \mathsf{d}-4$, and so on, implying a total of $O(\log d)$ meeting points. This means that at most $O(\log d)$ nodes need to be remembered together implying a size bound of $s^{O(\log d)}$ on the error-resilient DAG-protocol. Additionally, this ensures that, regardless of the number e of errors inserted by the adversary, there is always a meeting point at most $O(e)$ steps before that is remembered by the parties. Thus, e errors by the adversary can only hurt the pebble game by $O(e)$ rounds, resulting in resilience to a constant fraction of errors overall.

Note that if one decides to work with (asymptotically) fewer meeting points in an attempt to make the error-resilient pebble game smaller in size, then for all constants C, there must be two meeting points whose distances from d are more than a factor of C apart, say δ and $C\delta$. This means that the adversary can invest δ corruptions and ensure that the closest consistent meeting point is $C\delta$ rounds away, implying that the parties have to rewind at least $C\delta$ rounds. As δ corruptions cause $C\delta$ rewinds, such a protocol cannot be resilient to more than a $\frac{1}{C}$ fraction of errors. As this holds for all constants C, the protocol cannot be resilient to any constant fraction of errors. Thus, techniques like our must have a quasi-polynomial blowup in the size of the pebble game.

The variable $\mathtt{ct}$. An important subtlety in the above analysis is when the parties decide to rewind, say, 32 steps from depth d to reach the meeting point at depth $\mathsf{d}' = \mathsf{d} - 32$. Ideally, when the pebble is at depth d', we would like to have meeting points at depths $\mathsf{d}'-1, \mathsf{d}'-2, \mathsf{d}'-4, \cdots$ or equivalently, at depths $\mathsf{d}-33, \mathsf{d}-34, \mathsf{d}-36, \cdots$ remembered. However, the pebble just arrived here from depth d where these meeting points at these depths were not remembered. This lack of memory is a major problem, as if the adversary now spends one error to make the parties believe the node at depth d' is incorrect (when it is actually not), the closest meeting point they can rewind to is $\mathsf{d}' - 32 = \mathsf{d} - 64$, which is 32 rounds away. Thus, one error can hurt the pebble game by 32 rounds, or any other constant, and we cannot hope for resilience to a constant fraction of errors.

Again taking inspiration from interactive coding schemes, we handle this by maintaining a variable $\mathtt{ct}$, that is incremented whenever the parties detected an error. If the closest meeting point is D^* rounds away, the parties only rewind to that meeting point if $\mathtt{ct} \geq D^*$, whence they also decrease $\mathtt{ct}$ by D^*. This way, the only way the parties can rewind D^* steps from a correct node is if the adversary spends at least $O(D^*)$ errors, implying resilience to a constant fraction of errors. On the other hand, if the node the parties want to rewind from is actually incorrect, then the only reason parties do not have a meeting point D^* rounds away is if the parties went ahead at least $O(D^*)$ rounds from the incorrect node, which again happens only if the adversary spends at least $O(D^*)$ errors, again implying resilience to a constant fraction of errors.

2.4 Proving Rectangular Correctness

All the arguments above were only to show a pebble game that satisfies standard correctness in the presence of a constant fraction of adversarial errors. Recall that in order to get error-resilient circuits, we need the pebble game to satisfy the stronger notion of rectangular correctness. Ensuring this extra guarantee requires a non-trivial adaptation of the game above, which we touch on next.

Recall from Section 2.2 that rectangular correctness requires that for all inputs x, y for Alice and Bob, the output of all leaves v for which there exist x', y' such that v is reached when the inputs are x, y' and also reached when the inputs are x', y, should be correct for x and y. Extending this definition to the adversarial error setting, one gets that for all inputs x, y and all leaves v, if there exists an error pattern a, b that corrupt a small number of nodes (where a determines how the nodes owned by Alice are corrupted and b determines how the nodes owned by Bob are corrupted) and for which there exist x', y' and a', b' such that v is reached when the inputs are x, y' and errors are a, b' and also reached when the inputs are x', y and errors are a', b, should be correct for x and y.

Now, note that if the game is non-trivial, the graph must have a leaf v and inputs x and y such that the output of v is incorrect for x and y. By the above definition, this means that any such leaf must have a party, say Alice, such that when Alice's input is x, leaf v cannot be reached unless a lot of the nodes owned by Alice are corrupted, regardless of Bob's input or how many of the nodes owned by him are corrupted. This essentially means that we need to count Alice's and Bob's corruptions separately, and corruptions to one of the parties should not affect the other.

For this, we split the variable $\mathtt{ct}$ from the previous section into two variables $\mathtt{ct}_A$ and $\mathtt{ct}_B$, one controlled by Alice and the other controlled by Bob. In order to properly make this split, we also need to define two new variables $\mathtt{tct}_A$ and $\mathtt{tct}_B$ that roughly capture the number of nodes on the path that are controlled by Alice and Bob respectively[9]. We then carefully adapt our analysis from above to maintain and control these new variables, and use them to finish the proof.

[8] In the actual proof, this is implemented by constructing a graph of size $s^{O(\log d)}$, each of whose nodes determines a tuple of $O(\log d)$ nodes of the graph underlying the error-free DAG-protocol.

[9] The actual proof defines these to be total number of time $\mathtt{ct}$ is incremented, ignoring the times it is decreased, but the two versions are morally equivalent.

3 CIRCUITS AND DAG-PROTOCOLS

Circuits. We focus on Boolean circuits computing Boolean functions, namely circuits consisting of $\vee$, $\wedge$ and $\neg$ gates, and we assume without loss of generality that all negations are applied directly to the inputs.[10] Further, we assume without loss of generality that all the $\vee$ and $\wedge$ gates in the error-free circuit have fan-in 2. The size of a circuit C is the number of gates in the circuit and will be denoted by $|C|$.

Communication Search Problems. We consider search problems $S \subseteq \mathcal{X} \times \mathcal{Y} \times \mathcal{O}$ for finite input set $\mathcal{X} \times \mathcal{Y}$ and output set $\mathcal{O}$, where on input $(x, y) \in \mathcal{X} \times \mathcal{Y}$, the search problem is to find some output $o \in S(x, y) := \{o \in \mathcal{O} : (x, y, o) \in S\}$. For any Boolean function $f : \{0,1\}^n \to \{0,1\}$, Karchmer & Wigderson [19] introduced a corresponding search problem $\mathsf{KW}_f \subseteq \mathcal{X} \times \mathcal{Y} \times \mathcal{O}$, for $\mathcal{X} = f^{-1}(1)$, $\mathcal{Y} = f^{-1}(0)$ and $\mathcal{O} = [n]$, given as follows

Input: a pair $(x, y) \in f^{-1}(1) \times f^{-1}(0)$
Output: a coordinate $i \in [n]$ such that $x_i \neq y_i$

Note that KW_f is *total*, namely $\mathsf{KW}_f(x, y) \neq \emptyset$ for all $(x, y) \in f^{-1}(1) \times f^{-1}(0)$. Karchmer & Wigderson [19] established the following connection between the circuit depth of a function f and the communication complexity of KW_f.

Theorem 3.1 ([19]). *For all $f : \{0,1\}^n \to \{0,1\}$, the circuit depth of f is equal to the (deterministic) communication complexity of KW_f.*

An analogous connection was later established for *circuit size* by Razborov [25], and later simplified by Sokolov [29], using the notion of DAG-protocols. We next define DAG-protocols. However, we mention that the notion of DAG-like communication protocols we consider is equivalent to the definition in [13, 29], with a minor stylistic difference that we define the protocol in terms of the *message functions* for each party, whereas, previously it was defined directly in terms of the associated rectangles for each vertex. We find our style of definition to be more suitable for the purpose of designing error-resilient DAG protocols.

DAG-protocols. A DAG-protocol Π with inputs in $\mathcal{X} \times \mathcal{Y}$ and output in $\mathcal{O}$ is given as

$$\Pi = \Big(\Sigma, G = (V_A \sqcup V_B \sqcup V_O, E), \mathsf{rt}, \{h_v\}_{v \in V_A \sqcup V_B}, \{o_v\}_{v \in V_O}\Big),$$

where, Σ is a finite, non-empty alphabet, G is a (finite) directed acyclic graph with a designated root vertex rt and the vertex set partitioned into disjoint sets V_A, V_B, and V_O. The set V_O corresponds to the *sink* nodes (those with out-degree 0), V_A corresponds to nodes where Alice "speaks", and V_B corresponds to nodes where Bob "speaks". We use V to denote the set of all vertices $V = V_A \sqcup V_B \sqcup V_O$ (where $A \sqcup B$ denotes the union of *disjoint* sets A and B). All vertices in $V_A \sqcup V_B$ have out-degree at most $|\Sigma|$ and edges coming out of such a vertex are labeled by distinct elements in Σ; we use Σ_v to denote the labels on edges coming out of vertex v. For a vertex $v \in V_A \sqcup V_B$ and $\sigma \in \Sigma_v$, we shall denote by v_σ the vertex reached by following the out edge labeled σ from v (if one exists).

[10] Fact: Any Boolean circuit can be transformed to an equivalent Boolean circuit where all negations are applied directly to the inputs at the cost of at most doubling the number of gates.

For all $v \in V_A$, the "message function" $h_v : \mathcal{X} \to \Sigma$ encodes Alice's behavior at vertex v, and for all $v \in V_B$, $h_v : \mathcal{Y} \to \Sigma$ encodes Bob's behavior at vertex v. For ease of notation, we will often write $h_v : \mathcal{X} \times \mathcal{Y} \to \Sigma$ with the understanding that $h_v(x, y)$ only depends on x (resp. y) when $v \in V_A$ (resp. $v \in V_B$). It is required that $h_v(x, y) \in \Sigma_v$ for all v. Finally, every vertex $v \in V_O$ is labeled by an output value $o_v \in \mathcal{O}$.

The size of the DAG-protocol is $|\Pi| := |V|$, and the depth $\mathsf{d}(\Pi)$ is the length of the longest path starting from rt.

Execution of a DAG-protocol. The "execution" of a DAG-protocol Π corresponds to a labeling of all vertices $v \in V$ with rectangles $R_v := X_v \times Y_v \subseteq \mathcal{X} \times \mathcal{Y}$, following [13, 29].

Definition 3.2. For a DAG-protocol Π as above, we inductively, from the root, associate a *rectangle* R_v to each $v \in V$, as follows: $R_{\mathsf{rt}} = \mathcal{X} \times \mathcal{Y}$ and for all $v \neq \mathsf{rt}$, R_v is the smallest rectangle $X_v \times Y_v$ such that $R_v \supseteq \bigcup_{u \in V} \left\{(x, y) \in R_u : u_{h_u(x,y)} = v\right\}$.

Observation 3.3. *For a DAG-protocol Π as above with associated rectangles $\{R_v\}_{v \in V}$, it holds for all $v \neq \mathsf{rt} \in V$ and $x \in X_v$, there exists $u \in V$ and $y \in \mathcal{Y}$ such that $(x, y) \in R_u$ and $u_{h_u(x,y)} = v$. Similarly, for $y \in Y_v$, there exists a $u \in V$ and $x \in \mathcal{X}$ such that $(x, y) \in R_u$ and $u_{h_u(x,y)} = v$.*

Definition 3.4. We say a DAG-protocol Π is rectangular-correct w.r.t. a search problem $S \subseteq \mathcal{X} \times \mathcal{Y} \times \mathcal{O}$ if for all $v \in V_O$ and all $(x, y) \in R_v$, it holds that $o_v \in S(x, y)$.

Remark 3.5. We emphasize the correctness requirement of DAG-protocols is significantly stronger than that of memory-limited communication protocols [4]. Namely, memory-limited communication protocols (defined with the same tuple as a DAG-protocol) only require that $o_v \in S(x, y)$, where v is the *unique leaf* that is reached from the root on input (x, y), whereas, DAG-protocols require that $o_v \in S(x, y)$ for *every* leaf v such that $(x, y) \in R_v$, even if this leaf is not "reached from the root" on inputs (x, y).

Remark 3.6. Our definition of a DAG-protocol differs from [13, 29], in that, the latter is defined directly in terms of the associated rectangles, and not indirectly in terms of the message functions h_v as in ours. While these two definitions are essentially equivalent, our definition is more suitable for our context of designing error-resilient circuits.

Alternating DAG-protocols. A DAG-protocol Π of depth d is said to be *layered* if the vertices of Π can be partitioned into $d + 1$ layers indexed $0, \cdots, d$ such that the root rt is the only vertex in the layer 0, all the leaves are in layer d, and all edges in E go from one layer i to $i + 1$ for some i. Further, Π is said to be *alternating* if the even layers (including 0) form the set V_A (Alice's vertices) the odd layers form the set V_B (Bob's vertices), and the last layer forms the set V_O (sink vertices). Observe that at the cost of increasing the depth by a constant factor and increasing the size by a fixed polynomial, one can transform any DAG-protocol Π into another DAG-protocol Π' that is alternating such that if Π is rectangular-correct w.r.t. S, then so is Π'.

Trimmed DAG-protocols. Let Π be a DAG-protocol with associated rectangles $\{R_v\}_{v \in V}$. Let $(u, v) \in E$ be an edge, with $\sigma \in \Sigma$ being the label of (u, v), i.e., $v = u_\sigma$. We say that the edge (u, v) is *empty* if

$h_u(x,y) \neq \sigma$ for all $(x,y) \in R_u$. We say that a vertex v is *unreachable* if $R_v = \emptyset$. We use Π_{trim} to denote the *trimmed version* of Π, obtained by removing all empty edges and unreachable vertices. Observe that the associated rectangles of Π_{trim} are the same as that of Π (for vertices that remain). We say that Π is *trimmed* if $\Pi_{\mathsf{trim}} = \Pi$.

3.1 Equivalence between Circuits and DAG-Protocols

Theorem 3.7 ([25, 29]). *We have:*

(1) There is a transformation T that takes a circuit C and outputs a DAG-protocol Π with alphabet $[2]$ such that $|\Pi| = |C|$ and Π is rectangular-correct w.r.t. KW_f, where f is the function computed by C.

(2) There is a transformation T' that takes a DAG-protocol Π that is rectangular-correct w.r.t. KW_f for some Boolean function f, and outputs a circuit C such that $|C| \leq |\Pi|$ and C computes f.

In particular, it follows that for all Boolean functions f, the circuit-size of f is equal to the size of the smallest DAG-protocol that is rectangular-correct w.r.t. KW_f.

Proof. We prove only the first part as that is the only part we shall use. We also note that the proof for the second part is subsumed by the proof of Theorem 4.4. The transformation T on input C outputs a DAG-protocol $\Pi = \left([2], G = (V,E), \mathsf{rt}, \{h_v\}_{v \in V_A \cup V_B}, \{o_v\}_{v \in V_O}\right)$, such that:

- The graph G has the same structure as C, where rt corresponds to the output gate of C. The set V_A is the set of nodes corresponding to $\vee$ gates in C while V_B is the set of nodes corresponding to $\wedge$ gates. Recall that we assume that all the negation gates are applied directly to the inputs.
- For $v \in V_A$, $h_v(x)$ is the smallest $i \in [2]$ such that the gate corresponding to v_i in C evaluates to 1 on input x (defined arbitrarily if no such i exists). For $v \in V_B$, $h_v(y)$ is the smallest $i \in [2]$ such that the gate corresponding to v_i in C evaluates to 0 on input y (defined arbitrarily if no such i exists).
- The value $o_v \in [n]$ for $v \in V_O$ is coordinate input to the gate corresponding to v in C.

Clearly, $|\Pi| = |C|$ and we only have to show that Π is rectangular-correct w.r.t. KW_f, where f is the function computed by C. To this end, we prove by induction (starting from the root) that, for all $v \in V$, using v to also denote the function computed at the node corresponding to v in C, we have:

$$R_v \subseteq \{(x,y) \in \{0,1\}^n \times \{0,1\}^n : v(x) = 1 \ \wedge \ v(y) = 0\},$$

where R_v is as defined in Definition 3.2. This suffices due to Definition 3.4.

By definition, $R_{\mathsf{rt}} = \mathcal{X} \times \mathcal{Y} = f^{-1}(1) \times f^{-1}(0)$, and thus the base case holds. For the induction step, fix any node $v \in V$ and suppose the statement holds for all u such that $(u,v) \in E$. By Definition 3.2, for all $(x,y) \in R_v$, we have x', y', u', u'' such that $(x,y') \in R_{u'}$, $(x',y) \in R_{u''}$, $u'_{h_{u'}(x,y')} = v$, and $u''_{h_{u''}(x',y)} = v$. We now argue that $v(x) = 1$. The proof that $v(y) = 0$ is analogous.

As $(x,y') \in R_{u'}$, the induction hypothesis says that $u'(x) = 1$. If u' corresponds to an $\wedge$ gate, we must also have $v(x) = 1$ and there is nothing to show, so assume that u' corresponds to an $\vee$ gate. Equivalently, we have $u' \in V_A$ and together with $u'(x) = 1$ and $u'_{h_{u'}(x,y')} = v$, our definition of $h_{u'}$ implies that $v(x) = 1$ as desired. □

4 ERROR MODELS FOR CIRCUITS AND DAG-PROTOCOLS

4.1 Error Model for Circuits

We consider the short-circuit error model. Let C be a Boolean circuit with n inputs, and $V = V_\vee \sqcup V_\wedge$ be the set of all gates in C, where $V_\vee$ denotes the set of all $\vee$ gates and $V_\wedge$ denotes the set of all $\wedge$ gates. An error pattern for C is defined by a tuple $e = (a,b)$ where $a \in (V \cup \{*\})^{V_\vee}$ denotes a function mapping $v \in V_\vee$ to one of its children in V or to $*$, and $b \in (V \cup \{*\})^{V_\wedge}$ is defined analogously. For an error pattern $e = (a,b)$, we shall use e_v to denote a_v if $v \in V_\vee$ and b_v if $v \in V_\wedge$.

Intuitively, if e is an error pattern and v is a gate in C, then $e_v = *$ means that the gate v is error-free. On the other hand, if $e_v \neq *$, then e_v is equal to one of the children u of v in C, and this means that the gate v has been 'short-circuited' to u. Formally, given an input $z \in \{0,1\}^n$, an error pattern e, the value $v(z,e)$ computed at gate v is as follows: If v is a leaf, then $v(z,e)$ is the value of the literal of v on z. For an internal gate v,

$$v(z,e) := \begin{cases} e_v(z,e), & \text{if } e_v \neq * \\ \bigvee_{u:(v,u)\in E} u(z,e), & \text{if } e_v = * \text{ and } v \in V_\vee \\ \bigwedge_{u:(v,u)\in E} u(z,e), & \text{if } e_v = * \text{ and } v \in V_\wedge \end{cases}. \quad (1)$$

The (final) output of C is $C(z,e) = \mathsf{rt}(z,e)$, where rt is the output gate of C.

Definition 4.1 (Error resilient circuits)**.** Let $n > 0$, C be a Boolean circuit with n inputs, $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function, and $\mathcal{E}$ be a set of error patterns for C. We say that C computes f despite $\mathcal{E}$ if $C(z,e) = f(z)$ for all $z \in \{0,1\}^n$ and $e \in \mathcal{E}$.

4.2 Error Model for DAG-Protocols

The error model for DAG-protocols is defined similarly to the error model for (tree-like) communication protocols in the context of constructing error resilient formulas [3, 18]. Consider a DAG-protocol

$$\Pi = \left(\Sigma, G = (V_A \sqcup V_B \sqcup V_O, E), \mathsf{rt}, \{h_v\}_{v \in V_A \sqcup V_B}, \{o_v\}_{v \in V_O}\right).$$

An error pattern for Π is defined by a tuple $e = (a,b)$, where $a \in (\Sigma \cup \{*\})^{V_A}$ is a function mapping $v \in V_A$ to one of its out-edges (equivalently, children) and $b \in (\Sigma \cup \{*\})^{V_B}$ is defined analogously. Let $\mathcal{E}$ be a *rectangular* set of allowed error patterns for Π, *i.e.*, $\mathcal{E} = \mathcal{E}_A \times \mathcal{E}_B$, where $\mathcal{E}_A \subseteq (\Sigma \cup \{*\})^{V_A}$ and $\mathcal{E}_B \subseteq (\Sigma \cup \{*\})^{V_B}$. Define the DAG-protocol $\Pi_{\mathcal{E}}$ with inputs in $\mathcal{X}_{\mathcal{E}} \times \mathcal{Y}_{\mathcal{E}}$ (for $\mathcal{X}_{\mathcal{E}} := \mathcal{X} \times \mathcal{E}_A$ and $\mathcal{Y}_{\mathcal{E}} := \mathcal{Y} \times \mathcal{E}_B$) and output in O, as:

$$\Pi_{\mathcal{E}} = \left(\Sigma, G, \mathsf{rt}, \{h_{\mathcal{E},v}\}_{v \in V_A \sqcup V_B}, \{o_v\}_{v \in V_O}\right),$$

where $h_{\mathcal{E},v}$ for $v \in V_A \sqcup V_B$ is defined as:

$$h_{\mathcal{E},v}((x,a),(y,b)) = \begin{cases} a_v, & \text{if } v \in V_A \text{ and } a_v \in \Sigma \\ b_v, & \text{if } v \in V_B \text{ and } b_v \in \Sigma \\ h_v(x,y), & \text{otherwise} \end{cases}. \quad (2)$$

Intuitively, $\Pi_{\mathcal{E}}$ is a protocol defined on the same graph as Π with the same behavior except for the error pattern $(a,b) \in \mathcal{E}$. For every node $v \in V_A$, if $a_v = *$ then this node is not corrupted, and

if $a_v \in \Sigma$ then this node is corrupted, and Bob gets the signal that Alice proceeded to node v_{a_v} (independently of where she actually proceeded to).

Definition 4.2 (The search problem $S_{\mathcal{E}}$)**.** For Π and $\mathcal{E}$ as above and a search problem $S \subseteq \mathcal{X} \times \mathcal{Y} \times \mathcal{O}$, the search problem $S_{\mathcal{E}} \subseteq \mathcal{X}_{\mathcal{E}} \times \mathcal{Y}_{\mathcal{E}} \times \mathcal{O}$ is defined to be such that for all $(x,a) \in \mathcal{X}_{\mathcal{E}}$, $(y,b) \in \mathcal{Y}_{\mathcal{E}}$, and $o \in \mathcal{O}$, we have $((x,a),(y,b),o) \in S_{\mathcal{E}} \iff (x,y,o) \in S$.

Definition 4.3 (Error resilient DAG-protocols)**.** For Π and $\mathcal{E}$ as above and a search problem $S \subseteq \mathcal{X} \times \mathcal{Y} \times \mathcal{O}$, we say Π solves S despite $\mathcal{E}$ (or that Π is resilient to errors in $\mathcal{E}$) if $\Pi_{\mathcal{E}}$ is rectangular-correct w.r.t. $S_{\mathcal{E}}$.

4.3 The KW Transformation with Errors

In this subsection, we prove an error-resilient version of Theorem 3.7.

Theorem 4.4. *There is a transformation T^* that takes as input a DAG-protocol Π for which there exists a Boolean function f such that Π solves KW_f despite $\mathcal{E}$, and a rectangular set of error patterns $\mathcal{E}$ for Π, and outputs a circuit C and a set of error patterns $\mathcal{E}'$ for C such that:*

(1) $|C| \le |\Pi|$ and the fan-out of all gates in C is at most $|\Sigma|$.
(2) C computes f despite $\mathcal{E}'$.

Furthermore, for all $\theta > 0$, if d is the depth of Π and $\mathcal{E} = \mathcal{E}_A \times \mathcal{E}_B$ where $\mathcal{E}_A$ is the set of all $a \in (\Sigma \cup \{*\})^{V_A}$ such that $a_v \neq *$ for at most θd values of v on any root to leaf path in G, and $\mathcal{E}_B$ is defined analogously, then $\mathcal{E}'$ contains all error patterns $e' = (a', b')$ such that $a'_{v'} \neq *$ on at most θd values of v' on an input to output path in C and likewise for b'.

Proof. The transformation T^* on input Π and $\mathcal{E}$ has the following steps:

(1) Let $\Pi_{\mathcal{E}}$ be constructed as in Section 4.2. Trim $\Pi_{\mathcal{E}}$ to a protocol $\Pi_{\mathcal{E},\mathsf{trim}}$ as in Section 3. Define $\mathcal{E}_{\mathsf{trim}}$ to be the same as $\mathcal{E}$ but restricted to the vertices that were not trimmed, *i.e.*, we have $e' = (a', b') \in \mathcal{E}_{\mathsf{trim}}$ if and only if there exists $e = (a, b) \in \mathcal{E}$ such that a (resp. b) agrees with a' (resp. b') on all the vertices that were not trimmed. Observe that $\Pi_{\mathcal{E},\mathsf{trim}}$ has the same associated rectangle $R_{\mathcal{E},v} = X_{\mathcal{E},v} \times Y_{\mathcal{E},v}$ for vertex v as in $\Pi_{\mathcal{E}}$.
(2) Create a circuit C that has the exact same structure as $\Pi_{\mathcal{E},\mathsf{trim}}$ with the nodes in V_A replaced by $\vee$ gates, the nodes in V_B replaced by $\wedge$ gates, and the nodes $v \in V_O$ replaced by the literal z_{o_v}. If for some $v \in V_O$, there exists $((x,a),(y,b)) \in R_{\mathcal{E},v}$ that satisfies that $x_{o_v} = 0$ and $y_{o_v} = 1$, then negate the input to the input gate corresponding to v.
(3) Define the set $\mathcal{E}'$ to be the set of all (a', b') for which there exists $(a, b) \in \mathcal{E}_{\mathsf{trim}}$ such that a' is the same as a except that if a coordinate v was equal to $\sigma \in \Sigma$ in a, then that coordinate is now equal to v_σ in a', and the same holds for b' and b.

Item 1 holds straightforwardly. We show Item 2 by showing via induction (from the leaves up) that for all untrimmed nodes $v \in V$, and all $((x,a),(y,b)) \in R_{\mathcal{E},v}$, letting v also denote the corresponding gate in C and $\star$ denote the error pattern $*^{V_\wedge}$ or $*^{V_\vee}$ for the circuit C (exactly which will be clear from context), we have that:

$$v(x,(a,\star)) = 1 \qquad \text{and} \qquad v(y,(\star,b)) = 0. \tag{3}$$

This suffices as short circuiting an $\wedge$ gate cannot change the output from 1 to 0 and similarly short circuiting an $\vee$ gate cannot change the output from 0 to 1. For the base case, v is an input gate and Eq. (3) holds because of the way we negate the gates in the transformation T^* and the fact that Π solves KW_f despite $\mathcal{E}$. For the inductive step, we fix a node $v \in V_A$ (the case $v \in V_B$ is analogous). Let $\Sigma_v \subseteq \Sigma$ be the set of out-edges of v. As $\Pi_{\mathcal{E},\mathsf{trim}}$ is trimmed, for all $\sigma \in \Sigma_v$, there is an $((x,a),(y,b)) \in R_{\mathcal{E},v}$ such that $h_{\mathcal{E},v}((x,a),(y,b)) = \sigma$. As $v \in V_A$, we have that $h_{\mathcal{E},v}$ is a function of its first coordinate implying that $h_{\mathcal{E},v}((x,a),(y',b')) = \sigma$ for all $(y',b') \in \mathcal{Y}_{\mathcal{E}}$. By Definition 3.2, we have that:

$$X_{\mathcal{E},v} \subseteq \bigcup_{\sigma \in \Sigma_v} X_{\mathcal{E},v_\sigma} \qquad \text{and} \qquad Y_{\mathcal{E},v} \subseteq \bigcap_{\sigma \in \Sigma_v} Y_{\mathcal{E},v_\sigma}.$$

To see why this implies Eq. (3), note that for any $((x,a),(y,b)) \in R_{\mathcal{E},v}$, we have

$$\begin{aligned} &\exists \sigma \in \Sigma_v : ((x,a),(y,b)) \in R_{\mathcal{E},v_\sigma} \\ \forall \sigma \in \Sigma_v \ &\exists (x_\sigma, a_\sigma) \in \mathcal{X}_{\mathcal{E}} : ((x_\sigma, a_\sigma),(y,b)) \in R_{\mathcal{E},v_\sigma}. \end{aligned} \tag{4}$$

By our induction hypothesis, the second part of Eq. (4) implies $v_\sigma(y,(\star,b)) = 0$ for all $\sigma \in \Sigma_v$. It follows that $v(y,(\star,b)) = 0$. It remains to show $v(x,(a,\star)) = 1$. If $a_v = *$, this is because of the first part of Eq. (4) and the induction hypothesis. Otherwise, $a_v = \sigma^*$ for some $\sigma^* \in \Sigma_v$, then Eq. (2) implies that $((x,a),(y,b)) \in R_{\mathcal{E},v_{\sigma^*}}$ and an application of the induction hypothesis finishes the proof. □

5 CONSTRUCTING ERROR RESILIENT DAG-PROTOCOLS

In this section, we show a general transformation that maps a DAG-protocol to an error resilient DAG-protocol. Note that some of the proofs are deferred to the full version. Formally, we show that:

Theorem 5.1. *Let $\theta = \frac{1}{50}$ and $\epsilon > 0$. Let Π be a DAG-protocol of size s and depth d that is rectangular-correct w.r.t. a search problem S. There exists a DAG-protocol Π' (as defined in Section 5.2) such that*

(1) Π' has size $s' = s^{O(\log d)}$ and depth $d' = O(d)$.
(2) If p is the number of root-to-leaf paths in Π, then it holds that $s' = p \cdot \mathrm{poly}(d)$.
(3) Π' solves S despite $\mathcal{E} = \mathcal{E}_A \times \mathcal{E}_B$ where $\mathcal{E}_A \subseteq (\Sigma' \cup \{*\})^{V'_A}$ is the set of all $a \in (\Sigma' \cup \{*\})^{V'_A}$ such that $a_{v'} \neq *$ for at most $(\theta - \epsilon)d'$ values of v' on any root to leaf path in G', and $\mathcal{E}_B$ is defined analogously.

Observe that the transformation in Theorem 5.1 together with Theorems 3.7 and 4.4 proves Theorems 1.1 and 1.2. Our transformation is inspired by the "rewind-if-error" framework used in many interactive coding schemes, starting with the work of Schulman [26]. The basic idea is to communicate according to the original protocol until an error is detected, at which points the parties backtrack until they reach a point of agreement. Indeed this is precisely the error resilient protocol used in [18] for the standard tree-like communication protocols (in the context of constructing error-resilient formulas).

The main problem in our setting, is that a DAG-protocol can have many paths from the root to any vertex and hence we can no longer recall the entire transcript. We get around this problem

by remembering only $O(\log d)$ nodes on the path, and it is these nodes that the parties backtrack to. The nodes which we remember are carefully chosen, and are referred to as "meeting points" (In the actual proof, we use MP to denote these nodes and MPL to denote the depths[11] of these nodes) . We note that it is this additional storage that causes the blowup in the size of the DAG-protocol.

More precisely, for any node in the original DAG-protocol, the depth x of the node determines a set $\mathsf{MPL}(x)$ of size at most $O(\log d)$, such that for all $x' \in \mathsf{MPL}(x)$, at most one meeting point at depth x' is "remembered", *i.e.*, stored in MP, at this node, and no meeting point at a depth outside $\mathsf{MPL}(x)$ is remembered.

5.1 Meeting Points

First, we define and establish properties of the set $\mathsf{MPL}(\cdot)$. Throughout this section, for non-negative integers x, y, we shall use $\lfloor x \rfloor_y$ to denote the largest multiple of y that is at most x, *i.e.*, $\lfloor x \rfloor_y = \lfloor \frac{x}{y} \rfloor \cdot y$. The notation $[x, y]$ will denote the set of integers between x and y (including x and y) while $(x, y]$ will denote $[x, y] \setminus \{x\}$. The notations $[x, y)$ and (x, y) are defined similarly. We work with a fixed $z > 0$ in this section and consider the set $Z = [0, 2^z)$ of integers that can be represented using z bits.

5.1.1 Definition. We define the notion of meeting points:

Definition 5.2. Let $x \in Z$ and $j \in [z]$. The j^{th} meeting point of x is defined as

$$\mathsf{MPL}_j(x) = \max\{\lfloor x \rfloor_{2^{j-1}} - 2^{j-1}, 0\}.$$

We also define, for $S \subseteq [z]$, the notation $\mathsf{MPL}_S(x) = \{\mathsf{MPL}_j(x) \mid j \in S\}$. We shall omit the subscript S when $S = [z]$.

Observe that the j^{th} meeting point of x is at most 2^j away from x, and thus, the set of meetings points is (roughly) evenly spaced out geometrically. The term 2^{j-1} is subtracted to make the meeting points distinct for all j. The most important property of Definition 5.2 is that the function MPL changes in a very controlled way as x, the depth of the node our simulation is currently at, is updated. If we advance the simulation by one step and x increases by 1, then we have that $\mathsf{MPL}(x+1)$ is at most one element short of $\mathsf{MPL}(x) \cup \{x\}$. Similarly, we can also make precise claims (see Section 5.1.2) about what happens when our simulation rewinds to an earlier meeting point.

Next, we define some helpful notation concerning the binary representation of an integer $x \in Z$. For $x \in Z$, we define the set:

$$\mathsf{ones}(x) = \{j \in [z] \mid \lfloor x \rfloor_{2^j} < \lfloor x \rfloor_{2^{j-1}}\}. \tag{5}$$

In other words, $\mathsf{ones}(x)$ is the set of all positions (ordered from the least to the most significant) that are 1 in the binary representation of x. Observe that $\mathsf{ones}(x)$ is non-empty unless $x = 0$. The following properties are straightforward consequences of the above definitions.

Lemma 5.3. *We have* $\mathsf{MPL}(0) = \{0\}$. *For all* $x \neq 0 \in Z$ *and* $j \geq \max(\mathsf{ones}(x))$, *we have* $\mathsf{MPL}_j(x) = 0$.

Proof. That $\mathsf{MPL}(0) = \{0\}$ is direct. For the other part, note that $j \geq \max(\mathsf{ones}(x))$ implies $\lfloor x \rfloor_{2^{j-1}} \leq 2^{j-1}$ and use Definition 5.2. □

[11] More precisely, MPL will denote the "levels" of the nodes, and the level will be a deterministic function of the depth.

Lemma 5.4. *For all* $x \neq 0 \in Z$ *and all* $j < j' \in [\max(\mathsf{ones}(x))]$, *we have* $\mathsf{MPL}_j(x) > \mathsf{MPL}_{j'}(x)$.

Proof. We first prove the result with $j' = \max(\mathsf{ones}(x))$. As $j < j'$, we have that $x > 2^j$ and from Definition 5.2 and Lemma 5.3, we have $\mathsf{MPL}_j(x) > 0 = \mathsf{MPL}_{j'}(x)$. Now, consider the case $1 \leq j < j' < \max(\mathsf{ones}(x))$ and use $\mathsf{MPL}_j(x), \mathsf{MPL}_{j'}(x) > 0$ to get (by Definition 5.2):

$$\begin{aligned}\mathsf{MPL}_j(x) = \lfloor x \rfloor_{2^{j-1}} - 2^{j-1} &\geq \lfloor x \rfloor_{2^{j'-1}} - 2^{j-1} \\ &> \lfloor x \rfloor_{2^{j'-1}} - 2^{j'-1} = \mathsf{MPL}_{j'}(x).\end{aligned}$$

□

5.1.2 Maintaining Meeting Points. We next argue that Definition 5.2 ensures that the meeting points change in a fairly systematic way as the protocol proceeds. For instance, the following lemma shows that going from depth x to $x+1$ "loses" at most one meeting point (and also says which one), and gains one meeting point, which is the node at depth x.

Lemma 5.5. *Let* $x, x+1$ *be two consecutive elements of* Z. *Observe that* $\mathsf{ones}(x+1) \neq \emptyset$ *and define* $k = \min(\mathsf{ones}(x+1))$. *For all* $j \in [z]$, *we have:*

$$\mathsf{MPL}_j(x+1) = \begin{cases} x, & \text{if } j = 1 \\ \mathsf{MPL}_{j-1}(x), & \text{if } 1 < j \leq k \\ \mathsf{MPL}_j(x), & \text{if } k < j \leq z \end{cases}.$$

We can also compute what happens when our protocol backtracks from a node at depth x to a node at depth $x' \in \mathsf{MPL}(x)$. This is done in the following two lemmas, where the first lemma considers the case where we backtrack to the k'th meeting point for $k \in \mathsf{ones}(x)$, and the latter considers backtracking to the k'th meeting point where $k \in [z] \setminus \mathsf{ones}(x)$. For the latter lemma, it is sufficient to consider $k < \max(\mathsf{ones}(x))$ as otherwise, the k'th meeting point is 0.

Lemma 5.6. *Let* $x \in Z$, $k \in \mathsf{ones}(x)$ *define* $x' = \mathsf{MPL}_k(x)$. *If* $x' > 0$, *then, for all* $j \in [z]$, *we have:*

$$\mathsf{MPL}_j(x') = \begin{cases} x' - 2^{j-1}, & \text{if } j \in [k] \\ \mathsf{MPL}_j(x), & \text{if } k < j \leq z \end{cases}.$$

Observe that in the foregoing lemma, $\mathsf{MPL}_j(x')$ for $j \in [k]$ is not guaranteed to be in $\mathsf{MPL}(x)$, and thus may not be remembered at x, and therefore also not remembered at x'. This is okay because the meeting points are designed to be roughly geometrically apart, and the fact that our simulation went from x to x' means that it rewound roughly 2^k steps. Not remembering the meeting points for $j \in [k]$ just means that the rewinds will be "delayed" by an additional 2^k steps which is just a constant factor more.

Lemma 5.7. *Let* $x \in Z$ *such that* $x > 0$, *and let* $k \in [z] \setminus \mathsf{ones}(x)$ *satisfy* $k < \max(\mathsf{ones}(x))$. *Define* $x' = \mathsf{MPL}_k(x)$ *and* i^* *to be the smallest* $i \in \mathsf{ones}(x)$ *such that* $i > k$. *For all* $j \in [z]$, *we have:*

$$\mathsf{MPL}_j(x') = \begin{cases} x' - 2^{j-1}, & \text{if } 1 \leq j < k \\ \mathsf{MPL}_{j+1}(x), & \text{if } k \leq j < i^* \\ \max\{\lfloor x \rfloor_{2^{j-1}} - 2^j, 0\}, & \text{if } j = i^* \\ \mathsf{MPL}_j(x), & \text{if } i^* < j \leq z \end{cases}.$$

Lemma 5.7 follows from the following stronger lemma.

Lemma 5.8. *Let $x \in Z$ such that $x > 0$, and let $k \in [z] \setminus \mathsf{ones}(x)$ satisfy $k < \max(\mathsf{ones}(x))$. Define $x' = \mathsf{MPL}_k(x)$ and i^* to be the smallest $i \in \mathsf{ones}(x)$ such that $i > k$. For all $j \in [k, i^*]$, we have $\lfloor x \rfloor_{2^{j-1}} = \lfloor x \rfloor_{2^{i^*-1}}$. For all $j \in [0, z]$, we have*

$$\lfloor x' \rfloor_{2^j} = \begin{cases} x', & \text{if } 0 \le j < k \\ x' + 2^{k-1} - 2^j, & \text{if } k \le j < i^* \\ \lfloor x \rfloor_{2^j}, & \text{if } i^* \le j \le z \end{cases}.$$

Proof. The first part follows because $[k, i^*) \cap \mathsf{ones}(x) = \emptyset$ by our choice of i^* and Eq. (5). For the second part, we first derive several equivalent ways of writing x'. As $x' = \mathsf{MPL}_k(x) > 0$, we have from Definition 5.2 that $x' = \lfloor x \rfloor_{2^{k-1}} - 2^{k-1} = \lfloor x \rfloor_{2^{i^*-1}} - 2^{k-1}$ by the first part. As $i^* \in \mathsf{ones}(x)$, we can extend this using Eq. (5) to

$$x' = \lfloor x \rfloor_{2^{k-1}} - 2^{k-1} = \lfloor x \rfloor_{2^{i^*-1}} - 2^{k-1} = \lfloor x \rfloor_{2^{i^*}} + 2^{i^*-1} - 2^{k-1}.$$

To finish the proof of the claim, we use $x' = \lfloor x \rfloor_{2^{k-1}} - 2^{k-1}$ implying that x' is a multiple of 2^j for $j \in [0, k)$ for the first case. For the case $k \le j < i^*$, we use $x' = \lfloor x \rfloor_{2^{i^*-1}} - 2^{k-1}$ with the observation that $\lfloor x \rfloor_{2^{i^*-1}}$ is a multiple of 2^j. Finally, for $i^* \le j \le z$, we use $x' = \lfloor x \rfloor_{2^{i^*}} + 2^{i^*-1} - 2^{k-1}$ and get:

$$\lfloor x' \rfloor_{2^j} = \lfloor \lfloor x' \rfloor_{2^{i^*}} \rfloor_{2^j} = \lfloor \lfloor x \rfloor_{2^{i^*}} \rfloor_{2^j} = \lfloor x \rfloor_{2^j}.$$

□

5.2 The Error Resilient Protocol

We now define our transformation. The input to the transformation is a parameter $\epsilon > 0$ and a DAG-protocol:

$$\Pi = \left(\Sigma, G = (V_A \sqcup V_B \sqcup V_0, E), \mathsf{rt}, \{h_v\}_{v \in V_A \cup V_B}, \{o_v\}_{v \in V_0}\right).$$

Let ρ be a special 'rewind' symbol that is not an element of Σ and define $\Sigma' = \Sigma \cup \{\rho\}$. Our transformation outputs a new DAG-protocol

$$\Pi' = \left(\Sigma', G' = \left(V'_A \sqcup V'_B \sqcup V'_0, E'\right), \mathsf{rt}', \{h'_{v'}\}_{v' \in V'_A \cup V'_B}, \{o'_{v'}\}_{v' \in V'_0}\right).$$

We assume without loss of generality that $\epsilon = 2^{-k}$ for some integer $k > 0$. We shall also assume that the protocol Π is alternating and trimmed as defined in Section 3 and the number of layers d is a power of 2. We first augment Π by adding an alternating path (*i.e.*, a path where the nodes at even locations, starting from 0, are in V_A, and those at odd locations are in V_B) of length Kd to each of the leaves $v \in V_0$ where $K = 2^{2k} - 1$. These assumptions can be realized by increasing the depth and the number of root-to-leaf paths by a constant factor, the size by a factor of $O(d)$, and preserve rectangular-correctness (Definition 3.4). From now on, when we say Π and d, we refer to this new protocol. We observe that Π is still alternating and the depth d is still a power of 2.

As Π is alternating, the even layers (including 0) form the set V_A and the odd layers form the set V_B. We group every pair of consecutive internal layers, *i.e.*, all the layers except the last one, into a *level* and use $\mathsf{L}(v)$ to denote the level of an internal node v. Formally, we define $\mathsf{d}(v)$ to be the depth of a node v, and $\mathsf{L}(v) = \left\lfloor \frac{\mathsf{d}(v)}{2} \right\rfloor$. Thus, the root rt satisfies $\mathsf{d}(\mathsf{rt}) = \mathsf{L}(\mathsf{rt}) = 0$ and for all internal nodes v, the value of $\mathsf{L}(v) \in Z = [0, 2^z)$, where $z = \log_2 d$ is an integer. We shall apply the theory developed in Section 5.1 with these values of z and Z.

We now proceed to define the protocol Π' formally.

The set $V' = V'_A \sqcup V'_B \sqcup V'_0$. We define the set V' to be:

$$V' = V \times \mathcal{V} \times [0, 6d] \times [0, 6d] \times [0, 6d] \times [0, 6d],$$

where $\mathcal{V}$ is the set of all subsets of V of size at most $10z$. That is, each element of V' is a six-tuple, where (1) The first coordinate, denoted by $v(v')$ is an element of V. (2) The second coordinate, denoted by $\mathsf{MP}(v')$ is a non-empty subset of V, of size at most $10z$. (3) The remaining coordinates, respectively $\mathsf{ct}_A(v')$, $\mathsf{ct}_B(v')$, $\mathsf{tct}_A(v')$, and $\mathsf{tct}_B(v')$, are integers in $[0, 6d]$. We shall omit the argument v' from the above when it is clear from context. Note that $|V'| = |V|^{O(z)}$. We shall have $v' \in V'_0$ if either $v \in V_0$ or $\max\{\mathsf{tct}_A, \mathsf{tct}_B\} = 6d$. If $v' \notin V'_0$, we shall have $v' \in V'_A$ (respectively, V'_B) if $v \in V_A$ (respectively, V_B). We shall abbreviate $\mathsf{ct}_A + \mathsf{ct}_B$ and $\mathsf{tct}_A + \mathsf{tct}_B$ as ct and tct respectively.

The set E'. Each vertex $v' \in V'_A \cup V'_B$ has two kinds of edges coming out of it, the *forward* edges and the *rewind* edge. There is one forward edge for every out-edge of $v = v(v')$ and in addition, there is a single extra rewind edge. We first specify the forward edges. Let $\sigma \in \Sigma$ be such that v has an out-edge labeled σ. We define the edge corresponding to σ in G' by specifying the vertex v'_σ it leads to. This is:

$$v'_\sigma = (v_\sigma, \mathsf{MP}^{**}, \mathsf{ct}_A, \mathsf{ct}_B, \mathsf{tct}_A, \mathsf{tct}_B), \tag{6}$$

where:

$$\mathsf{MP}^{**} = \{u \in \mathsf{MP} \cup \{v\} \mid \mathsf{L}(u) \in \mathsf{MPL}(\mathsf{L}(v_\sigma)) \cup \{\mathsf{L}(v_\sigma)\}\}.$$

We now specify the rewind edge by specifying the vertex v'_ρ it leads to. We do this assuming $v' \in V'_A$ as the case $v' \in V'_B$ is symmetric. First, define u^* to be the element in $\mathsf{MP} \cap V_A$ that maximizes $\mathsf{L}(\cdot)$ and define $D^* = 2(\mathsf{L}(v) - \mathsf{L}(u^*)) \ge 0$. Note that we can equivalently write $D^* = \mathsf{d}(v) - \mathsf{d}(u^*)$ as both $v, u^* \in V_A$. Define:

$$v'_\rho = \begin{cases} (v, \mathsf{MP}, \mathsf{ct}_A + 1, \mathsf{ct}_B, \mathsf{tct}_A + 1, \mathsf{tct}_B), & \text{if } \mathsf{ct}_A + 1 < D^* \\ (u^*, \mathsf{MP}^*, \mathsf{ct}_A + 1 - D^*, \mathsf{ct}_B, \mathsf{tct}_A + 1, \mathsf{tct}_B), & \text{o.w.} \end{cases}, \tag{7}$$

where:

$$\mathsf{MP}^* = \{\mathsf{rt}\} \cup \{u \in \mathsf{MP} \mid \mathsf{d}(u) < \mathsf{d}(u^*) \wedge \mathsf{L}(u) \in \mathsf{MPL}(\mathsf{L}(u^*)) \cup \{\mathsf{L}(u^*)\}\}.$$

Intuitively, a rewind edge first looks for the largest $u^* \in \mathsf{MP}$ that it can rewind to. Then, if the counter ct_A is more than the difference D^* in the depths of u^* and v, it jumps to u^*, and otherwise, it increments the counter and stays at v. The reason we keep a counter instead of jumping to u^* in "one go" is that maybe this edge was taken due to an adversarial corruption, and one corruption should not make us go back by D^* in the depth. Also, note that, as $u^*, v \in V_A$, a rewind edge may lead to a node in V'_A from another node in V'_A and thus, the DAG-protocol Π' is *not* alternating. Our definitions above imply that for any edge $(u', v') \in E'$, we have

$$(2 \cdot \mathsf{tct}(v') + \mathsf{d}(v(v')) - \mathsf{ct}(v')) - (2 \cdot \mathsf{tct}(u') + \mathsf{d}(v(u')) - \mathsf{ct}(u')) = 1.$$

This together with the fact that $\mathtt{tct}(v') \leq 12d$ implies that the graph G' is acyclic and has depth

$$d' \leq 25d. \tag{8}$$

The root rt'. We define:

$$\mathsf{rt}' = (\mathsf{rt}, \{\mathsf{rt}\}, 0, 0, 0, 0)\,. \tag{9}$$

Throughout, we shall restrict attention to vertices v' such that there is a path from rt' to v' by removing all the other vertices.

The functions $\{h'_{v'}\}_{v' \in V'_A \cup V'_B}$. We state the definitions only for $v' \in V'_A$ as the definitions for $v' \in V'_B$ are analogous. Recall that $h'_{v'} : \mathcal{X} \times \mathcal{Y} \to \Sigma'$. Let $R_v = X_v \times Y_v$ be the associated rectangles of Π as in Definition 3.2. We have:

$$h'_{v'}(x,y) = \begin{cases} h_v(x,y), & \text{if } \forall u \in \mathsf{MP} \cup \{v\} : x \in X_u \\ \rho, & \text{otherwise} \end{cases}. \tag{10}$$

The values $\{o'_{v'}\}_{v' \in V'_0}$. Recall that $v' \in V'_0$ if either $v \in V_0$ or $\max\{\mathtt{tct}_A, \mathtt{tct}_B\} = 6d$. In the former case, we define $o'_{v'} = o_v$ while in the latter case, we define $o'_{v'}$ to be an arbitrary value $\in \mathcal{O}$.

5.2.1 Some Observations. The following observations follow from the foregoing definitions. We use $\mathbb{1}(E)$ to denote the indicator function for the condition E, *i.e.*, $\mathbb{1}(E) = 1$ if E holds and 0 otherwise.

Observation 5.9. *For all* $(u', v') \in E'$, *we have:*

(1) $\mathtt{tct}_A(v') - \mathtt{tct}_A(u') = \mathbb{1}\left(u' \in V'_A \wedge v' = u'_\rho\right)$.

(2) $\mathsf{L}(v(v')) - \mathsf{L}(v(u')) \leq \mathbb{1}\left(u' \in V'_B \wedge v' \neq u'_\rho\right)$.

(3) $(\mathsf{d}(v(v')) - \mathtt{ct}_A(v')) - (\mathsf{d}(v(u')) - \mathtt{ct}_A(u')) \leq \mathbb{1}\left(v' \neq u'_\rho\right) - \mathbb{1}\left(u' \in V'_A \wedge v' = u'_\rho\right)$.

The following observations make use of Definition 5.2 and the way E' is defined.

Observation 5.10. *For all* $v' \in V'$ *reachable from* rt', *and all* $u \in \mathsf{MP}$, *we have* $\mathsf{d}(u) \leq \mathsf{d}(v)$. *Moreover, the inequality is strict unless* $v = \mathsf{rt}$.

Observation 5.11. *For all* $v' \in V'$ *reachable from* rt', *we have* $\mathsf{d}(u) \neq \mathsf{d}(u')$ *for all* $u \neq u' \in \mathsf{MP}$. *We also have* $\mathsf{L}(u) \in \mathsf{MPL}(\mathsf{L}(v)) \cup \{\mathsf{L}(v)\}$ *for all* $u \in \mathsf{MP} \cup \{v\}$.

Note that Observation 5.11 implies that the number of vertices stored in $\mathsf{MP}(v')$ for any v' that is reachable from rt' is at most $10z$.

Lemma 5.12. *For all* $v' \in V'$ *reachable from* rt', *there exists a path from* rt *to* v *in* G *such that contains all vertices in* MP.

Proof. Proof by induction on the distance from rt' to v'. The base case is when $\mathsf{rt}' = v'$ and holds trivially from our definitions. For the inductive step, let $v' \neq \mathsf{rt}' \in V'$ and u' be arbitrary such that $(u', v') \in E'$. This gives us two cases based on whether the edge (u', v') is a forward or a rewind edge. In the former case, Eq. (6) holds and $v = u_\sigma$ for some $\sigma \in \Sigma$ and $\mathsf{MP} \subseteq \mathsf{MP}(u') \cup \{u\}$. The result now follows from the induction hypothesis on u'.

Assume now that the edge (u', v') is a rewind edge implying that Eq. (7) holds. If the first case of Eq. (7) is true, then $u = v$ and $\mathsf{MP}(u') = \mathsf{MP}$ and the result follows from the induction hypothesis. We can therefore assume that the second case of Eq. (7) is true. In this case $v = u^*$ for some $u^* \in \mathsf{MP}(u')$ and all vertices $w \in \mathsf{MP}$ satisfy $w \in \mathsf{MP}(u^*)$ and $\mathsf{d}(w) \leq \mathsf{d}(u^*)$. The result now follows from the induction hypothesis. □

REFERENCES

[1] Maria Luisa Bonet and Samuel R. Buss. 1994. Size-Depth Tradeoffs for Boolean Fomulae. *Inform. Process. Lett.* 49, 3 (1994), 151–155.

[2] Maria Luisa Bonet, Toniann Pitassi, and Ran Raz. 1997. Lower Bounds for Cutting Planes Proofs with Small Coefficients. *Journal of Symbolic Logic* 62, 3 (1997), 708–728.

[3] Mark Braverman, Klim Efremenko, Ran Gelles, and Michael A. Yitayew. 2019. Optimal Short-Circuit Resilient Formulas. In *Computational Complexity Conference (CCC)*, Vol. 137. 10:1–10:22.

[4] T.-H. Hubert Chan, Zhibin Liang, Antigoni Polychroniadou, and Elaine Shi. 2020. Small Memory Robust Simulation of Client-Server Interactive Protocols over Oblivious Noisy Channels. In *Symposium on Discrete Algorithms (SODA)*. 2349–2365.

[5] Roland L'vovich Dobrushin and SI Ortyukov. 1977. Upper bound on the redundancy of self-correcting arrangements of unreliable functional elements. *Problemy Peredachi Informatsii* 13, 3 (1977), 56–76.

[6] William S. Evans and Nicholas Pippenger. 1998. On the Maximum Tolerable Noise for Reliable Computation by Formulas. *IEEE Transactions on Information Theory* 44, 3 (1998), 1299–1305.

[7] William S. Evans and Leonard J. Schulman. 1999. Signal propagation and noisy circuits. *IEEE Transactions on Information Theory* 45, 7 (1999), 2367–2373.

[8] William S. Evans and Leonard J. Schulman. 2003. On the maximum tolerable noise of k-input gates for reliable computation by formulas. *IEEE Transactions on Information Theory* 49 (2003), 3094–3098.

[9] Tomás Feder. 1989. Reliable computation by networks in the presence of noise. *IEEE Transactions on Information Theory* 35, 3 (1989), 569–571.

[10] Péter Gács and Anna Gál. 1994. Lower bounds for the complexity of reliable Boolean circuits with noisy gates. *IEEE Transactions on Information Theory* 40, 2 (1994), 579–583.

[11] Anna Gál. 1991. Lower Bounds for the Complexity of Reliable Boolean Circuits with Noisy Gates. In *Foundations of Computer Science (FOCS)*. 594–601.

[12] Anna Gál and Mario Szegedy. 1995. Fault tolerant circuits and probabilistically checkable proofs. In *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*. 65–73.

[13] Ankit Garg, Mika Göös, Pritish Kamath, and Dmitry Sokolov. 2018. Monotone circuit lower bounds from resolution. In *Symposium on Theory of Computing (STOC)*. 902–911.

[14] Ran Gelles. 2017. Coding for Interactive Communication: A Survey. *Foundations and Trends in Theoretical Computer Science* 13, 1–2 (2017), 1–157.

[15] Bernhard Haeupler and Nicolas Resch. 2018. Coding for Interactive Communication with Small Memory and Applications to Robust Circuits. *arXiv* abs/1805.06872v1 (2018). arXiv:1805.06872v1

[16] Bruce E. Hajek and Timothy Weller. 1991. On the maximum tolerable noise for reliable computation by formulas. *IEEE Transactions on Information Theory* 37, 2 (1991), 388–391.

[17] Pavel Hrubeš and Pavel Pudlák. 2018. A note on monotone real circuits. *Inform. Process. Lett.* 131 (2018), 15–19.

[18] Yael Tauman Kalai, Allison B. Lewko, and Anup Rao. 2012. Formulas Resilient to Short-Circuit Errors. In *Foundations of Computer Science (FOCS)*. 490–499.

[19] Mauricio Karchmer and Avi Wigderson. 1988. Monotone circuits for connectivity require super-logarithmic depth. In *Symposium on Theory of Computing (STOC)*. 539–550.

[20] Daniel J. Kleitman, Frank Thomson Leighton, and Yuan Ma. 1997. On the Design of Reliable Boolean Circuits That Contain Partially Unreliable Gates. *J. Comput. System Sci.* 55, 3 (1997), 385–401.

[21] Jan Krajícek. 1997. Interpolation Theorems, Lower Bounds for Proof Systems, and Independence Results for Bounded Arithmetic. *Journal of Symbolic Logic* 62, 2 (1997), 457–486.

[22] Nicholas Pippenger. 1985. On networks of noisy gates. In *Foundations of Computer Science (FOCS)*. 30–38.

[23] Nicholas Pippenger. 1988. Reliable computation by formulas in the presence of noise. *IEEE Transactions on Information Theory* 34, 2 (1988), 194–197.

[24] Pavel Pudlák. 2010. On extracting computations from propositional proofs (a survey). In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, Vol. 8. 30–41.

[25] Alexander Razborov. 1995. Unprovability of lower bounds on circuit size in certain fragments of bounded arithmetic. *Izvestiya of the RAN* (1995), 201–224. Issue 1.

[26] Leonard J Schulman. 1992. Communication on noisy channels: A coding theorem for computation. In *Foundations of Computer Science (FOCS)*. IEEE, 724–733.

[27] Leonard J. Schulman. 1993. Deterministic Coding for Interactive Communication. In *Symposium on Theory of Computing (STOC)*. 747–756.

[28] Leonard J Schulman. 1996. Coding for interactive communication. *IEEE Transactions on Information Theory* 42, 6 (1996), 1745–1756.
[29] Dmitry Sokolov. 2017. Dag-Like Communication and Its Applications. In *Proceedings of the 12th Computer Science Symposium in Russia (CSR)*. Springer, 294–307.
[30] John Von Neumann. 1956. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies* (1956), 43–98.