Electronic Colloquium on Computational Complexity, Report No. 13 (2025)

Polynomial Size, Short-Circuit Resilient Circuits for NC

Yael Tauman Kalai[*]
Microsoft Research and MIT

Raghuvansh R. Saxena[†]
Tata Institute of Fundamental Research

Abstract

We show how to convert any circuit of poly-logarithmic depth and polynomial size into a functionally equivalent circuit of polynomial size (and polynomial depth) that is resilient to adversarial short-circuit errors. Specifically, the resulting circuit computes the same function even if up to ϵd gates on every root-to-leaf path are short-circuited, *i.e.*, their output is replaced with the value of one of its inputs, where d is the depth of the circuit and $\epsilon > 0$ is a fixed constant.

Previously, such a result was known for formulas (Kalai-Lewko-Rao, FOCS 2012). It was also known how to convert general circuits to error resilient ones whose size is quasi-polynomial in the size of the original circuit (Efremenko et al. STOC 2022). The reason both these works do not extend to our setting is that there may be many paths from the root to a given gate, and the resilient circuits needs to "remember" a lot of information about these paths, which causes it to be large. Our main idea is to reduce the amount of this information at the cost of increasing the depth of the resilient circuit.

[*]yaelism@gmail.com.

[†]raghuvansh.saxena@gmail.com. Supported by the Department of Atomic Energy, Government of India, under project no. RTI4001.

ISSN 1433-8092

Contents

1 Introduction

Constructing error-resilient circuits is one of the most fundamental problems in theoretical computer science, dating back to von Neumann [vN56a]. In the study of error resilient circuits, the goal is to convert any circuit C into a circuit C' that computes the same function as C even if some of the gates of C' are faulty. Moreover, the goal is to do so with a small overhead in size, *i.e.*, constructing a circuit C' whose size is bounded by a polynomial in the size of C.

In this paper, we show how to convert any polynomial size circuit of poly-logarithmic depth into a polynomial size circuit that is functionally equivalent and is resilient to short-circuit errors, an error model that was introduced by Kleitman, Leighton, and Ma [KLM97], and was soon adopted as a central model in the study of error-resilient circuits [KLR12, BEGY19, EHK$^+$22]. In this error model, the adversary can replace any gate in the circuit with an arbitrary gate $g : \{0,1\}^2 \to \{0,1\}$ as long as $g(0,0) = 0$ and $g(1,1) = 1$ (and in particular can replace an AND gate with an OR gate, and vice versa)[1]. Equivalently, we can say that the value of the gate g is replaced by the value of one of its children (the wire to the other child is "cut out"). This model is motivated by applications; indeed, as noted in [KLM97], "stuck-at" and "power-ground" failures resulting from short-circuits or broken connections are more common than other types of errors.

Similar to the works of [KLR12, BEGY19, EHK$^+$22], we consider omniscient adversaries that have full information about the entire circuit, and can corrupt ϵd of the gates on every root-to-leaf path, where d is the depth of the circuit and $\epsilon > 0$ is some fixed constant. These prior works were either restricted to formulas, *i.e.*, logarithmic depth circuits [KLR12, BEGY19], or incurred a quasi-polynomial blowup to the circuit size [EHK$^+$22]. Moreover, it was conjectured in [EHK$^+$22] that, unlike the case of formulas [KLR12, BEGY19], a quasi-polynomial blowup in the circuit size is necessary when converting general circuits into noise resilient ones.

1.1 Our Results

We show that how to convert any circuit of polynomial size and poly-logarithmic depth (i.e., any NC circuit) into a functionally equivalent circuit of polynomial size (and polynomial depth) that is error resilient.

Theorem 1.1 (Informal, see formal version in Theorem 3.6)**.** *Let C be a Boolean circuit in the class NC that*[2] *computes a function f. There exists a Boolean circuit C' whose size and*

[1] Note that the restriction that $g(b,b) = b$ for all $b \in \{0,1\}$ is necessary since otherwise an adversary can simply flip the result of the output gate, and thus no circuit can be resilient to even a single error.

[2] Thus, if f takes n variables as input, the number of gates in C is bounded by a polynomial in n and the depth of C is bounded by a polynomial in $\log n$. The other constants in the theorem statement depend on the degree of these polynomials; see the formal version of the theorem (Theorem 3.6) for the precise dependence.

depth are polynomial in the size of C that computes f even when a constant fraction of gates in any input to output path in C' are short-circuited.

In terms of comparison, Theorem 1.1 improves on the main result of [KLR12] in the sense that it works for all circuits with polynomial size and poly-logarithmic depth while [KLR12] only works for formulas. It also improves upon [EHK+22] in the sense that it gets a polynomial blowup in the size instead of the quasi-polynomial blowup obtained in [EHK+22]. However, it does not subsume either of these results. For example, [KLR12] also preserve the depth of the original formula up to constant factors but we do not provide such a guarantee, and [EHK+22] works for general circuits, including those not in the class NC.

We also mention that [EHK+22] believed that the quasi-polynomial blowup they obtain is inherent and conjectured that a polynomial blowup will not suffice like it does in the case of formulas [KLR12, BEGY19]. Theorem 1.1 disproves this conjecture for the restricted case of circuits in NC. Proving/disproving the conjecture for general circuits is an amazing problem that we leave open. However, proving this conjecture, even in an existential manner, would imply $\mathsf{P/poly} \not\subseteq \mathsf{NC}$, and thus may be extremely hard. Indeed, if $\mathsf{P/poly} \subseteq \mathsf{NC}$, then a general polynomial size circuit has a functionally equivalent circuit in NC and we can use Theorem 1.1 to make it error resilient with only a polynomial blowup.

Error-resilient DAG-protocols. Computation and communication have a deep connection. Just like prior work [KLR12, BEGY19, EHK+22], the current work also exploits this connection via the Karchmer-Wigderson transformation. Specifically, Karchmer and Wigderson [KW88] show that any circuit computing a function f can be converted into a communication protocol for a related search problem KW_f such that the depth of the circuit matches the length of the communication protocol, and vice versa. An analogous transformation can be shown in the reverse direction, *i.e.* a transformation that takes error-resilient communication protocols to error resilient circuits, and was used in [KLR12, BEGY19] to build error resilient formulas.

The Karchmer-Wigderson transformation does not preserve the size of the circuit and in fact, blows it up to be exponential in the depth. To circumvent this blowup in size, [Raz95, Sok17] showed how to convert any circuit into a "DAG-protocol" that preserves both its size and depth. DAG-protocols are a generalization of communication protocols, and can be represented by the following pebble game: There is a rooted directed acyclic graph, and each non-sink node in the graph belongs to one of the two parties, while each sink node is associated with an output. The game starts with a "pebble" at the root of graph, that is moved along the edges of the graph. At every step, if the pebble is not already at a sink node, the party who owns that node will move the pebble along one of its out-edges. This will end when the pebble reaches a sink node, and the output will be the output of the sink node.

[Raz95, Sok17] showed that every circuit computing a function f can be converted to a DAG-protocol with the same size and depth that solves KW_f with a strong correctness guarantee called rectangular correctness (see Definition 3.3 for a precise definition). The reverse direction also holds, and in fact even holds in the error-resilient setting. This direction was used to get error-resilient circuits from error-resilient DAG-protocols in [EHK+22]. We adopt a similar approach and use the [Raz95, Sok17] transformation to go from circuits to DAG-protocols and back. Indeed, our main technical result is the following theorem about error-resilient DAG-protocols.

Theorem 1.2 (Informal, see formal version in Theorem 3.7)**.** *Let S be a search problem and Π be a DAG-protocol that solves S with rectangular correctness. Assume that the depth of Π is poly-logarithmic in the size. There exists a DAG-protocol Π' with a polynomially larger size that solves S with rectangular correctness even if a constant fraction of nodes on any root to leaf path in the protocol Π' are adversarially corrupted.*

1.2 Related Work

The works most closely related to ours are the works [KLR12, BEGY19, EHK+22] cited above. We discuss additional related work below.

Other noise models for circuits. Even though the short-circuit error model we consider is the most studied one, other models have also been considered. For example, von Neumann [vN56b] initiated the study of 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 works, including [DO77, Pip85, Pip88, Fed89, EP98, ES99, HW91, Gál91, GG94, ES03]. In this model, it is known that a circuit of size s can 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 $\Omega(s' \log s')$ [vN56b, DO77, Pip85, Gál91, GG94].

In this paper, we study the short-circuit error model of [KLM97] but there is also a different adversarial model studied by [GS95], 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. They show how to construct error resilient circuits for symmetric functions in this model, by exploiting interesting connections between their model and probabilistically checkable proofs. However, the obtained circuit is only guaranteed to compute, what they call, a "loose version" of the function, and may err on many inputs. Lastly, we mention the orthogonal direction of constructing testable circuits which are circuits on which errors can detected (but not necessarily corrected) efficiently [BCD+23b, BCD+23a].

Codes for interactive communication. As our work develops error-resilient circuits via error-resilient DAG-protocols, it is also connected with the vast literature on codes for interactive communication, or simply "interactive codes", that are used to make communication protocols resilient to noise. This field, initiated by the seminal works [Sch92, Sch93, Sch96] has received a lot of attention over the last three decades, and various aspects of interactive codes have been extensively studied. A work loosely related to circuits is [CLPS20] but see [Gel17] for an excellent survey. We note that all constructions of error resilient DAG-protocols are heavily inspired by an underlying interactive coding scheme (recall that DAG-protocols are a generalization of communication protocols).

DAG-protocols. Razborov [Raz95] introduced a model of *PLS communication protocols*, and used it generalize the Karchmer-Wigderson transformation [KW88] in a way that preserves both the size and the depth of the circuit. This connection was used by Krajícek [Kra97], 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 [BB94], by reducing to monotone circuit lower bounds. The notion of PLS communication protocols was simplified by Pudlak [Pud10] and Sokolov [Sok17] to the notion of DAG-*like communication protocols*, which we call DAG-protocols. Subsequently, a "converse" to monotone feasible interpolation was established in [GGKS18] to prove new lower bounds on monotone circuits by lifting lower bounds on Resolution refutations. To the best of our knowledge, our work is the only work, other than [EHK+22], to use the notion of DAG-protocols in a "positive" manner, namely to construct error-resilient circuits, in contrast to prior work which mainly used this notion to prove lower bounds in proof complexity and circuit complexity.

1.3 Additional Discussion

An obvious problem left open by our work is whether general circuits can be made error resilient with only a polynomial overhead. As mentioned above, the only work in this regard is [EHK+22], that showed that general circuits can be made error-resilient by incurring a quasi-polynomial overhead. Specifically, if the original circuit has size s and depth d, they construct a resilient version of this circuit that has size $s^{\mathcal{O}(\log d)}$ and depth $\mathcal{O}(d)$. [EHK+22] further say that the power of $\mathcal{O}(\log d)$ seems inherent.

The current work manages to get around the $\mathcal{O}(\log d)$ in the exponent, at the cost of increasing depth to be a polynomial (note that $d = (\log s)^{\mathcal{O}(1)}$ in our setting). It is interesting to see if our techniques can be combined with [EHK+22] to shave off a small factor off of this exponent for general circuits, at the expense of a larger depth. We leave this investigation to future work. Another great question for future research is whether the polynomial blowup in depth that we suffer is inherent, or is there a way to make circuits in NC error resilient while preserving their depth up to a constant factor.

Another interesting direction for future work is whether the resilient circuit can be computed efficiently, say in time polynomial in the size of the circuit. The initial work of [KLR12] guaranteed efficiency when the initial circuit is a formula although the later work [EHK+22] was not able to generalize it to circuits[3]. The current work for circuits in NC also leaves this

[3]Note that the error-resilient circuit of [EHK+22] is of quasi-polynomial size, and thus polynomial in this size of this circuit means that it is quasi-polynomial in the size of the original circuit.

direction open. Finally, even though we get resilience to some constant fraction of adversarial errors, we make no efforts to optimize this constant. Finding the right constant is a great question for future work (see [BEGY19]).

2 Technical Overview

We now overview the ideas behind Theorem 1.1 and contrast it with prior work [KLR12, BEGY19, EHK+22]. Roughly speaking, Theorem 1.1 says that there exist polynomial sized resilient circuits for every circuit C in NC. We let s and d be the size and the depth of C, respectively, and use f to denote the Boolean function computed by C. We will also assume without loss of generality that the circuit C is layered and alternating. Just like prior work, our construction is in three-steps via DAG-protocols[4]:

1. **Circuits → DAG-protocols.** This step is the same as prior work, and uses the Karchmer-Wigderson transformation [KW88] to transform the circuit that computes f to a DAG-protocol that solves the Karchmer-Wigderson game KW_f associated with f with rectangular correctness. This transformation satisfies the property that short-circuit errors in the original circuit correspond to running the protocol over a channel with corruption noise and perfect feedback. We will call such channels feedback channels for simplicity and use this property in the Step 3 below.

2. **DAG-protocols → error-resilient DAG-protocols.** This is the main step in the proof where the input is a DAG-protocol and the goal is to output a functionally equivalent DAG-protocol that works on feedback channels. This is the main contribution of this work and is detailed below.

3. **Error-resilient DAG-protocols → error-resilient circuits.** Having constructed a protocol that solves KW_f with rectangular correctness even on feedback channels, we use the "inverse" Karchmer-Wigderson transformation to transform it to a Boolean circuit computing f that is resilient to short-circuit errors. This step in our proof can also be found in prior work.

The main contribution of this work is Step 2 above, which transforms DAG-protocols to error-resilient DAG-protocols. Analogous transformations in prior work were usually done using interactive coding schemes, an area of research initiated by the seminal works [Sch92, Sch93, Sch96] that is dedicated to making communication protocols resilient to errors. At an extremely high level, this is done by designing mechanisms to detect errors fast and then using rewind mechanisms to rewind to a point in the communication history that was before these errors occurred.

[4]Readers not familiar with DAG-protocols may find it useful to first go over Section 2.1 of [EHK+22].

However, to rewind to a point in the communication history before the errors occurred, the resilient protocol needs to remember the state at that point. After the Karchmer-Wigderson transformation, this extra memory shows up as a blowup in the size of the resilient circuit, which we want to minimize. To be more specific, prior work on resilient circuits [EHK+22], remembers $\mathcal{O}(\log d)$ different states at every point in the resilient protocol which means that each layer in the resilient circuit is quasi-polynomially larger (*i.e.*, has size $s^{\mathcal{O}(\log d)}$) than each layer in the original circuit. Thus, even though [EHK+22] increase the number of layers by only a constant factor, the blowup in the size of each layer makes the overall blowup quasi-polynomial.

Our approach is to greatly reduce the size of every layer, at the expense of the increasing the total number of layers. This means that we cannot even afford to remember $\mathcal{O}(\log d)$ many states at any point. As any interactive coding scheme we are aware of remembers at least logarithmically many states, they would not suit us, and new ideas are needed.

2.1 Our Approach For NC^1

We start by considering a restricted case where the original circuit C lies in NC^1. Even though circuits in NC^1 can be transformed into formulas with only a polynomial blowup, we will stick with the circuit view as it will allow us to generalize later on. Note that prior work already gives a resilient circuit for this case that has size polynomial in s and has depth $\mathcal{O}(d)$. We construct a different resilient circuit that has the same size guarantee but relaxes the depth guarantee[5]. This alternate construction allows us to extend to all of NC.

Our approach. We devise a way to make DAG-protocols corresponding to NC^1 circuits resilient while remembering only a constant number (in fact, 2) states at each point, at the cost of blowing up the number of layers exponentially. As the depth of NC^1 circuits is logarithmic in n, this exponential blowup is still polynomial in n. Moreover, as we only remember a constant number of states at each point, the blowup in the size of each layer is also polynomial, and combining this with the blowup in the number of layers still gives a polynomial blowup.

Note that any protocol must at least remember the current state, and thus a protocol that remembers only 2 states actually remembers only 1 extra state! Interestingly, for our protocol, this extra state is not a state in the past but actually the next state in the future that the protocol is considering going to. Specifically, if the protocol is currently in state in layer $i - 1$ (for some $i > 0$), and wants to advance to a state in layer i, then instead of jumping to that state directly, it is remembered as a "candidate" state until the protocol has enough "confidence" in the candidate to advance. This confidence is implemented via a variable `cnf` that is incremented every time the parties want to advance and decremented every time they suspect that the states in memory are incorrect. Only after the confidence `cnf` hits a certain threshold $\mathfrak{C}_i$ does the protocol actually advance to the state in layer i.

[5]The depth is at most the size, so it must still be polynomial in s.

We will view the thresholds as being cumulative, *i.e.*, $\mathfrak{C}_i$ is the total confidence required to go from layer 0 to layer i. This means that $\mathfrak{C}_i - \mathfrak{C}_{i-1}$ is the additional confidence needed to advance from layer $i-1$ to layer i. With this said, the obvious question is how large does $\mathfrak{C}_i - \mathfrak{C}_{i-1}$ need to be? In other words, how much confidence should the protocol have in the candidate before they advance to a state in layer i? To answer this question, one needs to consider the total "loss" of the protocol in case it advances wrongly and reaches an incorrect state. As no state in the past is remembered, the only way the protocol can recover from this incorrect state is if it starts all over again and reaches the state in layer $i-1$, which requires a total increase in confidence of $\mathfrak{C}_{i-1}$. Thus, the increase (which equals $\mathfrak{C}_i - \mathfrak{C}_{i-1}$) of the confidence required to jump from layer $i-1$ to layer i should be the same order of magnitude as the confidence $\mathfrak{C}_{i-1}$ needed to reach layer $i-1$ from the beginning of the protocol. Solving, we get that $\mathfrak{C}_i$ should be (at least) exponential in i. This is the choice that we make, setting $\mathfrak{C}_i = c^i$ for some suitable constant $c > 1$.

Protocol details. We now provide more details of our protocol and argue its correctness. The protocol starts at the initial (root) node in layer 0 with confidence $\texttt{cnf} = \mathfrak{C}_0 = 0$. At any given time, the protocol is in some layer, say layer $i-1$ with some confidence $\texttt{cnf}$, and trying to continue the simulation from some state in this layer. For this, the protocol first checks[6] if the current state has any errors. If so, the protocol decreases the confidence $\texttt{cnf}$ by 1. This may cause $\texttt{cnf}$ to hit 0, which is taken as a signal to forget all progress and restart the simulation. If no errors are found, then the current state is assumed to be correct and the action to be taken is determined by the value of the confidence $\texttt{cnf}$.

If we have $\texttt{cnf} < \mathfrak{C}_{i-1}$, then it must be the case that the confidence was decreased after reaching layer $i-1$. The protocol simply increases it by 1 trying to get it back to $\texttt{cnf}_{i-1}$ so that further progress can be made. If the confidence $\texttt{cnf} = \mathfrak{C}_{i-1}$, the confidence in layer $i-1$ is high enough for the protocol to compute a candidate in layer i. A candidate $\texttt{nxt}$ for the state to advance to is computed and the confidence is further increased by 1.

Finally, if the confidence $\texttt{cnf} > \mathfrak{C}_{i-1}$, then the protocol must already have a candidate $\texttt{nxt}$ that it is considering going to. The protocol then checks if this candidate was caused due to errors, and if so, decreases the confidence by 1. If the confidence is lowered all the way back to $\mathfrak{C}_{i-1}$, the protocol "forgets" about the candidate $\texttt{nxt}$ (and will compute a new one in the future if needed). On the other hand, if the candidate $\texttt{nxt}$ was not due to errors, the protocol increases the confidence $\texttt{cnf}$ by 1. If this increased confidence reaches $\texttt{cnf}_i$, it is high enough to advance. The protocol forgets about the current state in layer $i-1$ and continue the rest of the simulation from the state $\texttt{nxt}$ in layer i.

[6] As in prior work, assuming that the original DAG-protocol is rectangular correct is crucial for the protocol to be able to check whether or not there are errors. As this is not a novel idea for the current work, we omit a precise description in this sketch. (Very) roughly speaking (see formal definition in Definition 3.3), a state is said to be rectangular correct for inputs x and y for Alice and Bob respectively, if there exists inputs x' for Alice and y' for Bob such that the protocol reaches that state when executed with inputs x and y' and also reaches it with inputs x' and y. This means that the parties can go over all possible inputs of the other party in order to check if the state is (rectangular) correct or not.

Arguing correctness. We now show that the above protocol indeed simulates the original protocol correctly. For this, we first show that if the protocol has high confidence at the end of the protocol, specifically, if $\texttt{cnf} \geq \mathfrak{C}_d$, where d is the total number of layers (depth) of the original protocol, then the output of the protocol must be correct. Indeed, if the confidence is that high, the protocol is in a state in layer d or higher[7], and the only reason this state would not be correct is if the protocol advanced to it incorrectly. However, this means that there must have been at least $\mathfrak{C}_d - \mathfrak{C}_{d-1}$ errors while this state was the next candidate. As the confidence thresholds are exponential, we have that the total number of errors is at least $\mathfrak{C}_d - \mathfrak{C}_{d-1} = \Omega(\mathfrak{C}_d)$, which is more than a constant fraction and therefore, unaffordable.

Now, all we need to show is that the confidence is indeed high at the end of the protocol. For this, we fix any execution of the protocol and partition the rounds into rounds where the protocol wants to "go back" and decrease the confidence $\texttt{cnf}$ (this happens when either the current state or the next state $\texttt{nxt}$ has any errors) and rounds where it does not. We collect the former type of rounds in the set Back. By definition, the confidence will increase in any round not in Back unless there are errors, and will decrease in any round in Back unless there are errors. As the total number of errors is only a small constant fraction, this means that all we need to do to show that the confidence is high at the end of the protocol is to show that the size of Back is small.

To show this, we argue that the size of Back is at most a constant factor more than the number of rounds in Back where the protocol did not go back. To see why, note that the first round in Back must be a round where the protocol added an incorrect next state as their candidate $\texttt{nxt}$. As mentioned above, if the next state is in layer i, this only happens when the confidence is exactly $\mathfrak{C}_{i-1}$.

As long as the protocol does not advance to this next state (in other words, it remains a candidate), the confidence must stay above $\mathfrak{C}_{i-1}$, as otherwise the protocol will forget the incorrect candidate and look for a new one. This means that the total number of rounds where the protocol does not go back and increase the confidence exceeds the total number of rounds where the protocol did go back, and we are done. On the other hand, if the protocol does advance to the candidate, there must have been at least $\mathfrak{C}_i - \mathfrak{C}_{i-1}$ errors, and by definition of $\mathfrak{C}$, these are high enough (up to constant factors) for $\mathfrak{C}_i$ rounds of progress. Thus, the protocol can wait until the confidence hits 0 and restart the simulation from there.

2.2 From NC^1 to NC

We now show how we can extend the ideas above to get a simulation for all of NC. Note that the Karchmer-Wigderson transformation transforms general NC circuits to DAG-protocols of poly-logarithmic depth. As our arguments above increase the depth exponentially, using them on general NC circuits would make the depth super-polynomial.

To reduce the depth, we extend our protocol above to also remember some states in the past. Specifically, if the current state is layer $i - 1$, the protocol also remembers the state it

[7]We append the original protocol with dummy states to get states beyond layer d.

encountered in the most recent layer that was a multiple of $(\log s)^j$, for all $j \in \mathbb{N}$. As the total number of layers we have is poly-logarithmic, we get that when j is larger than some fixed constant, the most recent layer that is a multiple of $(\log s)^j$ is just layer 0. This means that the parties only remember a constant number of distinct states, implying as above that the blowup in the size of each layer is still polynomial.

We now argue why remembering these extra states ("meeting points") in the past will also reduce the total number of layers back to polynomial. The reason is that if the protocol remembers meeting point in the near future, then, when the confidence is too low, the protocol does not have to forget all progress and restart the simulation from scratch. Instead, the protocol only has to forget the progress since the last meeting point and restart the simulation from said meeting point. Because of the loss in progress is now smaller, we can allow the confidence increase (which equals $\mathfrak{C}_i - \mathfrak{C}_{i-1}$) required to advance to also be smaller. As this happens for almost all i, the value of $\mathfrak{C}_d$, the highest confidence threshold is significantly smaller, and we can bound it by a polynomial.

The new confidence thresholds. We are yet to specify the precise increase in confidence (which equals $\mathfrak{C}_i - \mathfrak{C}_{i-1}$) required to advance to a state in layer i. For this, we look at the representation of i as an integer[8] in base $\log s$. If $a_1, a_2, \ldots$ are the digits in this representation, we set $\mathfrak{C}_i - \mathfrak{C}_{i-1}$ to be proportional to $c^{\sum_i a_i}$. As there is a meeting point in memory for every power of $\log s$, it can be seen that $c^{\sum_i a_i}$ takes into account the distance from all the meeting points stored in memory, while maintaining the exponential growth required between two meeting points. We defer the remaining details to the proof.

Getting rectangular correctness. Just like prior work, for Step 2 above to work, it is crucial that our simulation protocol is rectangular correct, which is a stronger notion of correctness than standard correctness, and is necessary for Step 3 to work. Without going into the precise definition, this means that we need to account for the errors in the rounds where the parties (Alice and Bob) are talking separately. That is, the errors in rounds where Alice is talking cannot be used to offset the rounds where Bob is going back and decreasing the confidence, and vice versa.

Other than doubling the notation we use (*e.g.*, we now need to remember a pair of meeting points, one for Alice and one for Bob, *etc.*), this also introduces a more subtle challenge regarding the fraction of errors that we can be resilient to. To understand this, recall that the layers in the original circuit, and therefore the layers in the original DAG-protocol, are alternating. This means that the party controlling the layer changes every time the protocol advances a layer. Now, imagine what happens when the adversary spends $\mathfrak{C}_i - \mathfrak{C}_{i-1}$ errors in a layer controlled by Alice to advance it incorrectly to a layer controlled by Bob.

As the errors for the both the parties are accounted for separately, Bob cannot detect and correct these errors, and the protocol will run as if there were no errors till the next time

[8]The actual proof uses the representation of $i - 1$ for technical reasons.

Alice gets to speak. This happens only after $\mathfrak{C}_{i+1} - \mathfrak{C}_i$ rounds, which due to our definition of $\mathfrak{C}$, may be a factor of c larger than $\mathfrak{C}_i - \mathfrak{C}_{i-1}$, the total number of errors inserted by the adversary. Thus, it is possible for the adversary to waste c rounds of simulation per error, implying that there has to be a multiplicative $1/c$ loss in the overall error resilience. Moreover, c cannot be too small, as then the parties will not be able to correctly maintain their meeting points. We are able to show that this loss is a constant proportional to the total number of meeting points; see the formal version of our main theorem (Theorem 3.6).

3 Model and Preliminaries

We now describe the model and our result more formally. Note that our description closely follows [EHK+22].

3.1 Circuits and DAG-protocols

Circuits. We will consider Boolean circuits with negations at the inputs. That is, a Boolean circuit will be a directed acyclic graph C where each non-source node (or gate) is labelled as an $\wedge$ gate or an $\vee$ gate, while each source node is labelled with an input variable or the negation of the input variable. The source nodes are also called input gates and one designated node in C is called the output gate. We use $V_\wedge$ and $V_\vee$ to denote the set of all the nodes in C that are labelled $\wedge$ and $\vee$ respectively. We define how computation over a circuit C takes place by inductively defining the functions $f_{C,v}$ computed at each gate v. We have:

$$f_{C,v} = \begin{cases} z, & \text{if } v \text{ is an input gate with label } z \\ \bigvee_{u:(u,v) \text{ is an edge in } C} f_{C,u}, & \text{if } v \in V_\vee \\ \bigwedge_{u:(u,v) \text{ is an edge in } C} f_{C,u}, & \text{if } v \in V_\wedge \end{cases} . \tag{1}$$

When v is the output gate, we omit it from the notation and simply write f_C, which we call the function computed by C. We will assume that every gate in C that is not an input gate has at least one edge coming to it. We define the size of C to be the number of nodes in C and the depth of C to be the length of the longest path in C starting from an input gate and ending at the output gate. We use $\|C\|$ to denote the size of C.

Note that edges in C are going "up" from the inputs to the output gate. This contrasts with the edges in DAG-protocols that are going "down" from the root to the leaves as defined below. These two directions, although different, follow the conventions for both circuits and DAG-protocols.

Search problems and KW-games. Let $\mathcal{X}$ and $\mathcal{Y}$ be input sets for Alice and Bob respectively and $\mathcal{O}$ be a set of outputs. A search problem on these input and output sets is defined by a relation $S \subseteq \mathcal{X} \times \mathcal{Y} \times \mathcal{O}$, and the goal of Alice and Bob is to determine, given inputs $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ respectively, an element $o \in \mathcal{O}$ such that $(x, y, o) \in S$.

KW-games are special type of search problems that are defined using a Boolean function f on n variables (for some n). Here, the set $\mathcal{X}$ is the set $f^{-1}(1)$ of all inputs for which f evaluates to 1, $\mathcal{Y}$ is the set $f^{-1}(0)$ of all inputs for which f evaluates to 0, and $\mathcal{O}$ is the set $[n]$. As $\mathcal{X}$ and $\mathcal{Y}$ are disjoint by definition, for any $x \in \mathcal{X}$, $y \in \mathcal{Y}$, there exists $o \in [n]$ such that $x_o \neq y_o$. The search problem KW_f is the problem of finding such an o, namely, it is the subset:

$$\mathsf{KW}_f = \{(x, y, o) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{O} \mid x_o \neq y_o\}.$$

DAG-protocols. We now define the notion of DAG-protocols. Let $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{O}$ be input and output sets as above. A DAG-protocol is defined by a tuple:

$$\Pi = \big(G = (V_A \cup V_B \cup V_O, E), \mathsf{rt}, \{h_v\}_{v \in V_A \cup V_B}, \{o_v\}_{v \in V_O}\big).$$

Here, G is a directed acyclic graph with vertices partitioned into V_A, V_B, and V_O respectively, and edges E. We will use $V = V_A \cup V_B \cup V_O$ to be the set of all vertices in G and $\mathsf{rt} \in V$ is a special vertex called the root. For all $v \in V_A$, the function h_v is a "message function" that maps Alice's input set $\mathcal{X}$ to a vertex in V that v has an edge to. Similarly, for all $v \in V_B$, the function h_v maps Bob's input set $\mathcal{Y}$ to a vertex in V that v has an edge to. We often conflate the two and write $h_v : \mathcal{X} \times \mathcal{Y} \to V$ with the understanding that the second argument is redundant if $v \in V_A$ and the first argument is redundant if $v \in V_B$. Finally, for all $v \in V_O$, the value $o_v \in \mathcal{O}$ is the output value for the vertex v. We will assume that vertices in V_O do not have any out-edges, all other vertices have at least one outgoing edge (as otherwise the message functions will not be defined) and rt does not have any incoming edges. We define the size of Π to be $|V|$, the number of nodes in Π, and the depth of Π to be the length of the longest path in G starting from rt. We use $\|\Pi\|$ to denote the size of Π.

As mentioned above, edges in a DAG-protocol go "down" from the root to the leaves, and are thus opposite to our convention for circuits. We are explicit about the direction in our exposition whenever there is room for ambiguity.

Execution of a DAG-protocol. A DAG-protocol Π as above is executed by start with inputs $x \in \mathcal{X}$, $y \in \mathcal{Y}$ and a vertex $v = \mathsf{rt}$, and continuing as follows: If $v \in V_A \cup V_B$ then the execution simply uses the message function and updates the current vertex to $h_v(x, y)$[9]. Otherwise, we have $v \in V_O$, and the execution terminates with the output o_v. As this output is determined by x and y, we will denote it using $\Pi(x, y)$. For a search problem $S \subseteq \mathcal{X} \times \mathcal{Y} \times \mathcal{O}$, we say that Π solves S if for all $x \in \mathcal{X}$, $y \in \mathcal{Y}$, we have $(x, y, \Pi(x, y)) \in S$.

[9]Recall that $h_v(x, y)$ is independent of y if $v \in V_A$ and is independent of x if $v \in V_B$.

3.2 Rectangular Correctness & Equivalence of Circuits and DAG-protocols

The models of Boolean circuits and DAG-protocols described above have a deep connection. To understand this connection, we first define the notion of rectangular correctness for DAG-protocols.

Definition 3.1 (Rectangles for DAG-protocols)**.** *Let Π be a DAG-protocol with input sets $\mathcal{X}$, $\mathcal{Y}$, and output set $\mathcal{O}$. We inductively define the (combinatorial) rectangle associated with each vertex. For the root rt, define the rectangle to $R_{\mathsf{rt}} = \mathcal{X} \times \mathcal{Y}$. For a node $v \neq \mathsf{rt}$, define the (combinatorial) rectangle $R_v \subseteq \mathcal{X} \times \mathcal{Y}$ to be the smallest rectangle containing $\bigcup_{u:(u,v)\in E}\{(x,y) \in R_u \mid h_u(x,y) = v\}$.*

For a combinatorial rectangle $R \subseteq \mathcal{X} \times \mathcal{Y}$, we will use R_A to denote the projection of the rectangle on the set $\mathcal{X}$ and R_B to denote the projection on the set $\mathcal{Y}$. As we defined R_v to be the smallest rectangle in Definition 3.1, we have the following observation.

Observation 3.2. *Let Π be a DAG-protocol with input sets $\mathcal{X}$, $\mathcal{Y}$, and output set $\mathcal{O}$ and $\{R_v\}_{v\in V}$ be the associated rectangles. For all $v \neq \mathsf{rt}$ and all $x \in (R_v)_A$, there exists a $u \in V$ such that $(u,v) \in E$ and $y \in \mathcal{Y}$ such that $(x,y) \in R_u$ and $h_u(x,y) = v$. An analogous claim holds with the roles of x and y reversed.*

Additionally, observe that for all vertices v, the rectangle R_v contains all the pairs (x,y) such that the execution of Π goes to v on inputs x and y. Thus, the following notion of rectangular correctness is stronger than the "standard" notion of correctness.

Definition 3.3 (Rectangular Correctness)**.** *Let Π be a DAG-protocol with inputs sets $\mathcal{X}$, $\mathcal{Y}$, and output set $\mathcal{O}$, and $\{R_v\}_{v\in V}$ be the associated rectangles. For a search problem $S \subseteq \mathcal{X} \times \mathcal{Y} \times \mathcal{O}$, we say that Π solves S with rectangular correctness if for all $v \in V_O$ and all $(x,y) \in R_v$, we have $(x,y,o_v) \in S$.*

We are now ready to state the equivalence between circuits and DAG-protocols formally. This theorem can also be found in [EHK+22]. We include a proof for completeness.

Theorem 3.4 ([Raz95, Sok17])**.** *Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function.*

1. *For any circuit C that computes f, there is a DAG-protocol Π with the same size and depth as C that solves KW_f with rectangular correctness.*
2. *For any DAG-protocol Π that solves KW_f with rectangular correctness, there is a circuit C with $\|C\| \leq \|\Pi\|$ that computes f.*

Proof. We only prove Item 1 as Item 2 is implied by Theorem 3.5 proved later. Fix a circuit C and define a DAG-protocol Π for KW_f by setting the graph G to be the same graph as C except that the directions of all the edges are reversed. Let V_A be the set of all $\vee$ gates in C, V_B be the set of all the $\wedge$ gates in C and V_O be the set of all the inputs gates. Let

rt be the output gate of C. For all $v \in V_O$, we define o_v to be the variable that (possibly after negation) is input at that gate. It remains to define the message transmission functions $\{h_v\}_{v \in V_A \cup V_B}$.

Fix $v \in V_A \cup V_B$ and recall that the input set $\mathcal{X} = f^{-1}(1)$ and $\mathcal{Y} = f^{-1}(0)$. For $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, we define $h_v(x, y)$ as follows: If $v \in V_A$, then $h_v(x, y)$ is just a function of x. If $f_{C,v}(x) = 1$, then, as v is a $\vee$ gate[10], by Equation (1), there exists a neighbor u (in-neighbor in C and out-neighbor in G) of v such that $f_{C,u}(x) = 1$, and we set $h_v(x, y)$ to be the lexicographically smallest such neighbor u. Otherwise, we set $h_v(x, y)$ to be the lexicographically smallest neighbor u of v. Observe that such a neighbor always exists as v is not an input gate. Similarly, if $v \in V_B$, then $h_v(x, y)$ is just a function of y. If $f_{C,v}(y) = 0$, then, as v is a $\wedge$ gate, by Equation (1), there exists a neighbor u (in-neighbor in C and out-neighbor in G) of v such that $f_{C,u}(y) = 0$, and we set $h_v(x, y)$ to be the lexicographically smallest such neighbor u. Otherwise, we set $h_v(x, y)$ to be the lexicographically smallest neighbor u of v. As before, such a neighbor always exists as v is not an input gate.

We clearly have $\|\Pi\| = \|C\|$ and we just have to show that Π solves KW_f with rectangular correctness. By Definition 3.3, we have to show that, if $\{R_v\}_{v \in V}$ are the rectangles associated with Π as in Definition 3.1, then, for all $v \in V_O$ and all $(x, y) \in R_v$, we have $(x, y, o_v) \in \mathsf{KW}_f$. In fact, we will show the stronger statement that for all $v \in V$ and all $(x, y) \in R_v$, we have:

$$f_{C,v}(x) = 1 \qquad \text{and} \qquad f_{C,v}(y) = 0.$$

This is indeed stronger, as if $v \in V_O$, then Equation (1) says that $f_{C,v}$ is just the (possibly negated) input variable o_v implying that $(x, y, o_v) \in \mathsf{KW}_f$. We show this by induction starting from the root rt downwards to the nodes reachable from it[11]. For the root rt, we use the fact that it corresponds to the output gate of C implying that $f_{C,\mathsf{rt}} = f$. The result follows as $\mathcal{X} = f^{-1}(1)$ and $\mathcal{Y} = f^{-1}(0)$.

We now show it for $v \neq \mathsf{rt}$ assuming it holds for all u that have edges to v in Π (equivalently, all u that v has an edge to in the circuit C). Let $(x, y) \in R_v$ be arbitrary. We only show that $f_{C,v}(x) = 1$ as the proof that $f_{C,v}(y) = 0$ is analogous. As $(x, y) \in R_v$, we have by Observation 3.2 that there exists u that has an edge to v in Π and a $y' \in \mathcal{Y}$ such that $(x, y') \in R_u$ and $h_u(x, y') = v$. Applying the induction hypothesis on u, it follows that $f_{C,u}(x) = 1$. Now, if $u \in V_\wedge$, then the fact that v has an edge to u in C together with Equation (1) implies that $f_{C,v}(x) = 1$, as desired. On the other hand, if $u \in V_\vee$, then, use the fact that $h_u(x, y') = v$ to get that $f_{C,v}(x) = 1$, finishing the proof. $\square$

[10] We abuse notation and use v to denote both the vertices in G and the corresponding vertices in C.

[11] Observe that if v is not reachable from rt, then Definition 3.1 says that $R_v = \emptyset$ and there is nothing to show.

3.3 Error Models

Circuits. We consider short-circuit errors. Let C be a Boolean circuit as above. An error pattern for C is defined by a function $e : V \to V \cup \{*\}$ satisfying the property that $e(v)$ maps v to an in-neighbor of v, *i.e.* to a neighbor of v that is "below" it, or to $*$, for all $v \in V$. In particular, if $v \notin V_\wedge \cup V_\vee$, we have $e(v) = *$. It will often be convenient to separate e into two functions and write $e = (e_\vee, e_\wedge)$, where $e_\vee$ is the function e restricted to the vertices in $V_\vee$ and $e_\wedge$ is the function e restricted to the vertices in $V_\wedge$. Let C be a circuit and e be an error pattern for C. Let $v \in C$ be a gate. We define how computation takes place in C in the presence of errors e by inductively defining the functions $f_{C,e,v}$ computed at each gate v. We have[12]:

$$f_{C,e,v} = \begin{cases} z, & \text{if } v \text{ is an input gate with label } z \\ \bigvee_{u:(u,v) \text{ is an edge in } C} f_{C,e,u}, & \text{if } v \in V_\vee \text{ and } e(v) = * \\ \bigwedge_{u:(u,v) \text{ is an edge in } C} f_{C,e,u}, & \text{if } v \in V_\wedge \text{ and } e(v) = * \\ f_{C,e,e(v)}, & \text{if } e(v) \neq * \end{cases} . \qquad (2)$$

We omit v from the notation if v is the output gate of C. Let C be a circuit and $\mathcal{E}$ be a set of error patterns for C. For a function f, we say that C computes f despite $\mathcal{E}$, if for all $e \in \mathcal{E}$, we have $f_{C,e} = f$. We say that $\mathcal{E}$ is rectangular if viewing every $e \in \mathcal{E}$ as a pair $e = (e_\vee, e_\wedge)$ gives a combinatorial rectangle. When this happens, we use $\mathcal{E}_\vee$ to denote the projection of this rectangle on the first coordinate and $\mathcal{E}_\wedge$ to denote the projection on the second coordinate (thus, $\mathcal{E} = \mathcal{E}_\vee \times \mathcal{E}_\wedge$)

We now define the set of error patterns that we work with. Let $\Theta \geq 0$ be an integer parameter. For a circuit C, define the set $\mathcal{E}_\Theta(C)$ to be the set of all error patterns e such that for any path in C that starts at an input gate and ends the output gate, at most Θ gates $v \in V_\wedge$ on the path satisfy $e(v) \neq *$ and at most Θ gates $v \in V_\vee$ on the path satisfy $e(v) \neq *$. Observe that the set $\mathcal{E}_\Theta(C)$ is rectangular for all $\Theta \geq 0$.

DAG-protocols. Let Π be a DAG-protocol as above. An error pattern ξ for Π is defined by a function $\xi : V \to V \cup \{*\}$ satisfying the property that $\xi(v)$ maps v to an out-neighbor of v, *i.e.* to a neighbor of v that is "below" it[13], or to $*$. In particular, if $v \in V_O$, we have $\xi(v) = *$. It will often be convenient to separate ξ into two functions $\xi = (\xi_A, \xi_B)$, where ξ_A is the function ξ restricted to the vertices in V_A, and ξ_B is the function ξ restricted to the vertices in V_B. The error pattern ξ affects the execution of Π as follows: If the current vertex is v and $\xi(v) = *$, the execution proceeds as before. On the other hand, if $\xi(v) \neq *$ (observe that this can only happen if $v \in V_A \cup V_B$), the execution updates the current vertex to $\xi(v)$ instead of updating using the function h_v as before.

Let Π be a DAG-protocol and Ξ be a set of error patterns for Π. Similarly to before, we

[12]Recall that the label of an input gate is either an input variable or the negation of an input variable.

[13]Recall that our conventions for the directions of edges in circuits is opposite that in DAG-protocols.

say that Ξ is rectangular if viewing every $\xi \in \Xi$ as a pair $\xi = (\xi_A, \xi_B)$ gives a combinatorial rectangle. When this happens, we use Ξ_A to denote the projection of this rectangle on the first coordinate and Ξ_B to denote the projection on the second coordinate (thus, $\Xi = \Xi_A \times \Xi_B$). For a rectangular Ξ, define the DAG-protocol Π_Ξ to be the same as Π except that the input sets are now $\mathcal{X} \times \Xi_A$ and $\mathcal{Y} \times \Xi_B$ and the message functions $h_{\Xi,v}$ are defined as (recall that $\xi = (\xi_A, \xi_B)$):

$$h_{\Xi,v}((x,\xi_A),(y,\xi_B)) = \begin{cases} h_v(x,y), & \text{if } \xi(v) = * \\ \xi(v), & \text{if } \xi(v) \neq * \end{cases}. \tag{3}$$

For a search problem $S \subseteq \mathcal{X} \times \mathcal{Y} \times \mathcal{O}$ and a rectangular Ξ, we say that Π solves S with rectangular correctness despite Ξ if Π_Ξ solves S_Ξ with rectangular correctness, where $S_\Xi \subseteq (\mathcal{X} \times \Xi_A) \times (\mathcal{Y} \times \Xi_B) \times \mathcal{O}$ is the search problem satisfying $((x,\xi_A),(y,\xi_B),o) \in S_\Xi \iff (x,y,o) \in S$ for all values of x, y, o, ξ_A, ξ_B.

We now define the set of error patterns that we work with. Let $\Theta \geq 0$ be an integer parameter. For a DAG-protocol Π, define the set $\Xi_\Theta(\Pi)$ to be the set of all error patterns ξ such that for any path in G that starts at rt and ends at a node in V_O, at most Θ nodes $v \in V_A$ on the path satisfy $\xi(v) \neq *$ and at most Θ nodes $v \in V_B$ on the path satisfy $\xi(v) \neq *$. Observe that the set $\Xi_\Theta(\Pi)$ is rectangular for all $\Theta \geq 0$. For convenience, we will often abbreviate $\Pi_{\Xi_\Theta(\Pi)}$ to Π_Θ.

3.4 Connecting the Error Models

We now show that error resilient DAG-protocols are stronger than error-resilient circuits. Observe that the theorem below implies Item 2 of Theorem 3.4 as can be seen by setting $\Theta = 0$. Additionally, note that the theorem below differs from the analogous theorem in [EHK+22] in that the number of allowed errors Θ is the same for every path, while in [EHK+22], it was a constant fraction of the length of the path. Having it as a constant fraction creates some minor complications while trimming the "empty edges" (in the first paragraph of the proof), that were overlooked in [EHK+22]. These complications do not affect the rest of the proof of [EHK+22], as [EHK+22] actually works even when the allowed number of errors is the same for every path (and equals a constant fraction of the length of the *longest* path). Nonetheless, we flesh out the details of trimming for our model formally in Appendix A. We also mention that having the same number of errors on all the paths, irrespective of their length, is a stronger error model than having the number of errors on each path be proportional to its length.

Theorem 3.5. *Let $\Theta \geq 0$ be given and $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. For any DAG-protocol Π that solves KW_f with rectangular correctness despite $\Xi_\Theta(\Pi)$, there is a circuit C with size and depth at most that of Π that computes f despite $\mathcal{E}_\Theta(C)$.*

Proof. Fix a protocol Π that solves KW_f with rectangular correctness despite $\Xi_\Theta(\Pi)$. By repeatedly applying Theorem A.6 and removing some edges from Π, we can assume without loss of generality that there are no empty edges in Π, as defined in Definition A.2. Equivalently, if $\{R_{\Theta,v}\}_{v\in V}$ are the rectangles associated with Π_Θ according to Definition 3.1 and $\{h_{\Theta,v}\}_{v\in V_A\cup V_B}$ are the message functions, for all edges $e=(u,v)\in E$ we have:

$$R_{\Theta,u}=\emptyset \quad \text{or} \quad \exists((x,\xi_A),(y,\xi_B))\in R_{\Theta,u}: \qquad h_{\Theta,u}((x,\xi_A),(y,\xi_B))=v. \tag{4}$$

As Π that solves KW_f with rectangular correctness despite $\Xi_\Theta(\Pi)$, we have that Π_Θ solves $\mathsf{KW}_{f,\Xi_\Theta(\Pi)}$ with rectangular correctness. From Definition 3.3, we get that for all $v\in V_O$ and all $((x,\xi_A),(y,\xi_B))\in R_{\Theta,v}$, we have $((x,\xi_A),(y,\xi_B),o_v)\in\mathsf{KW}_{f,\Xi_\Theta(\Pi)}$. Simplifying using the definition of $\mathsf{KW}_{f,\Xi_\Theta(\Pi)}$, we get:

$$\forall v\in V_O,((x,\xi_A),(y,\xi_B))\in R_{\Theta,v}: \qquad x_{o_v}\neq y_{o_v}. \tag{5}$$

We now define the circuit C. The gates in the circuit C are exactly the nodes in V and we abuse notation slightly by using u,v *etc.* to refer to both. For any edge $(u,v)\in E$, we add the flipped edge (v,u) to the circuit C. We define all gates in V_A to be $\vee$ gates and all gates in V_B to be $\wedge$ gates while rt is chosen as the output gate. The remaining gates are in V_O are input gates whose labels are defined next. As $R_{\Theta,v}$ in Equation (5) is a combinatorial rectangle, we get that there exists $b\in\{0,1\}$ such that

$$\forall((x,\xi_A),(y,\xi_B))\in R_{\Theta,v}: \qquad x_{o_v}=b \quad \text{and} \quad y_{o_v}=\overline{b}.$$

If $b=1$, we label the input gate v by the variable o_v unnegated, and we label it by the negated variable $\overline{o_v}$ otherwise. This finishes the description of C. As the claim about the size $\|C\|$ is straightforward, we focus on showing that C computes f despite $\mathcal{E}_\Theta(C)$. As both C and Π have the same graph upto the directions of the edges, observe that any error pattern for C can also be seen as a pattern for Π and that $\mathcal{E}_\Theta(C)=\Xi_\Theta(\Pi)$. Because of this, we interpret any error pattern $\xi=(\xi_A,\xi_B)$ as an error pattern $e=(e_\vee,e_\wedge)$ for C and we ξ and subscripts A and B to refer to both. We also define the error patterns $*_A$ and $*_B$ to be those that map all gates in V_A (equivalently, $V_\vee$) and V_B (equivalently, $V_\wedge$) respectively to $*$. To show that C computes f despite $\mathcal{E}_\Theta(C)$, we show inductively that for all $u\in V$, we have:

$$\forall((x,\xi_A),(y,\xi_B))\in R_{\Theta,u}: \qquad f_{C,(\xi_A,*_B),u}(x)=1 \quad \text{and} \quad f_{C,(*_A,\xi_B),u}(y)=0. \tag{6}$$

This suffices as plugging $v=\mathsf{rt}$ and using the definition of $R_{\Theta,\mathsf{rt}}=(f^{-1}(1)\times(\Xi_\Theta(\Pi))_A)\times(f^{-1}(0)\times(\Xi_\Theta(\Pi))_B)$ from Definition 3.1, we get that:

$$\forall((x,\xi_A),(y,\xi_B))\in R_{\Theta,\mathsf{rt}}: \qquad f_{C,(\xi_A,*_B)}(x)=f(x) \quad \text{and} \quad f_{C,(*_A,\xi_B)}(y)=f(y).$$

As short-circuiting a gate in $V_\wedge$ can never change the output from 1 to 0 and short-circuiting a gate in $V_\vee$ can never change the output from 0 to 1, we get that:

$$\forall((x,\xi_A),(y,\xi_B)) \in R_{\Theta,\mathsf{rt}}: \quad f_{C,\xi}(x) = f(x) \quad \text{and} \quad f_{C,\xi}(y) = f(y).$$

Thus, we get that $f_{C,\xi} = f$ for all $\xi \in \Xi_\Theta(\Pi)$. As $\mathcal{E}_\Theta(C) = \Xi_\Theta(\Pi)$, we get that C computes f despite $\mathcal{E}_\Theta(C)$, as desired. Having shown that Equation (6) is sufficient, we now prove that Equation (6) holds by induction. For the base case, when $u \in V_O$, this follows by our labels and Equations (2) and (5). We show the results for $u \notin V_O$ assuming it holds for all v that have edges to u in C. If $R_{\Theta,u} = \emptyset$, there is nothing to show, so we assume otherwise. As the proof in the other case is similar, assume without loss of generality that $u \in V_A$. From Equation (4), we get that for all v that have edges to u in C, we have that there exists $((x,\xi_A),(y,\xi_B)) \in R_{\Theta,u}$ such that $h_{\Theta,u}((x,\xi_A),(y,\xi_B)) = v$. As $u \in V_A$, the message function is determined by the first argument and we get from Definition 3.1 that:

$$(R_{\Theta,u})_A \subseteq \bigcup_{v:(v,u) \text{ is an edge in } C} (R_{\Theta,v})_A \quad \text{and} \quad (R_{\Theta,u})_B \subseteq \bigcap_{v:(v,u) \text{ is an edge in } C} (R_{\Theta,v})_B. \tag{7}$$

We now show Equation (6). Fix $((x,\xi_A),(y,\xi_B)) \in R_{\Theta,u}$.

- **Showing that $f_{C,(\xi_A,*_B),u}(x) = 1$ when $\xi(u) \neq *$:** By Equation (3), we get that $h_{\Theta,u}((x,\xi_A),(y,\xi_B)) = \xi(u)$ implying from Definition 3.1 that $((x,\xi_A),(y,\xi_B)) \in R_{\Theta,\xi(u)}$. By the induction hypothesis on $\xi(u)$, we get that $f_{C,(\xi_A,*_B),\xi(u)}(x) = 1$. From Equation (2), we get:
$$f_{C,(\xi_A,*_B),u}(x) = f_{C,(\xi_A,*_B),\xi(u)}(x) = 1.$$

- **Showing that $f_{C,(\xi_A,*_B),u}(x) = 1$ when $\xi(u) = *$:** We have from $((x,\xi_A),(y,\xi_B)) \in R_{\Theta,u}$ that $(x,\xi_A) \in (R_{\Theta,u})_A$. From Equation (7), there exists an in-neighbor v' of u in C and $(y_{v'},\xi_{B,v'})$ such that $((x,\xi_A),(y_{v'},\xi_{B,v'})) \in R_{\Theta,v'}$. By the induction hypothesis on v', we have $f_{C,(\xi_A,*_B),v'}(x) = 1$. From Equation (2), we get:
$$f_{C,(\xi_A,*_B),u}(x) = \bigvee_{v:(v,u) \text{ is an edge in } C} f_{C,(\xi_A,*_B),v}(x) = 1.$$

- **Showing that $f_{C,(*_A,\xi_B),u}(y) = 0$:** We have from $((x,\xi_A),(y,\xi_B)) \in R_{\Theta,u}$ that $(y,\xi_B) \in (R_{\Theta,u})_B$. It follows from Equation (7) that for all in-neighbors v of u in C, we have that there exists $(x_v,\xi_{A,v})$ such that $((x_v,\xi_{A,v}),(y,\xi_B)) \in R_{\Theta,v}$. From the induction hypothesis on v, we get that $f_{C,(*_A,\xi_B),v}(y) = 0$ for all such v. From Equation (2), we get:
$$f_{C,(*_A,\xi_B),u}(y) = \bigvee_{v:(v,u) \text{ is an edge in } C} f_{C,(*_A,\xi_B),v}(y) = 0.$$

□

3.5 Our Results

We are now ready to state our main result formally. We show that:

Theorem 3.6 (Formal version of Theorem 1.1)**.** *Let $c > 1$, $n > 0$ be integers and $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. Let C be a Boolean circuit of size $\|C\| = s$ and depth*[14] *$d < (\log n)^{c-1}$ that computes f. There exists $\Theta \le s^{\mathcal{O}(c \log c)}$ and a Boolean circuit C' with size at most $s^{\mathcal{O}(c \log c)}$ and depth $\mathcal{O}(c) \cdot \Theta$ that computes f despite $\mathcal{E}_\Theta(C')$.*

To prove Theorem 3.6, we need the following result about DAG-protocols, which forms the technical core of Theorem 3.6.

Theorem 3.7 (Formal version of Theorem 1.2)**.** *Let S be a search problem and Π be a* DAG*-protocol of size s and depth d that solves S with rectangular correctness. Let $c > 1$ be an integer satisfying*[15] *$d < (\log s)^{c-1}$. There exists $\Theta \le s^{\mathcal{O}(c \log c)}$ (defined in Section 4.1) and a* DAG*-protocol Π' (as defined in Section 4.2) with size $s' = s^{\mathcal{O}(c \log c)}$ and depth $\mathcal{O}(c) \cdot \Theta$ that solves S with rectangular correctness despite $\Xi_\Theta(\Pi')$.*

We prove Theorem 3.7 in Section 4, but use it here to prove Theorem 3.6.

Proof of Theorem 3.6 assuming Theorem 3.7. Fix a circuit C as in the theorem statement. Applying Theorem 3.4, we get that there exists a DAG-protocol Π with size s and depth d that solves KW_f with rectangular correctness. Applying Theorem 3.7, we get that there exists $\Theta \le s^{\mathcal{O}(c \log c)}$ and a DAG-protocol Π' with size $s' = s^{\mathcal{O}(c \log c)}$ and depth $\mathcal{O}(c) \cdot \Theta$ that solves KW_f with rectangular correctness despite $\Xi_\Theta(\Pi')$. Now, applying Theorem 3.5 on Π', we get that there exists a circuit C' with size at most $s^{\mathcal{O}(c \log c)}$ and depth at most $\mathcal{O}(c) \cdot \Theta$ that computes f despite $\mathcal{E}_\Theta(C')$, as desired. □

4 Resilient DAG-protocols for NC

The goal of this section is to show Theorem 3.7. For two integers x and y, we write $x \mid y$ if x divides y and $x \nmid y$ otherwise. Fix S and Π as in the theorem statement for the rest of this section and let

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

as in Section 3.1. Also, let R_v be the rectangles associated with Π as in Definition 3.1. We assume without loss of generality that d and $\log s$ are both even and that $\mathsf{rt} \in V_A$. We first remove from E all edges (u, v) for which there does not exist any $(x, y) \in R_u$ such that $h_u(x, y) = v$. Observe from Definition 3.1 that this preserves the rectangles R_v and therefore, also preserves rectangular correctness. In addition, as we only delete edges and

[14] The "-1" in this expression is for technical convenience.

[15] The "-1" in this expression is for technical convenience.

keep the vertices intact (even if they become isolated), the size s stays the same and the depth can only decrease. Thus, we can work with the same c as before. Rename E to be the set after all such edges have been removed. We have that for all $(u, v) \in E$, there exists $(x, y) \in R_u$ such that $h_u(x, y) = v$. By Definition 3.1 and the fact that h_u only depends on the first (respectively, second) argument if $u \in V_A$ (respectively, $u \in V_B$), this means that, for all $(u, v) \in E$:

$$u \in V_A \implies (R_u)_B \subseteq (R_v)_B \qquad \text{and} \qquad u \in V_B \implies (R_u)_A \subseteq (R_v)_A. \tag{8}$$

Next, we modify Π as follows. For every vertex $v \in V$, let d'_v be the length of the longest path from rt to v. If no such path exists, we set $d'_v = 0$. Note that, as G is a (finite) directed acyclic graph, the length of the longest path, and therefore the value d'_v, is well defined and satisfies $0 \le d'_v \le d$, with the latter inequality being tight only if $v \in V_O$ is a leaf.

We use the values d'_v to partition the vertices V into $2d+1$ "layers" indexed from $[0, 2d]$ as follows: All vertices in V_O go to layer $2d$ (the last layer). A vertex $v \in V_A$ goes to the layer $2d'_v$, which is an even number less than $2d$ as $d'_v = d$ only if $v \in V_O$. Similarly, a vertex $v \in V_B$ goes to the odd layer $2d'_v + 1$. Observe that, with this partition, edges only go from lower layers to higher layers, but may potentially "skip" some layers, *i.e.*, go from layer d_1 to layer $d_2 > d_1 + 1$.

We augment the protocol further by adding some "dummy" layers. For all multiples κ of $(\log s - 2)$ starting from 0, we add two dummy layers (containing no vertices), right before layer κ. After adding these layers, we get that or all multiplies κ of $\log s$ starting from 0, layer κ and layer $\kappa + 1$ are dummy layers. We also add two dummy layers right before the last layers. As these layers contain no vertices, what this does is it lets us pretend that the edges going from layer $\kappa - 1$ to layer κ (for non-zero κ) "skip" two layers. This allows us to modify these edges as described next. As this arrangement of layers does increase d by a small multiplicative factor that is < 5, we also have to increase c by a small additive constant for the bound in Theorem 3.7 to work. This increase gets absorbed in the $\mathcal{O}(\cdot)$ notation and we disregard it henceforth. In fact, by adding even more dummy layers right before the last layer assume that the inequality about c in Theorem 3.7 is satisfied with equality.

Next, we describe how we add some extra vertices to avoid the aforementioned "skipping". Specifically, we ensure that all edges connect vertices in adjacent layers by adding extra vertices to any edge that skips a layer so that we can convert it to a path that intersect all layers. This means that the in-degree and out-degree of these extra vertices is 1. Moreover, if the added vertex is in an even layer, we include it in V_A and if it is in an odd layer, we include it in V_B. Finally, as the out-degree is 1, the message function simply sends all inputs to the (unique) out-neighbor. This ensures the desired property for all layers except for the two (dummy) layers we added before (what was) layer 0. Due to these dummy layers, the 'root' of the graph is now in layer 2 instead of being in layer 0. To fix this, we simply add one extra vertex in both these layers and connect them to each other and also connect the vertex in layer 1 to the (former) root. This makes the vertex in layer 0 the new root.

Observe from Definition 3.1 that adding these extra vertices preserves the rectangles of the existing vertices and also preserves Equation (8). In addition, it preserves the depth d and can only increase the size s. Thus, we can work with the same value of c as before. Henceforth, when we refer to Π, s, d, c, we refer to this modified protocol. The protocol Π satisfies Equation (8) as argued above. For $v \in V$, we let $\mathtt{d}(v)$, denote the layer in $[0, d]$ that contains the vertex v. Equivalently, $\mathtt{d}(v)$ is the length of (any) path from the root $\mathtt{rt}$ to v. Because our edges do not skip any vertices, we have that:

$$\begin{aligned} \forall (u,v) \in E : \quad & \mathtt{d}(v) = \mathtt{d}(u) + 1, \\ \forall v \in V_O : \quad & \mathtt{d}(v) = d = (\log s)^{c-1} \geq 4, \\ \forall v \in V_A \cup V_B : \quad & \mathtt{d}(v) < d = (\log s)^{c-1}, \\ \forall v \in V : \quad & 2 \nmid \mathtt{d}(v) \iff v \in V_B. \end{aligned} \tag{9}$$

Finally, due to our dummy layers, we have that for all $v \in V$ for which $\log s$ divides either $\mathtt{d}(v)$ or $\mathtt{d}(v) - 1$ or for which $\mathtt{d}(v) \in \{d-1, d-2\}$, the out degree of v is 1. That is,

$$|\{v' \mid (v, v') \in E\}| = 1. \tag{10}$$

4.1 Defining Parameters

For integers $z \geq 0$, we define:

$$\begin{aligned} \forall i \geq 0 : \quad & z^{(i)} = (\log s)^i \cdot \left\lfloor \frac{z}{(\log s)^i} \right\rfloor, \\ \forall i > 0 : \quad & \{z\}_i = \frac{1}{(\log s)^{i-1}} \cdot \left(z^{(i-1)} - z^{(i)}\right). \end{aligned} \tag{11}$$

That is, $\{z\}_i$ is the i-th digit in the expansion of z using base $\log s$, where $i = 1$ corresponds to the least significant digit, $i = 2$ corresponds to the next least significant digit, and so on. Thus, $\{z\}_i$ is non-zero for only finitely many values of i. We define the function:

$$\mathfrak{C}(z) = \begin{cases} 0, & \text{if } z = 0 \\ \mathfrak{C}(z-1) + c^{1 + \sum_{i>0} \{z-1\}_i}, & \text{if } z > 0 \end{cases}. \tag{12}$$

The following lemmas are the main properties we need about $\mathfrak{C}$.

Lemma 4.1. *For all integers $z, i \geq 0$, it holds that:*

$$\mathfrak{C}(z) - \mathfrak{C}\left(z^{(i)}\right) = \sum_{j=1}^{i} c^{1 + \sum_{i' > j} \{z\}_{i'}} \cdot \frac{c^{\{z\}_j} - 1}{c - 1} \cdot \left(\frac{c^{\log s} - 1}{c - 1}\right)^{j-1}.$$

Proof. Observe that $z = z^{(0)} \geq z^{(1)} \geq \cdots \geq z^{(i)}$. Using Equation (12), this means that:

$$\mathfrak{C}(z) - \mathfrak{C}\big(z^{(i)}\big) = \sum_{j=1}^{i} \sum_{z' \in \left[z^{(j)}, z^{(j-1)}\right)} (\mathfrak{C}(z'+1) - \mathfrak{C}(z')) = \sum_{j=1}^{i} \sum_{z' \in \left[z^{(j)}, z^{(j-1)}\right)} c^{1+\sum_{i'>0}\{z'\}_{i'}}.$$

Next, observe that for all $j \in [i]$, we have that $z' \in \left[z^{(j)}, z^{(j-1)}\right)$ if and only if $\{z'\}_j < \{z\}_j$ and for all $j' > j$, we have $\{z'\}_{j'} = \{z\}_{j'}$. This means that going over all $z' \in \left[z^{(j)}, z^{(j-1)}\right)$ is the same as going over all $\{z'\}_j \in \left[0, \{z\}_j\right)$ and all $\{z'\}_{j'} \in [0, \log s)$ for all $j' < j$. This gives:

$$\begin{aligned}
\mathfrak{C}(z) - \mathfrak{C}\big(z^{(i)}\big) &= \sum_{j=1}^{i} c^{1+\sum_{i'>j}\{z\}_{i'}} \cdot \sum_{a_j \in \left[0, \{z\}_j\right)} \cdots \sum_{a_1 \in [0, \log s)} c^{\sum_{i'=1}^{j} a_{i'}} \\
&= \sum_{j=1}^{i} c^{1+\sum_{i'>j}\{z\}_{i'}} \cdot \left(\sum_{a_j \in \left[0, \{z\}_j\right)} c^{a_j} \right) \cdot \prod_{i'=1}^{j-1} \left(\sum_{a_{i'} \in [0, \log s)} c^{a_{i'}} \right) \\
&= \sum_{j=1}^{i} c^{1+\sum_{i'>j}\{z\}_{i'}} \cdot \frac{c^{\{z\}_j} - 1}{c-1} \cdot \left(\frac{c^{\log s} - 1}{c-1} \right)^{j-1}.
\end{aligned}$$

□

Corollary 4.2. *For all integers $z \geq 0$, it holds that:*

$$\mathfrak{C}(z) = \sum_{j>0} c^{1+\sum_{i'>j}\{z\}_{i'}} \cdot \frac{c^{\{z\}_j} - 1}{c-1} \cdot \left(\frac{c^{\log s} - 1}{c-1} \right)^{j-1}.$$

Lemma 4.3. *Consider an integer $z \geq 0$ and let $i^* > 0$ be the smallest such that $\{z+1\}_{i^*} \neq 0$. We have:*

$$\mathfrak{C}(z) - \mathfrak{C}\big(z^{(i^*)}\big) \leq c^{1+\sum_{i>0}\{z\}_i} \cdot \left(\frac{c}{c-1} \right)^{i^*} = (\mathfrak{C}(z+1) - \mathfrak{C}(z)) \cdot \left(\frac{c}{c-1} \right)^{i^*}.$$

Proof. The equality follows from Equation (12). We only show the inequality. By Lemma 4.1 with $i = i^*$, we get:

$$\begin{aligned}
\mathfrak{C}(z) - \mathfrak{C}\big(z^{(i^*)}\big) &= \sum_{j=1}^{i^*} c^{1+\sum_{i'>j}\{z\}_{i'}} \cdot \frac{c^{\{z\}_j} - 1}{c-1} \cdot \left(\frac{c^{\log s} - 1}{c-1} \right)^{j-1} \\
&= c^{1+\sum_{i'>i^*}\{z\}_{i'}} \cdot \sum_{j=1}^{i^*} c^{\sum_{i'=j+1}^{i^*}\{z\}_{i'}} \cdot \frac{c^{\{z\}_j} - 1}{c-1} \cdot \left(\frac{c^{\log s} - 1}{c-1} \right)^{j-1}
\end{aligned}$$

$$
\begin{aligned}
&\le c^{1+\sum_{i'>i^*}\{z\}_{i'}} \cdot \sum_{j=1}^{i^*} c^{\sum_{i'=j+1}^{i^*}\{z\}_{i'}} \cdot \frac{c^{\{z\}_j}}{c-1} \cdot \left(\frac{c^{\log s}}{c-1}\right)^{j-1} \\
&\le c^{1+\sum_{i'>i^*}\{z\}_{i'}} \cdot \sum_{j=1}^{i^*} c^{\sum_{i'=1}^{i^*}\{z\}_{i'}} \cdot \frac{1}{c-1} \cdot \left(\frac{c}{c-1}\right)^{j-1} && \text{(Definition of } i^*\text{)} \\
&\le c^{1+\sum_{i'>0}\{z\}_{i'}} \cdot \sum_{j=1}^{i^*} \frac{1}{c-1} \cdot \left(\frac{c}{c-1}\right)^{j-1} \\
&\le c^{1+\sum_{i'>0}\{z\}_{i'}} \cdot \left(\frac{c}{c-1}\right)^{i^*}.
\end{aligned}
$$

□

Corollary 4.4 (Corollary of Lemma 4.3). *We have:*

$$
\mathfrak{C}((\log s)^c) \le c^{1+c\cdot(\log s-1)} \cdot \left(1 + \left(\frac{c}{c-1}\right)^{c+1}\right) \le 30 \cdot s^{c\log c}.
$$

Lemma 4.5. *It holds that:*

$$
\mathfrak{C}\big((\log s)^{c-1}\big) \le 3 \cdot \mathfrak{C}\big((\log s)^{c-1} - 1\big).
$$

Proof. We have

$$
\begin{aligned}
\mathfrak{C}\big((\log s)^{c-1} - 1\big) &= \sum_{0<j<c} c^{1+\sum_{j<i'<c}(\log s-1)} \cdot \frac{c^{\log s-1}-1}{c-1} \cdot \left(\frac{c^{\log s}-1}{c-1}\right)^{j-1} \\
&\ge \sum_{0<j<c} c^{1+\sum_{0<i'<c}(\log s-1)} \cdot \big(1 - c^{1-\log s}\big)^j \cdot \frac{1}{c-1} \cdot \left(\frac{c}{c-1}\right)^{j-1} \\
&\ge c^{\sum_{0<i'<c}(\log s-1)} \cdot \sum_{0<j<c} \left(\frac{c\cdot\big(1-c^{1-\log s}\big)}{c-1}\right)^j \\
&\ge c^{\sum_{0<i'<c}(\log s-1)} \cdot (c-1) && \text{(As } \log s \text{ is even implying } \log s \ge 2\text{)} \\
&\ge \frac{1}{2} \cdot \Big(\mathfrak{C}\big((\log s)^{c-1}\big) - \mathfrak{C}\big((\log s)^{c-1} - 1\big)\Big). && \text{(Equation (12))}
\end{aligned}
$$

Rearranging gives the result. □

Owing to the foregoing corollary we can define:

$$
\begin{aligned}
\nabla &= 4 \cdot \mathfrak{C}\big((\log s)^{c-1} - 1\big), \\
\Theta &= \frac{\nabla}{5c} \cdot \left(\frac{c-1}{c}\right)^c \le \frac{25}{c} \cdot s^{c\log c}.
\end{aligned}
\tag{13}
$$

4.2 The Simulation Protocol

We now define the simulation protocol Π'.

The set of vertices $V' = V'_A \cup V'_B \cup V'_O$. We define the set V' to be:

$$V' = V \times V \times (V_A \times (V_B \cup \{\mathsf{rt}\}))^c \times ([0, \nabla] \times [0, \nabla] \times [0, \nabla]). \tag{14}$$

For a vertex $v' \in V'$, we define the following notation for the different coordinates of v'

$$v' = \Big(\mathtt{v}(v'), \mathtt{nxt}(v'), ((\mathtt{v}_{i,A}(v'), \mathtt{v}_{i,B}(v')))_{i\in[c]}, (\mathtt{cnf}(v'), \mathtt{rew}_A(v'), \mathtt{rew}_B(v'))\Big). \tag{15}$$

That is, the first element is denoted by $\mathtt{v}(v')$, the second is denoted by $\mathtt{nxt}(v')$, and so on. Intuitively, (1) $\mathtt{v}(v')$ denotes the vertex currently being simulated, (2) $\mathtt{nxt}(v')$ denotes the next vertex the simulation is considering going to, with $\mathtt{nxt}(v') = \mathsf{rt}$ if and only if no such vertex is under consideration, (3) $((\mathtt{v}_{i,A}(v'), \mathtt{v}_{i,B}(v')))_{i\in[c]}$ is a tuple of c pairs of vertices stored in memory to which the simulation may rewind to in case it is needed, (4) $\mathtt{cnf}(v')$ denotes the confidence in the current and the next vertex, and (5) $\mathtt{rew}_A(v')$ and $\mathtt{rew}_B(v')$ are the number of rewinds so far due to Alice and Bob respectively. We also define the following shorthand notation: For all $i \in [c]$, we will denote the pair $(\mathtt{v}_{i,A}(v'), \mathtt{v}_{i,B}(v'))$ by $\mathtt{v}_i(v')$. We will let $\mathtt{S}(v')$ denote the set $\mathtt{S}(v') = \{\mathtt{v}(v')\} \cup \{\mathtt{v}_{i,A}(v') \mid i \in [c]\} \cup \{\mathtt{v}_{i,B}(v') \mid i \in [c]\}$. Finally, we let $\mathtt{nums}(v')$ denote the triple $(\mathtt{cnf}(v'), \mathtt{rew}_A(v'), \mathtt{rew}_B(v'))$.

We define V'_A to be the set of $v' \in V'$ for which $\mathtt{v}(v') \in V_A$ and $\max(\mathtt{rew}_A(v'), \mathtt{rew}_B(v')) < \nabla$. We define V'_B analogously and set $V'_O = V' \setminus (V'_A \cup V'_B)$. Note that $v' \in V'_O$ if and only if $\mathtt{v}(v') \in V_O$ or $\max(\mathtt{rew}_A(v'), \mathtt{rew}_B(v')) = \nabla$. Finally, we define the root rt' as:

$$\mathsf{rt}' = \Big(\mathsf{rt}, \mathsf{rt}, ((\mathsf{rt}, \mathsf{rt}))_{i\in[c]}, (0, 0, 0)\Big). \tag{16}$$

The set of edges E'. We now specify the set of edges in the graph. For this, we first define edges from a vertex $v' \in V'_A$. Let $0 \leq i' \leq c$ be the largest such that $(\log s)^{i'}$ divides $\mathtt{d}(\mathtt{v}(v'))$. This is well defined as $i' = 0$ is one such value. Such a vertex has a forward edge for every forward edge $(\mathtt{v}(v'), u)$ from $\mathtt{v}(v')$ (in the original graph G) that goes to the vertex:

$$v'_u = \begin{cases} \Big(\mathtt{v}(v'), u, (\mathtt{v}_i(v'))_{i\in[c]}, \mathtt{nums}'\Big), & \text{if } \mathtt{cnf}(v') = \mathfrak{C}(\mathtt{d}(\mathtt{v}(v'))) \\ \Big(\mathtt{nxt}(v'), \mathsf{rt}, \Big(((\mathtt{v}(v'), \mathtt{nxt}(v')))_{i\in[i']}, (\mathtt{v}_i(v'))_{i\in(i',c]}\Big), \mathtt{nums}'\Big), & \text{if } \mathfrak{C}(\mathtt{d}(\mathtt{v}(v')) + 1) \leq \mathtt{cnf}(v') + 1 \\ \Big(\mathtt{v}(v'), \mathtt{nxt}(v'), (\mathtt{v}_i(v'))_{i\in[c]}, \mathtt{nums}'\Big), & \text{otherwise} \end{cases},$$

where: $\qquad \mathtt{nums}' = (\mathtt{cnf}(v') + 1, \mathtt{rew}_A(v'), \mathtt{rew}_B(v')).$

$$\tag{17}$$

Note that, in the last two cases, the vertex v'_u is the same for all u. The vertex v' also has a rewind edge. Let $i''_1 \in [c]$ be the smallest such that $\mathtt{d}\big(\mathtt{v}_{i''_1,A}(v')\big) < \mathtt{d}(\mathtt{v}(v'))$. If no such i''_1 exists, we set $i''_1 = c$. Define $\mathsf{C}' = \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}_{i''_1,A}(v')\big)\big) + 1$. Also, let $i''_1 \leq i''_2 < c$ be the smallest such that $\mathtt{d}\big(\mathtt{v}_{i''_2+1,A}(v')\big) < \mathtt{d}\big(\mathtt{v}_{i''_1,A}(v')\big)$. If no such i''_2 exists, we set $i''_2 = c$. The rewind edge goes to:

$$v'_r = \begin{cases} \Big(\mathtt{v}_{i''_1,A}(v'), \mathsf{rt}, \Big(\big((\mathtt{v}_{i''_1,A}(v'), \mathsf{rt})\big)_{i \in [i''_2]}, (\mathtt{v}_i(v'))_{i \in (i''_2, c]}\Big), \mathtt{nums}'\Big), & \text{if } \mathtt{cnf}(v') \leq \mathsf{C}' \\ \Big(\mathtt{v}(v'), \mathsf{rt}, (\mathtt{v}_i(v'))_{i \in [c]}, \mathtt{nums}'\Big), & \text{if } \mathsf{C}' < \mathtt{cnf}(v') \leq \mathfrak{C}(\mathtt{d}(\mathtt{v}(v'))) + 1 \\ \Big(\mathtt{v}(v'), \mathtt{nxt}(v'), (\mathtt{v}_i(v'))_{i \in [c]}, \mathtt{nums}'\Big), & \text{otherwise} \end{cases},$$

$$\text{where:} \qquad \mathtt{nums}' = (\max(\mathsf{C}', \mathtt{cnf}(v')) - 1, \mathtt{rew}_A(v') + 1, \mathtt{rew}_B(v')). \tag{18}$$

We now define edges from a vertex $v' \in V'_B$. These definitions are analogous to the above, except for the fact that the roles of Alice and Bob are interchanged. We define a forward edge for every forward edge $(\mathtt{v}(v'), u)$ from $\mathtt{v}(v')$ (in the original graph G). Let $0 \leq i' \leq c$ be the largest such that $(\log s)^{i'}$ divides $\mathtt{d}(\mathtt{v}(v')) + 1$. This is well defined as $i' = 0$ is one such value. The forward edge goes to:

$$v'_u = \begin{cases} \Big(\mathtt{v}(v'), u, (\mathtt{v}_i(v'))_{i \in [c]}, \mathtt{nums}'\Big), & \text{if } \mathtt{cnf}(v') = \mathfrak{C}(\mathtt{d}(\mathtt{v}(v'))) \\ \Big(\mathtt{nxt}(v'), \mathsf{rt}, \Big(((\mathtt{nxt}(v'), \mathsf{rt}))_{i \in [i']}, (\mathtt{v}_i(v'))_{i \in (i', c]}\Big), \mathtt{nums}'\Big), & \text{if } \mathfrak{C}(\mathtt{d}(\mathtt{v}(v')) + 1) \leq \mathtt{cnf}(v') + 1 \\ \Big(\mathtt{v}(v'), \mathtt{nxt}(v'), (\mathtt{v}_i(v'))_{i \in [c]}, \mathtt{nums}'\Big), & \text{otherwise} \end{cases},$$

$$\text{where:} \qquad \mathtt{nums}' = (\mathtt{cnf}(v') + 1, \mathtt{rew}_A(v'), \mathtt{rew}_B(v')). \tag{19}$$

As before, in the last two cases, the vertex v'_u is the same for all u. For the rewind edge, let $i'' \in [c]$ be the smallest such that $\mathtt{d}(\mathtt{v}_{i'',B}(v')) < \mathtt{d}(\mathtt{v}(v'))$. If no such i'' exists, we set $i'' = c$. Define $\mathsf{C}' = \mathfrak{C}(\mathtt{d}(\mathtt{v}_{i'',B}(v'))) + 1$. The rewind edge goes to:

$$v'_r = \begin{cases} \Big(\mathtt{v}_{i'',B}(v'), \mathsf{rt}, \big(\mathtt{v}_{\max(i,i'')}(v')\big)_{i \in [c]}, \mathtt{nums}'\Big), & \text{if } \mathtt{cnf}(v') \leq \mathsf{C}' \\ \Big(\mathtt{v}(v'), \mathsf{rt}, (\mathtt{v}_i(v'))_{i \in [c]}, \mathtt{nums}'\Big), & \text{if } \mathsf{C}' < \mathtt{cnf}(v') \leq \mathfrak{C}(\mathtt{d}(\mathtt{v}(v'))) + 1 \\ \Big(\mathtt{v}(v'), \mathtt{nxt}(v'), (\mathtt{v}_i(v'))_{i \in [c]}, \mathtt{nums}'\Big), & \text{otherwise} \end{cases}, \tag{20}$$

$$\text{where:} \qquad \mathtt{nums}' = (\max(\mathsf{C}', \mathtt{cnf}(v')) - 1, \mathtt{rew}_A(v'), \mathtt{rew}_B(v') + 1).$$

The message functions $\{h'_{v'}\}_{v' \in V'_A \cup V'_B}$. Recall that $h'_{v'} : \mathcal{X} \times \mathcal{Y} \to V'$ is the message function for the vertex v' and is determined by the first (respectively, second) argument if

$v' \in V'_A$ (resp. $v' \in V'_B$). Recall that $\{R_v\}_{v \in V}$ are the rectangles associated with Π as in Definition 3.1, where $R_v = (R_v)_A \times (R_v)_B$. For $v' \in V'_A$, define:

$$h'_{v'}(x, y) = \begin{cases} v'_{h_{\mathtt{v}(v')}(x,y)}, & \text{if } x \in \left(R_{\mathtt{nxt}(v')}\right)_A \cap \bigcap_{v \in \mathtt{S}(v')} (R_v)_A \\ v'_r, & \text{otherwise} \end{cases} \tag{21}$$

Similarly, for $v' \in V'_B$, define:

$$h'_{v'}(x, y) = \begin{cases} v'_{h_{\mathtt{v}(v')}(x,y)}, & \text{if } y \in \left(R_{\mathtt{nxt}(v')}\right)_B \cap \bigcap_{v \in \mathtt{S}(v')} (R_v)_B \\ v'_r, & \text{otherwise} \end{cases} \tag{22}$$

The output values $\{o'_{v'}\}_{v' \in V'_O}$. Recall that we have $v' \in V'_O$ if and only if $\mathtt{v}(v') \in V_O$ or $\max(\mathtt{rew}_A(v'), \mathtt{rew}_B(v')) = \nabla$. If the former holds, we define $o'_{v'} = o_{\mathtt{v}(v')}$ and if not, we define it to be an arbitrary value in the set $\mathcal{O}$.

4.2.1 Some Lemmas

We now list some lemmas about our definitions above. The proofs, though long, are a simple inductive argument starting from rt' and using Equations (17) to (20).

Lemma 4.6. *For all $v' \in V'$ reachable from rt', it holds that:*

1. *For all $i \in [c]$, we have:*

$$\mathtt{d}(\mathtt{v}_{i,A}(v')) = (\log s)^i \cdot \left\lfloor \frac{\mathtt{d}(\mathtt{v}(v'))}{(\log s)^i} \right\rfloor$$

$$\mathtt{d}(\mathtt{v}_{i,B}(v')) = \begin{cases} \mathtt{d}(\mathtt{v}_{i,A}(v')) + 1, & \textit{if } \mathtt{d}(\mathtt{v}_{i,A}(v')) < \mathtt{d}(\mathtt{v}(v')) \\ 0, & \textit{otherwise} \end{cases}.$$

   As a corollary, we get $\mathtt{d}(v) \leq \mathtt{d}(\mathtt{v}(v'))$ for all $v \in \mathtt{S}(v')$ and that $\mathtt{v}_{c,A}(v') = \mathsf{rt} \in \mathtt{S}(v')$ and:

$$0 = \mathtt{d}(\mathtt{v}_{c,A}(v')) \leq \mathtt{d}(\mathtt{v}_{c-1,A}(v')) \leq \cdots \leq \mathtt{d}(\mathtt{v}_{1,A}(v')) \leq \mathtt{d}(\mathtt{v}(v'))$$

   We also get that, if $v' \in V'_B$, then $\mathtt{d}(\mathtt{v}_{i,B}(v')) > 0$ for all $i \in [c]$.

2. *We have $\mathtt{cnf}(v') < \mathfrak{C}(\mathtt{d}(\mathtt{v}(v')) + 1)$. If $\mathtt{v}(v') \in V_B$, we also have that $\mathtt{cnf}(v') \geq \mathfrak{C}(1)$.*

3. *For all $i \in [c]$ satisfying $\mathtt{d}(\mathtt{v}_{i,A}(v')) + \mathbb{1}(\mathtt{v}(v') \in V_B) < \mathtt{d}(\mathtt{v}(v'))$, we have:*

$$\mathfrak{C}(\mathtt{d}(\mathtt{v}_{i,A}(v')) + \mathbb{1}(\mathtt{v}(v') \in V_B)) < \mathtt{cnf}(v').$$

4. *We have:*

$$\mathtt{cnf}(v') \leq \mathfrak{C}(\mathtt{d}(\mathtt{v}(v'))) \iff \mathtt{nxt}(v') = \mathsf{rt} \iff \mathtt{d}(\mathtt{v}(v')) + 1 \neq \mathtt{d}(\mathtt{nxt}(v')).$$

Before we continue, observe that for Equations (17) to (20) to be well defined, we have to show that the value of $\texttt{cnf}$ on the right hand side is at most ∇. Due to Lemma 4.5, this follows if we show that for all $v' \in V'_A \cup V'_B$, we have that $\texttt{cnf}(v')+1, \texttt{rew}_A(v')+1, \texttt{rew}_B(v')+1 \leq \nabla$. The last two follows from the definition of V'_A and V'_B while the first one follows from Item 2 above and Lemma 4.5. We now prove Lemma 4.6.

Proof of Lemma 4.6. Proof by induction on the length ℓ of the shortest path from $\texttt{rt}'$ to v'. For the base case $\ell = 0$, we have $v' = \texttt{rt}'$ and the lemma follows from Equation (16). We now prove the lemma for $\ell > 0$ assuming it holds for smaller values of ℓ. Let $v' \in V'$ be arbitrary such that the shortest path from $\texttt{rt}'$ to v' has length ℓ. Let $u' \in V'$ be the vertex immediately before v' on (an arbitrary such path) so that $(u', v') \in E'$. By the induction hypothesis, we have that Lemma 4.6 applies to u'. Consider the following cases:

When $u' \in V'_A$ and $v' \neq u'_r$. In this case, Equation (17) applies. For Item 1, note that it is direct from the induction hypothesis unless we are in Case 2 of Equation (17). If we are in Case 2 of Equation (17), we have $\texttt{d}(\texttt{v}(v')) = \texttt{d}(\texttt{nxt}(u')) = \texttt{d}(\texttt{v}(u')) + 1$ by Item 4 of the induction hypothesis on u'. As $u' \in V'_A$ implying that $\texttt{v}(u') \in V_A$, we get from Equation (9) that $\texttt{d}(\texttt{v}(v')) = \texttt{d}(\texttt{v}(u')) + 1$ is odd. As $\log s$ is even, this means that for all $i \in [c]$, we have that $(\log s)^i \nmid \texttt{d}(\texttt{v}(u')) + 1$. Letting i' be as defined before Equation (17), this gives, for all $i \in [c]$:

$$\begin{aligned}
(\log s)^i \cdot \left\lfloor \frac{\texttt{d}(\texttt{v}(v'))}{(\log s)^i} \right\rfloor &= (\log s)^i \cdot \left\lfloor \frac{\texttt{d}(\texttt{v}(u'))}{(\log s)^i} \right\rfloor \\
&= \texttt{d}(\texttt{v}_{i,A}(u')) && \text{(Induction hypothesis)} \\
&= \begin{cases} \texttt{d}(\texttt{v}(u')), & \text{if } i \in [i'] \\ \texttt{d}(\texttt{v}_{i,A}(u')), & \text{if } i \in (i', c] \end{cases} && \text{(Induction hypothesis)} \\
&= \texttt{d}(\texttt{v}_{i,A}(v')). && \text{(Equation (17))}
\end{aligned}$$

In addition, from the penultimate step and Item 1 of the induction hypothesis, it follows that $\texttt{d}(\texttt{v}_{i,A}(v')) \leq \texttt{d}(\texttt{v}(u')) < \texttt{d}(\texttt{v}(v'))$. Thus, to finish our proof of Item 1, we have to show that for all $i \in [c]$, it holds that $\texttt{d}(\texttt{v}_{i,B}(v')) = \texttt{d}(\texttt{v}_{i,A}(v')) + 1$. For $i \in [i']$, this is because $\texttt{d}(\texttt{nxt}(u')) = \texttt{d}(\texttt{v}(u')) + 1$. For $i \in (i', c]$, we use the definition of i' to get that $(\log s)^i \nmid \texttt{d}(\texttt{v}(u'))$. By Item 1 of the induction hypothesis, we get that $\texttt{d}(\texttt{v}_{i,A}(u')) < \texttt{d}(\texttt{v}(u'))$ implying that $\texttt{d}(\texttt{v}_{i,B}(u')) = \texttt{d}(\texttt{v}_{i,A}(u')) + 1$ and the result follows from Equation (17). This finishes the proof of Item 1. For Item 2, note from Equation (17) and Item 2 of the induction hypothesis that:

$$\texttt{cnf}(v') = \texttt{cnf}(u') + 1 \leq \mathfrak{C}(\texttt{d}(\texttt{v}(u')) + 1).$$

If we are in Case 1 or in Case 3 of Equation (17), we have that the above inequality is strict and $\texttt{d}(\texttt{v}(u')) = \texttt{d}(\texttt{v}(v'))$ and Item 2 follows. On the other hand, if we are in Case 2 of Equation (17), we have that the above inequality is tight and $\texttt{d}(\texttt{v}(v')) = \texttt{d}(\texttt{nxt}(u')) = \texttt{d}(\texttt{v}(u')) + 1$

by the argument above. This means that $\texttt{cnf}(v') = \mathfrak{C}(\texttt{d}(\texttt{v}(u')) + 1) < \mathfrak{C}(\texttt{d}(\texttt{v}(v')) + 1)$, as desired for Item 2.

For Item 3, note that it is direct from the induction hypothesis unless we are in Case 2 of Equation (17). If we are in Case 2 of Equation (17), we have $\texttt{d}(\texttt{v}(v')) = \texttt{d}(\texttt{nxt}(u')) = \texttt{d}(\texttt{v}(u')) + 1$ is odd as above. From Equation (9), we get $\texttt{v}(v') \in V_B$. To prove Item 3, let $i \in [c]$ be such that $\texttt{d}(\texttt{v}_{i,A}(v')) + 1 < \texttt{d}(\texttt{v}(v'))$. This rearranges to $\texttt{d}(\texttt{v}_{i,A}(v')) < \texttt{d}(\texttt{v}(u'))$ which implies that $\texttt{d}(\texttt{v}_{i,A}(u')) < \texttt{d}(\texttt{v}(u'))$ by our derivation above. From the induction hypothesis on Item 3, we get that.

$$\mathfrak{C}(\texttt{d}(\texttt{v}_{i,A}(v')) + 1) = \mathfrak{C}(\texttt{d}(\texttt{v}_{i,A}(u')) + 1) < \mathfrak{C}(\texttt{d}(\texttt{v}(u')) + 1) \leq \texttt{cnf}(u') + 1 \leq \texttt{cnf}(v'),$$

as we are in Case 2 of Equation (17). Thus, Item 3 follows. For Item 4, if we are in Case 1 of Equation (17), we have that $\texttt{cnf}(v') = \texttt{cnf}(u') + 1 > \mathfrak{C}(\texttt{d}(\texttt{v}(u'))) = \mathfrak{C}(\texttt{d}(\texttt{v}(v')))$ from Equation (17). We also have that $\texttt{nxt}(v') \neq \texttt{rt}$ and that $(\texttt{v}(u'), \texttt{nxt}(v')) \in E$ is an edge in the original graph. It follows from Equation (9) that $\texttt{d}(\texttt{nxt}(v')) = \texttt{d}(\texttt{v}(u')) + 1 = \texttt{d}(\texttt{v}(v')) + 1$, as desired for Item 4. Next, if we are in Case 2 of Equation (17), we have that $\texttt{nxt}(v') = \texttt{rt}$ implying that $\texttt{d}(\texttt{v}(v')) + 1 \neq \texttt{d}(\texttt{nxt}(v'))$. From Item 2 of the induction hypothesis, we also have that $\texttt{cnf}(v') = \texttt{cnf}(u') + 1 = \mathfrak{C}(\texttt{d}(\texttt{v}(u')) + 1) = \mathfrak{C}(\texttt{d}(\texttt{v}(v')))$, finishing the proof. Finally, if we are in Case 3 of Equation (17), we must have $\texttt{cnf}(u') \neq \mathfrak{C}(\texttt{d}(\texttt{v}(u')))$. Thus, we have $\texttt{cnf}(u') \leq \mathfrak{C}(\texttt{d}(\texttt{v}(u'))) \iff \texttt{cnf}(v') \leq \mathfrak{C}(\texttt{d}(\texttt{v}(v')))$. The rest is straightforward from the induction hypothesis.

When $u' \in V'_A$ and $v' = u'_r$. In this case, Equation (18) applies. Item 1 is straightforward from the induction hypothesis unless we are in Case 1 of Equation (18). If we are in Case 1 of Equation (18), letting i''_1, i''_2 be as in Equation (18), we have by Item 1 of the induction hypothesis that $\texttt{d}(\texttt{v}(v')) = \texttt{d}\big(\texttt{v}_{i''_1,A}(u')\big) = (\log s)^{i''_1} \cdot \left\lfloor \frac{\texttt{d}(\texttt{v}(u'))}{(\log s)^{i''_1}} \right\rfloor$. This means that, for all $i \in [i''_1]$, we have:

$$\begin{aligned}
(\log s)^i \cdot \left\lfloor \frac{\texttt{d}(\texttt{v}(v'))}{(\log s)^i} \right\rfloor &= (\log s)^i \cdot \left\lfloor (\log s)^{i''_1 - i} \cdot \left\lfloor \frac{\texttt{d}(\texttt{v}(u'))}{(\log s)^{i''_1}} \right\rfloor \right\rfloor \\
&= (\log s)^{i''_1} \cdot \left\lfloor \frac{\texttt{d}(\texttt{v}(u'))}{(\log s)^{i''_1}} \right\rfloor \\
&= \texttt{d}(\texttt{v}(v')) \\
&= \texttt{d}(\texttt{v}_{i,A}(v')).
\end{aligned}$$

Next, consider $i \in (i''_1, i''_2]$. We claim that for all such i, we have $\texttt{d}(\texttt{v}_{i,A}(u')) = \texttt{d}\big(\texttt{v}_{i''_1,A}(u')\big)$. Indeed, the $\geq$ direction is because of the choice of i''_2 and the $\leq$ direction is because of Item 1 of the induction hypothesis. This means that for $i \in (i''_1, c]$, we have:

$$\texttt{d}(\texttt{v}_{i,A}(v')) = \texttt{d}(\texttt{v}_{i,A}(u')) \qquad \text{(As } \texttt{d}(\texttt{v}_{i,A}(u')) = \texttt{d}\big(\texttt{v}_{i''_1,A}(u')\big) \text{ for all } i \in (i''_1, i''_2])$$

$$= (\log s)^i \cdot \left\lfloor \frac{\mathtt{d}(\mathtt{v}(u'))}{(\log s)^i} \right\rfloor \qquad \text{(Induction hypothesis)}$$
$$= (\log s)^i \cdot \left\lfloor \frac{(\log s)^{i_1''}}{(\log s)^i} \cdot \frac{\mathtt{d}(\mathtt{v}(u'))}{(\log s)^{i_1''}} \right\rfloor$$
$$= (\log s)^i \cdot \left\lfloor \frac{(\log s)^{i_1''}}{(\log s)^i} \cdot \left\lfloor \frac{\mathtt{d}(\mathtt{v}(u'))}{(\log s)^{i_1''}} \right\rfloor \right\rfloor$$
$$\text{(As } i \in (i_1'', c] \text{ and } \lfloor \lfloor x \rfloor / n \rfloor = \lfloor x/n \rfloor \text{ for integers } n \geq 1 \text{ and } x \geq 0\text{)}$$
$$= (\log s)^i \cdot \left\lfloor \frac{\mathtt{d}(\mathtt{v}(v'))}{(\log s)^i} \right\rfloor.$$

Combining, we get the first equation in Item 1. Observe from Equation (18) that the second equation is straightforward if $i \in [i_2'']$, and that for $i \in (i_2'', c]$, we have by the choice of i_2'' and the first equation that $\max(\mathtt{d}(\mathtt{v}_{i,A}(u')), \mathtt{d}(\mathtt{v}_{i,A}(v'))) \leq \mathtt{d}(\mathtt{v}_{i_2''+1,A}(v')) = \mathtt{d}(\mathtt{v}_{i_2''+1,A}(u')) < \mathtt{d}(\mathtt{v}_{i_1'',A}(u')) = \mathtt{d}(\mathtt{v}(v')) \leq \mathtt{d}(\mathtt{v}(u'))$. The second equation now follows from the induction hypothesis.

For Item 2, note that $\mathtt{v}(v') \notin V_B$ and thus, Item 2 is straightforward from the induction hypothesis unless we are in Case 1 of Equation (18). If we are in Case 1 of Equation (18), letting i_1'', C' be as in Equation (18), we have:

$$\mathtt{cnf}(v') = \mathsf{C}' - 1 = \mathfrak{C}(\mathtt{d}(\mathtt{v}_{i_1'',A}(u'))) = \mathfrak{C}(\mathtt{d}(\mathtt{v}(v'))) < \mathfrak{C}(\mathtt{d}(\mathtt{v}(v')) + 1).$$

We now prove Item 3. Observe that $\mathtt{v}(u'), \mathtt{v}(v') \notin V_B$ and fix any $i \in [c]$ satisfying $\mathtt{d}(\mathtt{v}_{i,A}(v')) < \mathtt{d}(\mathtt{v}(v'))$. By Equation (18), we get that $i \in (i_2'', c]$. For these i, we have:

$$\begin{aligned}
\mathfrak{C}(\mathtt{d}(\mathtt{v}_{i,A}(v'))) &= \mathfrak{C}(\mathtt{d}(\mathtt{v}_{i,A}(u'))) && \text{(Equation (18))}\\
&\leq \mathfrak{C}(\mathtt{d}(\mathtt{v}_{i_2''+1,A}(u'))) && \text{(Item 1 of the induction hypothesis)}\\
&< \mathfrak{C}(\mathtt{d}(\mathtt{v}_{i_1'',A}(u'))) && \text{(As } \mathtt{d}(\mathtt{v}_{i_2''+1,A}(u')) < \mathtt{d}(\mathtt{v}_{i_1'',A}(u')) < \mathtt{d}(\mathtt{v}(u'))\text{)}\\
&< \mathtt{cnf}(u')\\
&\text{(As } \mathtt{d}(\mathtt{v}_{i_2''+1,A}(u')) < \mathtt{d}(\mathtt{v}_{i_1'',A}(u')) < \mathtt{d}(\mathtt{v}(u')) \text{ and Item 3 of the induction hypothesis)}\\
&\leq \max(\mathsf{C}', \mathtt{cnf}(u'))\\
&\leq \mathtt{cnf}(v') + 1.
\end{aligned}$$

We are done as all quantities are integers. It remains to show Item 4. In case of Case 1 of Equation (18), we have that $\mathtt{nxt}(v') = \mathtt{rt}$. It follows that $\mathtt{d}(\mathtt{v}(v')) + 1 \neq \mathtt{d}(\mathtt{nxt}(v'))$. Letting i_1'', C' be as in Equation (18), we also have $\mathtt{cnf}(v') \leq \mathsf{C}' - 1 = \mathfrak{C}(\mathtt{d}(\mathtt{v}_{i_1'',A}(u'))) \leq \mathfrak{C}(\mathtt{d}(\mathtt{v}(v')))$. In case of Case 2 of Equation (18), we have that $\mathtt{nxt}(v') = \mathtt{rt}$. It follows that $\mathtt{d}(\mathtt{v}(v')) + 1 \neq \mathtt{d}(\mathtt{nxt}(v'))$. Letting i_1'', C' be as in Equation (18), we also have $\mathtt{cnf}(v') \leq \mathtt{cnf}(u') - 1 \leq \mathfrak{C}(\mathtt{d}(\mathtt{v}(u'))) = \mathfrak{C}(\mathtt{d}(\mathtt{v}(v')))$ as we are in Case 2 of Equation (18). Finally, we are in Case 3 of Equation (18), we have $\mathtt{cnf}(u') > \mathfrak{C}(\mathtt{d}(\mathtt{v}(u'))) + 1$. This means that $\mathtt{cnf}(u') > \mathfrak{C}(\mathtt{d}(\mathtt{v}(u')))$ if and only if $\mathtt{cnf}(v') > \mathfrak{C}(\mathtt{d}(\mathtt{v}(v')))$. Item 4 now follows from Item 4 of the induction hypothesis.

When $u' \in V_B'$ and $v' \neq u_r'$. In this case, Equation (19) applies. For Item 1, note that it is direct from the induction hypothesis unless we are in Case 2 of Equation (19). If we are in Case 2 of Equation (19), we have $\mathtt{d}(\mathtt{v}(v')) = \mathtt{d}(\mathtt{nxt}(u')) = \mathtt{d}(\mathtt{v}(u')) + 1$ by Item 4 of the induction hypothesis on u'. As $u' \in V_B'$ implying that $\mathtt{v}(u') \in V_B$, we get from Equation (9) that $\mathtt{d}(\mathtt{v}(v')) = \mathtt{d}(\mathtt{v}(u')) + 1$ is even. Letting i' be as defined before Equation (19), we have that $(\log s)^{i'} \mid \mathtt{d}(\mathtt{v}(v'))$ for all $i \in [i']$. Item 1 is now straightforward for all $i \in [i']$. For $i \in (i', c]$, we have that $(\log s)^i \nmid \mathtt{d}(\mathtt{v}(v'))$ by our choice of i'. we get:

$$\begin{aligned}\mathtt{d}(\mathtt{v}_{i,A}(v')) &= \mathtt{d}(\mathtt{v}_{i,A}(u')) \\ &= (\log s)^i \cdot \left\lfloor \frac{\mathtt{d}(\mathtt{v}(u'))}{(\log s)^i} \right\rfloor && \text{(Induction hypothesis)} \\ &= (\log s)^i \cdot \left\lfloor \frac{\mathtt{d}(\mathtt{v}(v'))}{(\log s)^i} \right\rfloor. && \text{(As } (\log s)^i \nmid \mathtt{d}(\mathtt{v}(v'))\text{)}\end{aligned}$$

This show the first equation in Item 1. Observe from Equation (19) that the second equation is straightforward when $i \in [i']$. For $i \in (i', c]$, we have $\mathtt{d}(\mathtt{v}_{i,A}(v')) = \mathtt{d}(\mathtt{v}_{i,A}(u')) = (\log s)^i \cdot \left\lfloor \frac{\mathtt{d}(\mathtt{v}(u'))}{(\log s)^i} \right\rfloor < \mathtt{d}(\mathtt{v}(u'))$ by the choice of i'. The second equation now follows from the induction hypothesis. For Item 2, note from Equation (19) and Item 2 of the induction hypothesis that:

$$\mathtt{cnf}(v') = \mathtt{cnf}(u') + 1 \leq \mathfrak{C}(\mathtt{d}(\mathtt{v}(u')) + 1).$$

If we are in Case 1 or in Case 3 of Equation (19), we have that the above inequality is strict and $\mathtt{d}(\mathtt{v}(u')) = \mathtt{d}(\mathtt{v}(v'))$ and Item 2 follows. On the other hand, if we are in Case 2 of Equation (19), we have that the above inequality is tight and $\mathtt{d}(\mathtt{v}(v')) = \mathtt{d}(\mathtt{nxt}(u')) = \mathtt{d}(\mathtt{v}(u')) + 1$ by the argument above. This means that $\mathtt{cnf}(v') = \mathfrak{C}(\mathtt{d}(\mathtt{v}(u')) + 1) < \mathfrak{C}(\mathtt{d}(\mathtt{v}(v')) + 1)$, as desired for Item 2.

For Item 3, note that it is direct from the induction hypothesis unless we are in Case 2 of Equation (19). Suppose that we are in Case 2 of Equation (19) and that $i \in [c]$ is such that $\mathtt{d}(\mathtt{v}_{i,A}(v')) < \mathtt{d}(\mathtt{v}(v'))$. By Equation (19), this means that $i \in (i', c]$ and we have:

$$\mathfrak{C}(\mathtt{d}(\mathtt{v}_{i,A}(v'))) = \mathfrak{C}(\mathtt{d}(\mathtt{v}_{i,A}(u'))) \leq \mathfrak{C}(\mathtt{d}(\mathtt{v}(u'))) < \mathfrak{C}(\mathtt{d}(\mathtt{v}(u')) + 1) \leq \mathtt{cnf}(u') + 1 = \mathtt{cnf}(v').$$

For Item 4, if we are in Case 1 of Equation (19), we have that $\mathtt{cnf}(v') = \mathtt{cnf}(u') + 1 > \mathfrak{C}(\mathtt{d}(\mathtt{v}(u'))) = \mathfrak{C}(\mathtt{d}(\mathtt{v}(v')))$ from Equation (19). We also have that $\mathtt{nxt}(v') \neq \mathtt{rt}$ and that $(\mathtt{v}(u'), \mathtt{nxt}(v')) \in E$ is an edge in the original graph. It follows from Equation (9) that $\mathtt{d}(\mathtt{nxt}(v')) = \mathtt{d}(\mathtt{v}(u')) + 1 = \mathtt{d}(\mathtt{v}(v')) + 1$, as desired for Item 4. Next, if we are in Case 2 of Equation (19), we have that $\mathtt{nxt}(v') = \mathtt{rt}$ implying that $\mathtt{d}(\mathtt{v}(v')) + 1 \neq \mathtt{d}(\mathtt{nxt}(v'))$. From Item 2 of the induction hypothesis, we also have that $\texttt{cnf}(v') = \texttt{cnf}(u') + 1 = \mathfrak{C}(\texttt{d}(\texttt{v}(u')) + 1) = \mathfrak{C}(\texttt{d}(\texttt{v}(v')))$, finishing the proof. Finally, if we are in Case 3 of Equation (19), we must have $\texttt{cnf}(u') \neq \mathfrak{C}(\texttt{d}(\texttt{v}(u')))$. Thus, we have $\texttt{cnf}(u') \leq \mathfrak{C}(\texttt{d}(\texttt{v}(u'))) \iff \texttt{cnf}(v') \leq \mathfrak{C}(\texttt{d}(\texttt{v}(v')))$. The rest is straightforward from the induction hypothesis.

When $u' \in V'_B$ and $v' = u'_r$. In this case, Equation (20) applies. Item 1 is straightforward from the induction hypothesis unless we are in Case 1 of Equation (20). If we are in Case 1 of Equation (20), letting i'' be as in Equation (20), for all $i \in [i'']$, we have:

$$\begin{aligned}
\texttt{d}(\texttt{v}_{i,A}(v')) &= \texttt{d}(\texttt{v}_{i'',A}(u')) \\
&= (\log s)^{i''} \cdot \left\lfloor \frac{\texttt{d}(\texttt{v}(u'))}{(\log s)^{i''}} \right\rfloor && \text{(Induction hypothesis)} \\
&= (\log s)^{i} \cdot \left\lfloor (\log s)^{i''-i} \cdot \left\lfloor \frac{\texttt{d}(\texttt{v}(u'))}{(\log s)^{i''}} \right\rfloor \right\rfloor \\
&= (\log s)^{i} \cdot \left\lfloor \frac{\texttt{d}(\texttt{v}_{i'',A}(u'))}{(\log s)^{i}} \right\rfloor .
\end{aligned}$$

Also, for $i \in (i'', c]$, we have:

$$\begin{aligned}
\texttt{d}(\texttt{v}_{i,A}(v')) &= \texttt{d}(\texttt{v}_{i,A}(u')) \\
&= (\log s)^{i} \cdot \left\lfloor \frac{\texttt{d}(\texttt{v}(u'))}{(\log s)^{i}} \right\rfloor && \text{(Induction hypothesis)} \\
&= (\log s)^{i} \cdot \left\lfloor \frac{(\log s)^{i''}}{(\log s)^{i}} \cdot \frac{\texttt{d}(\texttt{v}(u'))}{(\log s)^{i''}} \right\rfloor \\
&= (\log s)^{i} \cdot \left\lfloor \frac{(\log s)^{i''}}{(\log s)^{i}} \cdot \left\lfloor \frac{\texttt{d}(\texttt{v}(u'))}{(\log s)^{i''}} \right\rfloor \right\rfloor \\
& && \text{(As } i \in (i'', c] \text{ and } \lfloor \lfloor x \rfloor / n \rfloor = \lfloor x/n \rfloor \text{ for integers } n \geq 1 \text{ and } x \geq 0) \\
&= (\log s)^{i} \cdot \left\lfloor \frac{\texttt{d}(\texttt{v}_{i'',A}(u'))}{(\log s)^{i}} \right\rfloor . && \text{(Item 1 of the induction hypothesis)}
\end{aligned}$$

Thus, in either case, we have $\texttt{d}(\texttt{v}_{i,A}(v')) = (\log s)^{i} \cdot \left\lfloor \frac{\texttt{d}(\texttt{v}_{i'',A}(u'))}{(\log s)^{i}} \right\rfloor$. For all $i \in [c]$, we finish the proof as follows:

$$\begin{aligned}
\texttt{d}(\texttt{v}_{i,A}(v')) &= (\log s)^{i} \cdot \left\lceil \frac{\texttt{d}(\texttt{v}_{i'',A}(u')) + 1}{(\log s)^{i}} \right\rceil && \text{(As } (\log s)^{i} \text{ and } \texttt{d}(\texttt{v}_{i'',A}(u')) \text{ is even)} \\
&= (\log s)^{i} \cdot \left\lceil \frac{\texttt{d}(\texttt{v}_{i'',B}(u'))}{(\log s)^{i}} \right\rceil
\end{aligned}$$

(Item 1 of the induction hypothesis and $2 \mid \texttt{d}(\texttt{v}_{i'',A}(u'))$ and that $u' \in V'_B \implies 2 \nmid \texttt{d}(\texttt{v}(u'))$)

$$= (\log s)^i \cdot \left\lfloor \frac{\mathtt{d}(\mathtt{v}(v'))}{(\log s)^i} \right\rfloor.$$

Thus, we get the first equation in Item 1. For the second equation, note from Item 1 of the induction hypothesis that for all $i \in [c]$, we have $2 \mid \mathtt{d}(\mathtt{v}_{i,A}(u'))$ and that $u' \in V'_B \implies 2 \nmid \mathtt{d}(\mathtt{v}(u'))$. From Item 1 of the induction hypothesis, this means that, for all $i \in [c]$, we have $\mathtt{d}(\mathtt{v}_{i,A}(u')) < \mathtt{d}(\mathtt{v}(u'))$. Moreover, it means that $\mathtt{d}(\mathtt{v}(v')) = \mathtt{d}(\mathtt{v}_{i'',B}(u')) = \mathtt{d}(\mathtt{v}_{i'',A}(u')) + 1$ is odd. By the first equation for v', this gives that for all $i \in [c]$, we have $\mathtt{d}(\mathtt{v}_{i,A}(v')) < \mathtt{d}(\mathtt{v}(v'))$. The second equation now follows directly from the induction hypothesis, finishing the proof of Item 1.

For Item 2, note that it is straightforward from the induction hypothesis unless we are in Case 1 of Equation (20). If we are in Case 1 of Equation (20), letting i'', C' be as in Equation (20), we have:

$$\mathtt{cnf}(v') = \mathsf{C}' - 1 = \mathfrak{C}(\mathtt{d}(\mathtt{v}_{i'',B}(u'))) = \mathfrak{C}(\mathtt{d}(\mathtt{v}(v'))) < \mathfrak{C}(\mathtt{d}(\mathtt{v}(v')) + 1).$$

We now prove Item 3. Observe that $\mathtt{v}(u'), \mathtt{v}(v') \in V_B$ and fix any $i \in [c]$ for which we have $\mathtt{d}(\mathtt{v}_{i,A}(v')) + 1 < \mathtt{d}(\mathtt{v}(v'))$. By Equation (20), we get that $\mathtt{d}(\mathtt{v}_{i,A}(v')) + 1 < \mathtt{d}(\mathtt{v}(v')) = \mathtt{d}(\mathtt{v}_{i'',B}(u')) = \mathtt{d}(\mathtt{v}_{i'',A}(u')) + 1$. It follows that $i \in (i'', c]$. As $i \in (i'', c]$, we further get that $\mathtt{d}(\mathtt{v}_{i'',A}(u')) + 1 = \mathtt{d}(\mathtt{v}_{i'',B}(u')) < \mathtt{d}(\mathtt{v}(u'))$. Thus, we have:

$$\begin{aligned}
\mathfrak{C}(\mathtt{d}(\mathtt{v}_{i,A}(v')) + 1) &< \mathfrak{C}(\mathtt{d}(\mathtt{v}(v'))) \\
&= \mathfrak{C}(\mathtt{d}(\mathtt{v}_{i'',A}(u')) + 1) \\
&< \mathtt{cnf}(u') \\
&\text{(As } \mathtt{d}(\mathtt{v}_{i'',A}(u')) + 1 < \mathtt{d}(\mathtt{v}(u')) \text{ and Item 3 of the induction hypothesis)} \\
&\le \max(\mathsf{C}', \mathtt{cnf}(u')) \\
&\le \mathtt{cnf}(v') + 1.
\end{aligned}$$

We are done as all quantities are integers. It remains to show Item 4. In case of Case 1 of Equation (20), we have that $\mathtt{nxt}(v') = \mathsf{rt}$. It follows that $\mathtt{d}(\mathtt{v}(v')) + 1 \neq \mathtt{d}(\mathtt{nxt}(v'))$. Letting i'', C' be as in Equation (20), we also have $\mathtt{cnf}(v') \le \mathsf{C}' - 1 = \mathfrak{C}(\mathtt{d}(\mathtt{v}_{i'',B}(u'))) \le \mathfrak{C}(\mathtt{d}(\mathtt{v}(v')))$. In case of Case 2 of Equation (20), we have that $\mathtt{nxt}(v') = \mathsf{rt}$. It follows that $\mathtt{d}(\mathtt{v}(v')) + 1 \neq \mathtt{d}(\mathtt{nxt}(v'))$. Letting i'', C' be as in Equation (20), we also have $\mathtt{cnf}(v') \le \mathtt{cnf}(u') - 1 \le \mathfrak{C}(\mathtt{d}(\mathtt{v}(u'))) = \mathfrak{C}(\mathtt{d}(\mathtt{v}(v')))$ as we are in Case 2 of Equation (20). Finally, we are in Case 3 of Equation (20), we have $\mathtt{cnf}(u') > \mathfrak{C}(\mathtt{d}(\mathtt{v}(u'))) + 1$. This means that $\mathtt{cnf}(u') > \mathfrak{C}(\mathtt{d}(\mathtt{v}(u')))$ if and only if $\mathtt{cnf}(v') > \mathfrak{C}(\mathtt{d}(\mathtt{v}(v')))$. Item 4 now follows from Item 4 of the induction hypothesis. □

Lemma 4.7. *Let $u', v' \in V'$ be reachable from rt' such that $(u', v') \in E'$ and $v' \neq u'_r$.*

1. *It holds that:*

$$\mathtt{cnf}(v') = \mathtt{cnf}(u') + 1 \qquad \mathtt{rew}_A(v') = \mathtt{rew}_A(u') \qquad \mathtt{rew}_B(v') = \mathtt{rew}_B(u').$$

2. *We have* $\mathtt{S}(v') \setminus \{\mathtt{v}(v')\} \subseteq \mathtt{S}(u')$ *and also that* $\mathtt{v}(v') \notin \mathtt{S}(u')$ *implies:*

$$\mathtt{d}(\mathtt{v}(v')) = \mathtt{d}(\mathtt{v}(u')) + 1 \qquad\qquad \mathtt{cnf}(v') = \mathtt{cnf}(u') + 1 = \mathfrak{C}(\mathtt{d}(\mathtt{v}(v'))).$$

Proof. As $v' \neq u'_r$, one of Equations (17) and (19) applies. Item 1 is straightforward from Equations (17) and (19). For Item 2, $\mathtt{S}(v') \setminus \{\mathtt{v}(v')\} \subseteq \mathtt{S}(u')$ again follows from straightforwardly from Equations (17) and (19). For the rest of Item 2, assume that $\mathtt{v}(v') \notin \mathtt{S}(u')$. This means that either Case 2 of Equation (17) or Case 2 of Equation (19) applies. In either case, the second equation is direct from Item 2 of Lemma 4.6 and the fact that it is Case 2 and the first equation is because of Item 4 of Lemma 4.6. □

Lemma 4.8. *Let* $u', v' \in V'$ *be reachable from* rt' *such that* $v' = u'_r$ *(and thus* $(u', v') \in E'$*).*

1. *It holds that:*

$$\begin{aligned}
\mathtt{rew}_A(v') &= \mathtt{rew}_A(u') + \mathbb{1}(u' \in V'_A),\\
\mathtt{rew}_B(v') &= \mathtt{rew}_B(v') + \mathbb{1}(u' \in V'_B),\\
\mathtt{cnf}(v') + 1 &= \mathtt{cnf}(u') + \mathbb{1}(\mathtt{d}(\mathtt{v}(u')) < 2 \wedge \mathtt{cnf}(u') = \mathfrak{C}(\mathtt{d}(\mathtt{v}(u')))).
\end{aligned}$$

2. *We have* $\mathtt{nxt}(v') \in \{\mathtt{nxt}(u'), \mathsf{rt}\}$*.*

3. *We have* $\mathtt{S}(v') \subseteq \mathtt{S}(u')$*.*

Proof. As $v' = u'_r$, one of Equations (18) and (20) applies. Items 2 and 3 are straightforward from Equations (18) and (20). For Item 1, the equations about $\mathtt{rew}_A(v')$ and $\mathtt{rew}_B(v')$ are straightforward. For the equation about $\mathtt{cnf}(v')$, assume first that $u' \in V'_A$ so that Equation (18) applies. Let i''_1, C' be as defined in Equation (18). If $\mathtt{d}(\mathtt{v}(u')) \geq 2$, we have by Item 1 of Lemma 4.6 that $\mathtt{d}\big(\mathtt{v}_{i''_1,A}(u')\big) < \mathtt{d}(\mathtt{v}(u'))$. Putting this in Item 3 of Lemma 4.6, we get that $\mathsf{C}' - 1 = \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}_{i''_1,A}(u')\big)\big) < \mathtt{cnf}(u')$. From Equation (18), this means that $\mathtt{cnf}(v') + 1 = \mathtt{cnf}(u')$, as desired. Assume now that $\mathtt{d}(\mathtt{v}(u')) < 2$, we have by $u' \in V'_A$ that $\mathtt{v}(u') \in V_A$ implying by Equation (9) that $\mathtt{d}(\mathtt{v}(u')) = 0$. This means that $i''_1 = c$ and $\mathsf{C}' = 1$. We get:

$$\begin{aligned}
\mathtt{cnf}(v') + 1 &= \max(\mathsf{C}', \mathtt{cnf}(u'))\\
&= \mathtt{cnf}(u') + \max(1 - \mathtt{cnf}(u'), 0)\\
&= \mathtt{cnf}(u') + \mathbb{1}(\mathtt{cnf}(u') = 0)\\
&= \mathtt{cnf}(u') + \mathbb{1}(\mathtt{d}(\mathtt{v}(u')) < 2 \wedge \mathtt{cnf}(u') = \mathfrak{C}(\mathtt{d}(\mathtt{v}(u')))).
\end{aligned}$$

Now, assume that $u' \in V'_B$ so that Equation (20) applies. Let i'', C' be as defined in Equation (20). If $\mathtt{d}(\mathtt{v}(u')) \geq 2$, we have by Item 1 of Lemma 4.6 that $\mathtt{d}(\mathtt{v}_{i'',B}(u')) = \mathtt{d}(\mathtt{v}_{i'',A}(u')) + 1 < \mathtt{d}(\mathtt{v}(u'))$. Putting this in Item 3 of Lemma 4.6, we get that $\mathsf{C}' - 1 = \mathfrak{C}(\mathtt{d}(\mathtt{v}_{i'',B}(u'))) < \mathtt{cnf}(u')$. From Equation (20), this means that $\mathtt{cnf}(v') + 1 = \mathtt{cnf}(u')$, as desired. Assume now that $\mathtt{d}(\mathtt{v}(u')) < 2$, we have by $u' \in V'_B$ that $\mathtt{v}(u') \in V_B$ implying by Equation (9) that $\mathtt{d}(\mathtt{v}(u')) = 1$. This means that $i''_1 = c$ and $\mathsf{C}' = \mathfrak{C}(1) + 1$. We get:

$$\begin{aligned}\mathtt{cnf}(v') + 1 &= \max(\mathsf{C}', \mathtt{cnf}(u')) \\ &= \mathtt{cnf}(u') + \max(\mathfrak{C}(1) + 1 - \mathtt{cnf}(u'), 0) \\ &= \mathtt{cnf}(u') + \mathbb{1}(\mathtt{cnf}(u') = \mathfrak{C}(1)) && \text{(Lemma 4.6, Item 2)} \\ &= \mathtt{cnf}(u') + \mathbb{1}(\mathtt{d}(\mathtt{v}(u')) < 2 \wedge \mathtt{cnf}(u') = \mathfrak{C}(\mathtt{d}(\mathtt{v}(u')))).\end{aligned}$$

□

Lemma 4.9. *Let $u', v' \in V'$ be reachable from rt' such that $(u', v') \in E'$.*

1. *It holds that:*

$$\begin{aligned}(2 \cdot (\mathtt{rew}_A(v') + \mathtt{rew}_B(v')) + \mathtt{cnf}(v')) - (2 \cdot (\mathtt{rew}_A(u') + \mathtt{rew}_B(u')) + \mathtt{cnf}(u')) \\ = 1 + \mathbb{1}(v' = u'_r \wedge \mathtt{d}(\mathtt{v}(u')) < 2 \wedge \mathtt{cnf}(u') = \mathfrak{C}(\mathtt{d}(\mathtt{v}(u')))).\end{aligned}$$

2. *It holds that:*

$$\begin{aligned}\mathbb{1}(v' \neq u'_r) - \mathbb{1}(v' = u'_r) = \mathtt{cnf}(v') - \mathtt{cnf}(u') \\ - \mathbb{1}(v' = u'_r \wedge \mathtt{d}(\mathtt{v}(u')) < 2 \wedge \mathtt{cnf}(u') = \mathfrak{C}(\mathtt{d}(\mathtt{v}(u')))).\end{aligned}$$

3. *It holds that $\mathtt{d}(\mathtt{v}(v')) \leq \mathtt{d}(\mathtt{v}(u')) + 1$ and $\mathtt{d}(\mathtt{v}(v')) \neq \mathtt{d}(\mathtt{v}(u')) - 1$. Moreover, if $\mathtt{d}(\mathtt{v}(v')) \neq \mathtt{d}(\mathtt{v}(u'))$, we have:*

$$\mathtt{cnf}(v') = \mathfrak{C}(\mathtt{d}(\mathtt{v}(v'))).$$

4. *We have $\mathtt{S}(v') \subseteq \mathtt{S}(u') \cup \{\mathtt{nxt}(u')\}$.*

Proof. We prove each part in turn.

1. If $v' \neq u'_r$, this simplifies to showing that

$$(2 \cdot (\mathtt{rew}_A(v') + \mathtt{rew}_B(v')) + \mathtt{cnf}(v')) - (2 \cdot (\mathtt{rew}_A(u') + \mathtt{rew}_B(u')) + \mathtt{cnf}(u')) = 1.$$

   This follows from Item 1 of Lemma 4.7. If $v' = u'_r$, this simplifies to showing that:

$$\begin{aligned}(2 \cdot (\mathtt{rew}_A(v') + \mathtt{rew}_B(v')) + \mathtt{cnf}(v')) - (2 \cdot (\mathtt{rew}_A(u') + \mathtt{rew}_B(u')) + \mathtt{cnf}(u')) \\ = 1 + \mathbb{1}(\mathtt{d}(\mathtt{v}(u')) < 2 \wedge \mathtt{cnf}(u') = \mathfrak{C}(\mathtt{d}(\mathtt{v}(u')))),\end{aligned}$$

   which follows from Item 1 of Lemma 4.8.

2. If $v' \neq u'_r$, this simplifies to showing that $1 = \texttt{cnf}(v') - \texttt{cnf}(u')$. This follows from Item 1 of Lemma 4.7. If $v' = u'_r$, this simplifies to showing that:

$$-1 = \texttt{cnf}(v') - \texttt{cnf}(u') - \mathbb{1}(\texttt{d}(\texttt{v}(u')) < 2 \wedge \texttt{cnf}(u') = \mathfrak{C}(\texttt{d}(\texttt{v}(u')))),$$

which follows from Item 1 of Lemma 4.8.

3. We first show that $\texttt{d}(\texttt{v}(v')) \leq \texttt{d}(\texttt{v}(u')) + 1$. If $v' = u'_r$, we have that $\texttt{v}(v') \in \texttt{S}(v') \subseteq \texttt{S}(u')$ by Item 3 of Lemma 4.8, and the statement follows from Item 1 of Lemma 4.6. On the other hand, if $v' \neq u'_r$, we will show the stronger statement that $\texttt{d}(\texttt{v}(u')) \leq \texttt{d}(\texttt{v}(v')) \leq \texttt{d}(\texttt{v}(u')) + 1$. By Equations (17) and (19), it suffices to consider the case when $\mathfrak{C}(\texttt{d}(\texttt{v}(u')) + 1) \leq \texttt{cnf}(u') + 1$ but in this case, the statement follows from Item 4 of Lemma 4.6. We now show that $\texttt{d}(\texttt{v}(v')) \neq \texttt{d}(\texttt{v}(u')) - 1$. By our stronger statement above, it suffices to consider the case when $v' = u'_r$. Observe from Equations (18) and (20) and Item 1 of Lemma 4.6 that, in this case, we have $\texttt{d}(\texttt{v}(v')) - \texttt{d}(\texttt{v}(u'))$ is even and the claim follows.

   It remains to show that if $\texttt{d}(\texttt{v}(v')) \neq \texttt{d}(\texttt{v}(u'))$, we have $\texttt{cnf}(v') = \mathfrak{C}(\texttt{d}(\texttt{v}(v')))$. This is direct from Equations (18) and (20) if $v' = u'_r$ so we assume that $v' \neq u'_r$. From our stronger statement above, we get that $\texttt{d}(\texttt{v}(v')) = \texttt{d}(\texttt{v}(u')) + 1$. By Item 1 of Lemma 4.6, we get that $\texttt{v}(v') \notin \texttt{S}(u')$ and the proof is complete using Item 2 of Lemma 4.7.

4. We first show that $\texttt{S}(v') \subseteq \texttt{S}(u') \cup \{\texttt{nxt}(u')\}$. As this is direct from Item 3 of Lemma 4.8 if $v' = u'_r$, we assume without loss of generality that $v' \neq u'_r$. From Item 2 of Lemma 4.7, we get that it suffices to show that $\texttt{v}(v') \in \texttt{S}(u') \cup \{\texttt{nxt}(u')\}$. This follows from Equations (17) and (19).

$\square$

Corollary 4.10 (Corollary of Item 1 of Lemma 4.9 and Equations (13) and (14))**.** *The depth of* Π' *is at most* $5 \cdot \nabla \leq 150c \cdot \Theta$*.*

4.3 Proof of Theorem 3.7

Proof of Theorem 3.7. The claim about the size is straightforward and the claim about the depth of Π' is proved in Corollary 4.10. We show that Π' solves S with rectangular correctness despite $\Xi_\Theta(\Pi')$. By definition, this amounts to showing that Π'_Θ solves $S_{\Xi_\Theta(\Pi')}$ with rectangular correctness, where Π'_Θ and $S_{\Xi_\Theta(\Pi')}$ are as defined in Section 3.3. By Definition 3.3 and the definition of $S_{\Xi_\Theta(\Pi')}$, this amounts to showing that for all $v' \in V'_O$ and all $((x, \xi_A), (y, \xi_B)) \in R'_{\Theta,v'}$, we have $(x, y, o'_{v'}) \in S$, where $R'_{\Theta,v'}$ are the rectangles associated with Π'_Θ according to Definition 3.1. Fix an arbitrary $v' \in V'_O$. By definition of V'_O, we have that either $\texttt{v}(v') \in V_O$ or $\max(\texttt{rew}_A(v'), \texttt{rew}_B(v')) = \nabla$. This means that the following lemmas together imply the theorem.

Lemma 4.11. *If $\mathtt{v}(v') \in V_O$, then, for all $((x, \xi_A), (y, \xi_B)) \in R'_{\Theta,v'}$, we have $(x, y, o'_{v'}) \in S$.*

Lemma 4.12. *If $\max(\mathtt{rew}_A(v'), \mathtt{rew}_B(v')) = \nabla$, we have $R'_{\Theta,v'} = \emptyset$.*

□

We now prove the two lemmas. As our proof of Lemma 4.11 is shorter, we include it here. The longer proof of Lemma 4.12 forms the following section.

Proof of Lemma 4.11. Proof by contradiction. Suppose that there exists $((x, \xi_A), (y, \xi_B)) \in R'_{\Theta,v'}$ such that $(x, y, o'_{v'}) \notin S$. By definition of $o'_{v'}$, we get that $\left(x, y, o_{\mathtt{v}(v')}\right) \notin S$. As Π solves S with rectangular correctness, this means that[16] $(x, y) \notin R_{\mathtt{v}(v')}$, implying that either $x \notin \left(R_{\mathtt{v}(v')}\right)_A$ or $y \notin \left(R_{\mathtt{v}(v')}\right)_B$. We assume henceforth that $x \notin \left(R_{\mathtt{v}(v')}\right)_A$ as the proof in the case $y \notin \left(R_{\mathtt{v}(v')}\right)_B$ is similar.

As $((x, \xi_A), (y, \xi_B)) \in R'_{\Theta,v'}$, we can apply Observation 3.2 repeatedly to get that there exists an $\ell > 0$ and a sequence of vertices $\mathtt{rt}' = v'^{(0)}, v'^{(1)}, \ldots, v'^{(\ell)} = v'$ such that for all $l \in [\ell]$, we have $\left(v'^{(l-1)}, v'^{(l)}\right) \in E'$ and:

$$\exists\left((x, \xi_A), \left(y^{(l-1)}, \xi_B^{(l-1)}\right)\right) \in R'_{\Theta,v'^{(l-1)}} : \qquad h'_{\Theta,v'^{(l-1)}}\left((x, \xi_A), \left(y^{(l-1)}, \xi_B^{(l-1)}\right)\right) = v'^{(l)}. \quad (23)$$

We define $\left(y^{(\ell)}, \xi_B^{(\ell)}\right) = (y, \xi_B)$ for convenience. Henceforth, to simplify notation for the terms in Equation (15), we will abbreviate $\mathtt{v}\left(v'^{(l)}\right)$ to $\mathtt{v}^{(l)}$, $\mathtt{nxt}\left(v'^{(l)}\right)$ to $\mathtt{nxt}^{(l)}$, $\mathtt{cnf}\left(v'^{(l)}\right)$ to $\mathtt{cnf}^{(l)}$ *etc.* for all $0 \le l \le \ell$. From Equation (9), we have that $\mathtt{d}\left(\mathtt{v}^{(\ell)}\right) = (\log s)^{c-1} \ge 4$ is an even number.

Define α (respectively β) to be the largest value $l \in [\ell]$ such that $\mathtt{d}\left(\mathtt{v}^{(l-1)}\right) < (\log s)^{c-1} - 2$ (respectively, $\mathtt{d}\left(\mathtt{v}^{(l-1)}\right) < (\log s)^{c-1} - 1$). As $l = 0$ is one such value, both α and β are well defined. By our choice of α and β and Item 3 of Lemma 4.9, we have that:

$$\begin{aligned} \mathtt{d}\left(\mathtt{v}^{(\alpha-1)}\right) &= (\log s)^{c-1} - 3 & \qquad \mathtt{d}\left(\mathtt{v}^{(\alpha)}\right) &= (\log s)^{c-1} - 2, \\ \mathtt{d}\left(\mathtt{v}^{(\beta-1)}\right) &= (\log s)^{c-1} - 2 & \qquad \mathtt{d}\left(\mathtt{v}^{(\beta)}\right) &= (\log s)^{c-1} - 1. \end{aligned} \quad (24)$$

Observe from these and the definition of α and β that we must have $\alpha < \beta$. We next claim that, for all $l \in (\alpha, \beta]$, we have that $\mathtt{d}\left(\mathtt{v}^{(l-1)}\right) = (\log s)^{c-1} - 2$ implying that $v'^{(l-1)} \in V'_A$. Indeed, the $\ge$ direction directly follows from the definition of α. Assuming for the sake of contradiction that the $\le$ direction does not hold, we have an $l \in (\alpha, \beta]$ such that $\mathtt{d}\left(\mathtt{v}^{(l-1)}\right) > (\log s)^{c-1} - 2$. Pick the largest such l. Note that $l < \beta$ because of Equation (24) and that the $\ge$ direction with our choice of l implies that $\mathtt{d}\left(\mathtt{v}^{(l-1)}\right) = (\log s)^{c-1} - 1 > \mathtt{d}\left(\mathtt{v}^{(l)}\right) = (\log s)^{c-1} - 2$ (as a larger value would imply that $v'^{(l-1)}$ is a leaf node, see Equation (9)). This contradicts Item 3 of Lemma 4.9.

[16] Recall that $\{R_v\}_{v \in V}$ be the rectangles associated with Π as in Definition 3.1.

In addition to the claim above, by Equations (17) to (20) and Item 2 of Lemma 4.6, Equation (24) also says that $\mathtt{cnf}^{(\alpha)} = \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(\alpha)}\big)\big)$ and $\mathtt{cnf}^{(\beta)} = \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(\beta)}\big)\big)$. This means that:

$$\begin{aligned}
\mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(\beta)}\big)\big) - \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(\alpha)}\big)\big) &= \mathtt{cnf}^{(\beta)} - \mathtt{cnf}^{(\alpha)} \\
&\le \sum_{l \in (\alpha,\beta]} \mathbb{1}\big(\mathtt{cnf}^{(l)} = \mathtt{cnf}^{(l-1)} + 1\big) && \text{(Lemma 4.9, Item 2)} \\
&\le \sum_{l \in (\alpha,\beta]} \mathbb{1}\big(v'^{(l)} \ne v_r'^{(l-1)}\big). && \text{(Lemma 4.8, Item 1)}
\end{aligned}$$

To continue, we show that for all $l \in (\alpha,\beta]$, we have that $v'^{(l)} \ne v_r'^{(l-1)}$ only if $\xi_A\big(v'^{(l-1)}\big) \ne *$ (recall that $v'^{(l-1)} \in V_A'$ for all $l \in (\alpha,\beta]$). Indeed, suppose for the sake of contradiction that we have $l \in (\alpha,\beta]$ such that $v'^{(l)} \ne v_r'^{(l-1)}$ and $\xi_A\big(v'^{(l-1)}\big) = *$. This means that:

$$\begin{aligned}
v_r'^{(l-1)} &\ne v'^{(l)} \\
&= h'_{\Theta, v'^{(l-1)}}\Big((x,\xi_A), \Big(y^{(l-1)}, \xi_B^{(l-1)}\Big)\Big) && \text{(Equation (23))} \\
&= h'_{v'^{(l-1)}}\big(x, y^{(l-1)}\big) && \text{(Equation (3) and } \xi_A\big(v'^{(l-1)}\big) = *\text{)}
\end{aligned}$$

From Equation (21), we get that $x \in \big(R_{\mathtt{v}^{(l-1)}}\big)_A$. However, as $\mathtt{d}\big(\mathtt{v}^{(l-1)}\big) = (\log s)^{c-1} - 2$, Equation (10) says that has exactly one outgoing edge in V, and so does the vertex that edge leads to. By Definition 3.1, we get that $R_{\mathtt{v}^{(l-1)}} \subseteq R_{\mathtt{v}^{(\ell)}}$ implying that $x \in \big(R_{\mathtt{v}^{(\ell)}}\big)_A = \big(R_{\mathtt{v}(v')}\big)_A$, a contradiction. This allows us to continue as:

$$\begin{aligned}
\mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(\beta)}\big)\big) - \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(\alpha)}\big)\big) &\le \sum_{l \in (\alpha,\beta]} \mathbb{1}\big(\xi_A\big(v'^{(l-1)}\big) \ne *\big) \\
&\le \sum_{l \in (\alpha,\beta]} \mathbb{1}\big(v'^{(l-1)} \in V_A' \wedge \xi_A\big(v'^{(l-1)}\big) \ne *\big). \\
& \text{(As } v'^{(l-1)} \in V_A' \text{ for all } l \in (\alpha,\beta]\text{)}
\end{aligned}$$

Using Equations (12) and (24), we get:

$$\mathfrak{C}\big((\log s)^{c-1}\big) - \mathfrak{C}\big((\log s)^{c-1} - 1\big) \le c \cdot \sum_{l \in (\alpha,\beta]} \mathbb{1}\big(v'^{(l-1)} \in V_A' \wedge \xi_A\big(v'^{(l-1)}\big) \ne *\big).$$

Using Lemma 4.3 and Equation (13), we get:

$$\frac{\nabla}{4} = \mathfrak{C}\big((\log s)^{c-1} - 1\big) \le c \cdot \left(\frac{c}{c-1}\right)^c \cdot \sum_{l \in (\alpha,\beta]} \mathbb{1}\big(v'^{(l-1)} \in V_A' \wedge \xi_A\big(v'^{(l-1)}\big) \ne *\big).$$

Using Equation (13) again, this gives:

$$\frac{5}{4} \cdot \Theta \leq \sum_{l \in [\ell]} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge \xi_A\left(v'^{(l-1)}\right) \neq *\right),$$

a contradiction as $\xi_A \in (\Xi_\Theta(\Pi'))_A$. □

4.4 Proof of Lemma 4.12

We now prove Lemma 4.12.

Proof of Lemma 4.12. Proof by contradiction. Suppose that $\max(\mathtt{rew}_A(v'), \mathtt{rew}_B(v')) = \nabla$ and $R'_{\Theta,v'} \neq \emptyset$. As the proof is similar otherwise, we assume that $\mathtt{rew}_A(v') = \nabla$. Fix an arbitrary pair $((x, \xi_A), (y, \xi_B)) \in R'_{\Theta,v'}$, noting that such a pair must exist as $R'_{\Theta,v'} \neq \emptyset$. Applying Observation 3.2 repeatedly, we get that there exists an $\ell > 0$ and a sequence of vertices $\mathsf{rt}' = v'^{(0)}, v'^{(1)}, \ldots, v'^{(\ell)} = v'$ such that for all $l \in [\ell]$, we have $\left(v'^{(l-1)}, v'^{(l)}\right) \in E'$ and:

$$\exists\Big((x, \xi_A), \left(y^{(l-1)}, \xi_B^{(l-1)}\right)\Big) \in R'_{\Theta,v'^{(l-1)}} : \qquad h'_{\Theta,v'^{(l-1)}}\Big((x, \xi_A), \left(y^{(l-1)}, \xi_B^{(l-1)}\right)\Big) = v'^{(l)}. \quad (25)$$

We define $\left(y^{(\ell)}, \xi_B^{(\ell)}\right) = (y, \xi_B)$ for convenience. Henceforth, to simplify notation for the terms in Equation (15), we will abbreviate $\mathtt{v}\left(v'^{(l)}\right)$ to $\mathtt{v}^{(l)}$, $\mathtt{nxt}\left(v'^{(l)}\right)$ to $\mathtt{nxt}^{(l)}$, $\mathtt{cnf}\left(v'^{(l)}\right)$ to $\mathtt{cnf}^{(l)}$ *etc.* for all $0 \leq l \leq \ell$. We have from the definition of rt' in Equation (16) and Lemmas 4.7 and 4.8 that:

$$\nabla = \mathtt{rew}_A^{(\ell)} = \mathtt{rew}_A^{(\ell)} - \mathtt{rew}_A^{(0)} = \sum_{l=1}^{\ell} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge v'^{(l)} = v_r'^{(l-1)}\right). \quad (26)$$

In words, this means that the total number of rewinds from nodes in V'_A is ∇. The set defined next captures when these rewinds are legitimate, *i.e.* not a result of errors (see Equation (21)).

$$\mathsf{Back} = \left\{ l \in [\ell] \mid x \notin (R_{\mathtt{nxt}^{(l-1)}})_A \cap \bigcap_{v \in \mathtt{S}^{(l-1)}} (R_v)_A \right\}. \quad (27)$$

We now formalize the claim that any illegitimate rewinds must be due to errors. We also show that any forward edges in rounds where rewinds are legitimate must be due to errors.

Lemma 4.13. *For all* $l \in [\ell] \setminus \mathsf{Back}$*, we have that:*

$$\mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge v'^{(l)} = v_r'^{(l-1)}\right) \leq \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge \xi_A\left(v'^{(l-1)}\right) \neq *\right).$$

Proof. We have:

$$\mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge v'^{(l)} = v_r'^{(l-1)}\right)$$

$$= \mathbb{1}\Big(v'^{(l-1)} \in V'_A \wedge h'_{\Theta,v'^{(l-1)}}\Big((x,\xi_A),\Big(y^{(l-1)},\xi_B^{(l-1)}\Big)\Big) = v_r'^{(l-1)}\Big) \qquad \text{(Equation (25))}$$
$$\le \mathbb{1}\Big(v'^{(l-1)} \in V'_A \wedge h'_{\Theta,v'^{(l-1)}}\Big((x,\xi_A),\Big(y^{(l-1)},\xi_B^{(l-1)}\Big)\Big) \ne v'^{(l-1)}_{h_{\mathtt{v}^{(l-1)}}(x,y^{(l-1)})}\Big)$$
$$\le \mathbb{1}\Big(v'^{(l-1)} \in V'_A \wedge h'_{\Theta,v'^{(l-1)}}\Big((x,\xi_A),\Big(y^{(l-1)},\xi_B^{(l-1)}\Big)\Big) \ne h'_{v'^{(l-1)}}\big(x,y^{(l-1)}\big)\Big)$$
$$\text{(Equation (21) and } l \in [\ell] \setminus \mathsf{Back}\text{)}$$
$$\le \mathbb{1}\big(v'^{(l-1)} \in V'_A \wedge \xi_A\big(v'^{(l-1)}\big) \ne *\big). \qquad \text{(Equation (3))}$$

□

Lemma 4.14. *For all $l \in \mathsf{Back}$, we have that:*

$$\mathbb{1}\big(v'^{(l-1)} \in V'_A \wedge v'^{(l)} \ne v_r'^{(l-1)}\big) \le \mathbb{1}\big(v'^{(l-1)} \in V'_A \wedge \xi_A\big(v'^{(l-1)}\big) \ne *\big).$$

Proof. We have:

$$\mathbb{1}\big(v'^{(l-1)} \in V'_A \wedge v'^{(l)} \ne v_r'^{(l-1)}\big)$$
$$= \mathbb{1}\Big(v'^{(l-1)} \in V'_A \wedge h'_{\Theta,v'^{(l-1)}}\Big((x,\xi_A),\Big(y^{(l-1)},\xi_B^{(l-1)}\Big)\Big) \ne v_r'^{(l-1)}\Big) \qquad \text{(Equation (25))}$$
$$= \mathbb{1}\Big(v'^{(l-1)} \in V'_A \wedge h'_{\Theta,v'^{(l-1)}}\Big((x,\xi_A),\Big(y^{(l-1)},\xi_B^{(l-1)}\Big)\Big) \ne h'_{v'^{(l-1)}}\big(x,y^{(l-1)}\big)\Big)$$
$$\text{(Equation (21) and } l \in \mathsf{Back}\text{)}$$
$$= \mathbb{1}\big(v'^{(l-1)} \in V'_A \wedge \xi_A\big(v'^{(l-1)}\big) \ne *\big). \qquad \text{(Equation (3))}$$

□

Lemma 4.13 is all we need about $l \notin \mathsf{Back}$. For $l \in \mathsf{Back}$, we further analyze:

Claim 4.15. *For all $l \in \mathsf{Back}$, we have that:*

$$\mathbb{1}\big(\mathtt{d}\big(\mathtt{v}^{(l-1)}\big) < 2 \wedge \mathtt{cnf}^{(l-1)} = \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(l-1)}\big)\big)\big) = 0.$$

Proof. Suppose for the sake of contradiction that $l \in \mathsf{Back}$ and we have $\mathtt{d}\big(\mathtt{v}^{(l-1)}\big) < 2$ and $\mathtt{cnf}^{(l-1)} = \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(l-1)}\big)\big)$. From the former with Item 1 of Lemma 4.6, we get that $\mathtt{d}(v) < 2$ for all $v \in \mathtt{S}^{(l-1)}$. From Equation (10), this means that $(R_v)_A = \mathcal{X}$ for all $v \in \mathtt{S}^{(l-1)}$. Plugging into Equation (27), we get that $x \notin (R_{\mathtt{nxt}^{(l-1)}})_A$ implying that $\mathtt{nxt}^{(l-1)} \ne \mathsf{rt}$. Using Item 4 of Lemma 4.6, this gives a contradiction. □

Corollary 4.16 (Corollary of Claim 4.15 and Item 2 of Lemma 4.9)**.** *For all $l \in \mathsf{Back}$, we have that:*

$$\mathbb{1}\big(v'^{(l)} \ne v_r'^{(l-1)}\big) - \mathbb{1}\big(v'^{(l)} = v_r'^{(l-1)}\big) = \mathtt{cnf}^{(l)} - \mathtt{cnf}^{(l-1)}.$$

Next, we partition the set Back into disjoint intervals. Specifically, let $J \ge 0$ and

$$-1 = \beta_0 < \alpha_1 < \beta_1 < \cdots < \alpha_J < \beta_J < \alpha_{J+1} = \ell + 1, \tag{28}$$

be the unique values such that $\mathsf{Back} = \bigcup_{j\in[J]}(\alpha_j, \beta_j]$. The main lemma we need is the following:

Lemma 4.17. *For all $j \in [J]$, it holds that:*

1. *We have $\alpha_j > 0$ and $v'^{(\alpha_j-1)} \in V'_A$ and $\xi_A\big(v'^{(\alpha_j-1)}\big) \neq *$.*

2. *We have:*

$$\sum_{l=\alpha_j+1}^{\beta_j} \mathbb{1}\left(v'^{(l-1)} \in V'_A\right) \cdot \left(\mathtt{cnf}^{(l-1)} - \mathtt{cnf}^{(l)}\right)$$
$$\leq 3c \cdot \left(\frac{c}{c-1}\right)^c \cdot \left(1 + \sum_{l=\alpha_j+1}^{\beta_j} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge v'^{(l)} \neq v_r'^{(l-1)}\right)\right).$$

We defer the proof of Lemma 4.17 to later, but use it now to finish the proof of Lemma 4.12. We have:

$$\begin{aligned}
\nabla &\leq \sum_{l=1}^{\ell} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge v'^{(l)} = v_r'^{(l-1)}\right) && \text{(Equation (26))}\\
&\leq \sum_{l\in[\ell]\setminus\mathsf{Back}} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge v'^{(l)} = v_r'^{(l-1)}\right) + \sum_{l\in\mathsf{Back}} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge v'^{(l)} = v_r'^{(l-1)}\right)\\
&\leq \sum_{l\in[\ell]\setminus\mathsf{Back}} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge \xi_A\big(v'^{(l-1)}\big) \neq *\right) + \sum_{l\in\mathsf{Back}} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge v'^{(l)} = v_r'^{(l-1)}\right)\\
& && \text{(Lemma 4.13)}\\
&\leq \sum_{l\in[\ell]\setminus\mathsf{Back}} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge \xi_A\big(v'^{(l-1)}\big) \neq *\right) + \sum_{j=1}^{J}\sum_{l=\alpha_j+1}^{\beta_j} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge v'^{(l)} = v_r'^{(l-1)}\right)\\
&\leq \sum_{l\in[\ell]\setminus\mathsf{Back}} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge \xi_A\big(v'^{(l-1)}\big) \neq *\right)\\
&\qquad + \sum_{j=1}^{J}\sum_{l=\alpha_j+1}^{\beta_j} \mathbb{1}\left(v'^{(l-1)} \in V'_A\right) \cdot \left(\mathbb{1}\left(v'^{(l)} \neq v_r'^{(l-1)}\right) + \mathtt{cnf}^{(l-1)} - \mathtt{cnf}^{(l)}\right)\\
& && \text{(Corollary 4.16)}\\
&\leq \sum_{l\in[\ell]\setminus\mathsf{Back}} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge \xi_A\big(v'^{(l-1)}\big) \neq *\right)\\
&\qquad + 4c \cdot \left(\frac{c}{c-1}\right)^c \cdot \sum_{j=1}^{J}\sum_{l=\alpha_j+1}^{\beta_j} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge v'^{(l)} \neq v_r'^{(l-1)}\right)\\
&\qquad + 3c \cdot \left(\frac{c}{c-1}\right)^c \cdot \sum_{j=1}^{J} \mathbb{1}\left(v'^{(\alpha_j-1)} \in V'_A \wedge \xi_A\big(v'^{(\alpha_j-1)}\big) \neq *\right) && \text{(Lemma 4.17)}
\end{aligned}$$

$$\leq 4c \cdot \left(\frac{c}{c-1}\right)^c \cdot \sum_{l \in [\ell] \setminus \mathsf{Back}} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge \xi_A\left(v'^{(l-1)}\right) \neq *\right)$$
$$+ 4c \cdot \left(\frac{c}{c-1}\right)^c \cdot \sum_{l \in \mathsf{Back}} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge v'^{(l)} \neq v'^{(l-1)}_r\right)$$
$$\leq 4c \cdot \left(\frac{c}{c-1}\right)^c \cdot \sum_{l=1}^{\ell} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge \xi_A\left(v'^{(l-1)}\right) \neq *\right). \qquad \text{(Lemma 4.14)}$$

This means that:
$$\Theta \leq \frac{4}{5} \cdot \sum_{l \in [\ell]} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge \xi_A\left(v'^{(l-1)}\right) \neq *\right),$$
a contradiction as $\xi_A \in (\Xi_\Theta(\Pi'))_A$.

□

4.5 Proof of Lemma 4.17

We now prove Lemma 4.17.

Proof of Lemma 4.17. Fix an arbitrary $j \in [J]$. We first claim that $\alpha_j > 0$. Indeed, if $\alpha_j = 0$, then $\alpha_j + 1 \in \mathsf{Back}$ would imply by Equation (27) that $x \notin (R_{\mathtt{nxt}^{(0)}})_A \cap \bigcap_{v \in \mathtt{S}^{(0)}} (R_v)_A$, a contradiction as $v'^{(0)} = \mathsf{rt}' = \left(\mathsf{rt}, \mathsf{rt}, ((\mathsf{rt}, \mathsf{rt}))_{i \in [c]}, (0,0,0)\right)$ by Equation (16). Because $\alpha_j > 0 \notin \mathsf{Back}$ and $\alpha_j + 1 \in \mathsf{Back}$, we have from Equation (27) that:

$$x \in \left(\left(R_{\mathtt{nxt}^{(\alpha_j - 1)}}\right)_A \cap \bigcap_{v \in \mathtt{S}^{(\alpha_j - 1)}} (R_v)_A \right) \setminus \left(\left(R_{\mathtt{nxt}^{(\alpha_j)}}\right)_A \cap \bigcap_{v \in \mathtt{S}^{(\alpha_j)}} (R_v)_A \right). \tag{29}$$

Now, we use Equation (29) to show some useful properties about α_j, that in particular imply Item 1 of Lemma 4.17.

Claim 4.18. *It holds that* $\mathtt{cnf}^{(\alpha_j - 1)} = \mathtt{cnf}^{(\alpha_j)} - 1 = \mathfrak{C}\left(\mathtt{d}\left(\mathtt{v}^{(\alpha_j - 1)}\right)\right)$ *and* $\mathtt{v}^{(\alpha_j)} = \mathtt{v}^{(\alpha_j - 1)} \in V_A$ *and* $\log s \nmid \mathtt{d}\left(\mathtt{v}^{(\alpha_j)}\right)$ *and* $v'^{(\alpha_j - 1)} \in V'_A$ *and* $\xi_A\left(v'^{(\alpha_j - 1)}\right) \neq *$.

Proof. Together with Item 4 of Lemma 4.9, Equation (29) implies that $x \in \left(R_{\mathtt{nxt}^{(\alpha_j - 1)}}\right)_A \setminus \left(R_{\mathtt{nxt}^{(\alpha_j)}}\right)_A$. It follows that $\mathtt{nxt}^{(\alpha_j)} \notin \left\{\mathtt{nxt}^{(\alpha_j - 1)}, \mathsf{rt}\right\}$. By Equations (17) to (20), this is possible only if $\mathtt{cnf}^{(\alpha_j - 1)} = \mathtt{cnf}^{(\alpha_j)} - 1 = \mathfrak{C}\left(\mathtt{d}\left(\mathtt{v}^{(\alpha_j - 1)}\right)\right)$. This also means that $\mathtt{v}^{(\alpha_j)} = \mathtt{v}^{(\alpha_j - 1)}$ and $\left(\mathtt{v}^{(\alpha_j)}, \mathtt{nxt}^{(\alpha_j)}\right) \in E$ is an edge in the original graph G. Using Equation (29) again, this means that $x \in \left(R_{\mathtt{v}^{(\alpha_j)}}\right)_A \setminus \left(R_{\mathtt{nxt}^{(\alpha_j)}}\right)_A$. Because of Definition 3.1 and Equation (8), this is only possible if $\mathtt{v}^{(\alpha_j - 1)} \in V_A \implies v'^{(\alpha_j - 1)} \in V'_A$ and $\mathtt{nxt}^{(\alpha_j)} \neq h_{\mathtt{v}^{(\alpha_j - 1)}}\left(x, y^{(\alpha_j - 1)}\right)$. From the latter, we have $\log s \nmid \mathtt{d}\left(\mathtt{v}^{(\alpha_j - 1)}\right)$ and:

$$\mathtt{nxt}^{(\alpha_j)} \neq h_{\mathtt{v}^{(\alpha_j - 1)}}\left(x, y^{(\alpha_j - 1)}\right) \implies v'^{(\alpha_j)} = v'^{(\alpha_j - 1)}_{\mathtt{nxt}^{(\alpha_j)}} \neq v'^{(\alpha_j - 1)}_{h_{\mathtt{v}^{(\alpha_j - 1)}}\left(x, y^{(\alpha_j - 1)}\right)} \qquad \text{(Equation (17))}$$

$$\implies v'^{(\alpha_j)} \neq h'_{v'^{(\alpha_j-1)}}\big(x, y^{(\alpha_j-1)}\big) \qquad \text{(Equations (21) and (29))}$$

$$\implies h'_{\Theta, v'^{(\alpha_j-1)}}\Big((x,\xi_A), \Big(y^{(\alpha_j-1)}, \xi_B^{(\alpha_j-1)}\Big)\Big) \neq h'_{v'^{(\alpha_j-1)}}\big(x, y^{(\alpha_j-1)}\big)$$
$$\text{(Equation (25))}$$

$$\implies \xi_A\big(v'^{(\alpha_j-1)}\big) \neq *. \qquad \text{(Equation (3) and } v'^{(\alpha_j-1)} \in V'_A\text{)}$$

□

It now remains to prove Item 2 of Lemma 4.17. Using Claim 4.18 (which implies that $v'^{(\alpha_j)} \neq v_r'^{(\alpha_j-1)}$ by Item 1 of Lemma 4.8), this follows if we show:

$$\sum_{l=\alpha_j+1}^{\beta_j} \mathbb{1}\big(v'^{(l-1)} \in V'_A\big) \cdot \big(\mathtt{cnf}^{(l-1)} - \mathtt{cnf}^{(l)}\big)$$
$$\leq 3c \cdot \left(\frac{c}{c-1}\right)^c \cdot \sum_{l=\alpha_j}^{\beta_j} \mathbb{1}\big(v'^{(l-1)} \in V'_A \wedge v'^{(l)} \neq v_r'^{(l-1)}\big). \quad (30)$$

We show Equation (30) by showing Claims 4.21 and 4.22 below, which when added together imply Equation (30). However, we first prove two helper claims:

Claim 4.19. *For all $l \in [\alpha_j, \beta_j)$, the following hold:*

1. *For all $v \in \mathtt{S}^{(l)}$ such that $\mathtt{d}(v) \leq \mathtt{d}\big(\mathtt{v}^{(\alpha_j)}\big)$, we have $v \in \mathtt{S}^{(\alpha_j-1)}$.*

2. *We have $\mathtt{d}\big(\mathtt{v}^{(\alpha_j)}\big) \leq \mathtt{d}\big(\mathtt{v}^{(l)}\big)$.*

3. *If $\mathtt{d}\big(\mathtt{v}^{(\alpha_j)}\big) = \mathtt{d}\big(\mathtt{v}^{(l)}\big)$, then we have $\mathtt{cnf}^{(\alpha_j)} \leq \mathtt{cnf}^{(l)}$.*

Proof. Proof by induction. For the base case $l = \alpha_j$, Items 2 and 3 are straightforward. Item 1 follows from Claim 4.18 and Item 2 of Lemma 4.7. We now prove the lemma for $l \in (\alpha_j, \beta_j)$ assuming it holds for $l-1$. For Item 1, because of the induction hypothesis, we can assume that $v \in \mathtt{S}^{(l)} \setminus \mathtt{S}^{(l-1)}$. By Item 3 of Lemma 4.8 and Item 2 of Lemma 4.7, this means that $v = \mathtt{v}^{(l)}$ and $\mathtt{d}(v) = \mathtt{d}\big(\mathtt{v}^{(l)}\big) = \mathtt{d}\big(\mathtt{v}^{(l-1)}\big) + 1 > \mathtt{d}\big(\mathtt{v}^{(\alpha_j)}\big)$ by the induction hypothesis, and the statement is vacuously true. For Items 2 and 3, note that $\mathsf{Back} = \bigcup_{j \in [J]} (\alpha_j, \beta_j]$ implies from Equation (27) that:

$$x \in \left(\big(R_{\mathtt{nxt}^{(\alpha_j-1)}}\big)_A \cap \bigcap_{v \in \mathtt{S}^{(\alpha_j-1)}} (R_v)_A \right) \setminus \left(\big(R_{\mathtt{nxt}^{(l)}}\big)_A \cap \bigcap_{v \in \mathtt{S}^{(l)}} (R_v)_A \right).$$

This implies that either $\mathtt{nxt}^{(l)} \neq \mathsf{rt}$ or $\mathtt{S}^{(l)} \not\subseteq \mathtt{S}^{(\alpha_j-1)}$. If the former holds, Item 3 follows from Item 4 of Lemma 4.6 as we get $\mathtt{cnf}^{(l)} > \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(l)}\big)\big) = \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(\alpha_j)}\big)\big) = \mathtt{cnf}^{(\alpha_j)} - 1$, where we use Claim 4.18 in the last step. Item 2 also follows as $\mathtt{cnf}^{(l)} > \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(l)}\big)\big)$ implies by Item 3 of Lemma 4.9 that $\mathtt{d}\big(\mathtt{v}^{(l)}\big) = \mathtt{d}\big(\mathtt{v}^{(l-1)}\big)$ and we can get Item 2 using the induction hypothesis.

If the latter holds, Item 1 says we have $v \in \mathtt{S}^{(l)}$ such that $\mathtt{d}(v) > \mathtt{d}\big(\mathtt{v}^{(\alpha_j)}\big)$. This implies $\mathtt{d}\big(\mathtt{v}^{(l)}\big) > \mathtt{d}\big(\mathtt{v}^{(\alpha_j)}\big)$ using Item 1 of Lemma 4.6 and Items 2 and 3 follow. □

Claim 4.20. *Let $l'' \in [\ell]$ satisfy $\mathtt{d}\big(\mathtt{v}^{(l'')}\big) = \mathtt{d}\big(\mathtt{v}^{(l''-1)}\big) + 1$ is odd and $0 \le l' \le l''$ be such that $\mathtt{cnf}^{(l')} \le \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(l''-1)}\big)\big)$. We have:*

$$\mathfrak{C}\Big(\mathtt{d}\Big(\mathtt{v}^{(l'')}\Big)\Big) - \mathfrak{C}\Big(\mathtt{d}\Big(\mathtt{v}^{(l'')}\Big) - 1\Big) \le \sum_{l\in(l',l'']} \mathbb{1}\Big(v'^{(l-1)} \in V'_A \wedge v'^{(l)} \neq v_r'^{(l-1)}\Big).$$

An analogous result with Alice replaced by Bob holds if $\mathtt{d}\big(\mathtt{v}^{(l'')}\big)$ is even.

Proof. As it only makes the lemma stronger, we can assume without loss of generality that $0 \le l' \le l''$ is the largest such that $\mathtt{cnf}^{(l')} \le \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(l''-1)}\big)\big)$. We claim that for all $l' < l \le l''$, if $\mathtt{d}\big(\mathtt{v}^{(l)}\big) = \mathtt{d}\big(\mathtt{v}^{(l''-1)}\big)$, then we also have $\mathtt{d}\big(\mathtt{v}^{(l-1)}\big) = \mathtt{d}\big(\mathtt{v}^{(l''-1)}\big)$. Indeed, if not, we have $\mathtt{d}\big(\mathtt{v}^{(l)}\big) = \mathtt{d}\big(\mathtt{v}^{(l''-1)}\big) \neq \mathtt{d}\big(\mathtt{v}^{(l-1)}\big)$. From Item 3 of Lemma 4.9, this means that $\mathtt{cnf}^{(l)} = \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(l''-1)}\big)\big)$, a contradiction to the choice of l'. From this claim, it follows that for all $l' < l \le l''$, we have $\mathtt{d}\big(\mathtt{v}^{(l-1)}\big) = \mathtt{d}\big(\mathtt{v}^{(l''-1)}\big)$ implying (as $\mathtt{d}\big(\mathtt{v}^{(l''-1)}\big)$ is even) that $\mathtt{v}^{(l-1)} \in V_A$ and $v'^{(l-1)} \in V'_A$. This gives:

$$\sum_{l\in(l',l'']} \mathbb{1}\Big(v'^{(l-1)} \in V'_A \wedge v'^{(l)} \neq v_r'^{(l-1)}\Big) = \sum_{l\in(l',l'']} \mathbb{1}\Big(v'^{(l)} \neq v_r'^{(l-1)}\Big).$$

Thus, it suffices to bound $\mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(l'')}\big)\big) - \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(l''-1)}\big)\big)$ by the right hand side above. For this, note that as $\mathtt{d}\big(\mathtt{v}^{(l'')}\big) = \mathtt{d}\big(\mathtt{v}^{(l''-1)}\big) + 1$, we have from Item 1 of Lemma 4.6 that $\mathtt{v}^{(l'')} \notin \mathtt{S}^{(l''-1)}$. From Item 3 of Lemma 4.8 and Item 2 of Lemma 4.7, we get that $\mathtt{cnf}^{(l'')} = \mathtt{cnf}^{(l''-1)} + 1 = \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(l'')}\big)\big)$. This gives the desired inequality as follows:

$$\begin{aligned}
\mathfrak{C}\Big(\mathtt{d}\Big(\mathtt{v}^{(l'')}\Big)\Big) - \mathfrak{C}\Big(\mathtt{d}\Big(\mathtt{v}^{(l'')}\Big) - 1\Big) &= \mathfrak{C}\Big(\mathtt{d}\Big(\mathtt{v}^{(l'')}\Big)\Big) - \mathfrak{C}\Big(\mathtt{d}\Big(\mathtt{v}^{(l''-1)}\Big)\Big) \\
&\le \mathtt{cnf}^{(l'')} - \mathtt{cnf}^{(l')} \\
&= \sum_{l\in(l',l'']} \Big(\mathtt{cnf}^{(l)} - \mathtt{cnf}^{(l-1)}\Big) \\
&\le \sum_{l\in(l',l'']} \mathbb{1}\Big(\mathtt{cnf}^{(l)} - \mathtt{cnf}^{(l-1)} = 1\Big) \\
&\le \sum_{l\in(l',l'']} \mathbb{1}\Big(v'^{(l)} \neq v_r'^{(l-1)}\Big). && \text{(Lemma 4.8, Item 1)}
\end{aligned}$$

□

Claim 4.21. *We have:*

$$\mathtt{cnf}^{(\alpha_j)} - \mathtt{cnf}^{(\beta_j)} \le 2c \cdot \left(\frac{c}{c-1}\right)^c \cdot \sum_{l=\alpha_j}^{\beta_j} \mathbb{1}\Big(v'^{(l-1)} \in V'_A \wedge v'^{(l)} \neq v_r'^{(l-1)}\Big).$$

Proof. Because of Item 2 of Lemma 4.9, it suffices to show that $\mathtt{cnf}^{(\alpha_j)} - \mathtt{cnf}^{(\beta_j-1)}$ is at most half of the second term on the right hand side. This is trivial if $\mathtt{cnf}^{(\alpha_j)} \leq \mathtt{cnf}^{(\beta_j-1)}$, so we assume without loss of generality that $\mathtt{cnf}^{(\beta_j-1)} < \mathtt{cnf}^{(\alpha_j)}$. It follows from Items 2 and 3 of Claim 4.19 that $\mathtt{d}\big(\mathtt{v}^{(\alpha_j)}\big) < \mathtt{d}\big(\mathtt{v}^{(\beta_j-1)}\big)$. As $\mathtt{v}^{(\alpha_j-1)} \in V_A$ by Claim 4.18 implying that $\mathtt{d}\big(\mathtt{v}^{(\alpha_j-1)}\big)$ is even, this means that there must exists a pair $(l', j') \in (\alpha_j, \beta_j) \times [0, c)$ such that $\mathtt{d}\big(\mathtt{v}^{(l')}\big) = \mathtt{d}\big(\mathtt{v}^{(l'-1)}\big) + 1$ is odd and $(\log s)^{j'} \mid \mathtt{d}\big(\mathtt{v}^{(l'-1)}\big) + 2$. Let (l^*, j^*) be the maximizer of j' over all such pairs, breaking ties arbitrarily.

We claim that $\mathtt{d}\Big(\mathtt{v}^{(\beta_j-1)}_{j^*+1,A}\Big) = \mathtt{d}\Big(\mathtt{v}^{(\alpha_j)}_{j^*+1,A}\Big)$. Indeed, assume it is not, and let $l \in (\alpha_j, \beta_j)$ be the smallest such that $\mathtt{d}\Big(\mathtt{v}^{(l)}_{j^*+1,A}\Big) \neq \mathtt{d}\Big(\mathtt{v}^{(\alpha_j)}_{j^*+1,A}\Big)$. This is well defined as $l = \beta_j - 1$ is one such value by assumption. By choice of l, we have that $\mathtt{d}\Big(\mathtt{v}^{(l)}_{j^*+1,A}\Big) \neq \mathtt{d}\Big(\mathtt{v}^{(l-1)}_{j^*+1,A}\Big) = \mathtt{d}\Big(\mathtt{v}^{(\alpha_j)}_{j^*+1,A}\Big)$. From Item 2 of Claim 4.19 and Item 1 of Lemma 4.6, we get that $\left\lfloor \frac{\mathtt{d}(\mathtt{v}^{(l)})}{(\log s)^{j^*+1}} \right\rfloor > \left\lfloor \frac{\mathtt{d}(\mathtt{v}^{(l-1)})}{(\log s)^{j^*+1}} \right\rfloor = \left\lfloor \frac{\mathtt{d}(\mathtt{v}^{(\alpha_j)})}{(\log s)^{j^*+1}} \right\rfloor$. From Item 3 of Lemma 4.9, this implies that $(\log s)^{j^*+1} \mid \mathtt{d}\big(\mathtt{v}^{(l)}\big)$. From Claim 4.18, we get that $\mathtt{d}\big(\mathtt{v}^{(\alpha_j)}\big) + 2 \leq \mathtt{d}\big(\mathtt{v}^{(l)}\big)$. Using Item 3 of Lemma 4.9 again, this contradicts the choice of (l^*, j^*).

Having shown that $\mathtt{d}\Big(\mathtt{v}^{(\beta_j-1)}_{j^*+1,A}\Big) = \mathtt{d}\Big(\mathtt{v}^{(\alpha_j)}_{j^*+1,A}\Big)$, we combine it with Claim 4.18 and Item 1 of Lemma 4.6 to get that $\mathtt{d}\Big(\mathtt{v}^{(\beta_j-1)}_{j^*+1,A}\Big) = \mathtt{d}\Big(\mathtt{v}^{(\alpha_j)}_{j^*+1,A}\Big) < \mathtt{d}\big(\mathtt{v}^{(\alpha_j)}\big) < \mathtt{d}\big(\mathtt{v}^{(\beta_j-1)}\big)$. We emphasize that the first inequality is strict. Using the fact that these are all integers, we deduce that $\mathtt{d}\Big(\mathtt{v}^{(\beta_j-1)}_{j^*+1,A}\Big) + 1 < \mathtt{d}\big(\mathtt{v}^{(\beta_j-1)}\big)$. We derive:

$$\begin{aligned}
\mathtt{cnf}^{(\alpha_j)} - \mathtt{cnf}^{(\beta_j-1)} &\leq \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(\alpha_j)}\big)\big) - \mathtt{cnf}^{(\beta_j-1)} + 1 && \text{(Claim 4.18)}\\
&\leq \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(\alpha_j)}\big)\big) - \mathfrak{C}\Big(\mathtt{d}\Big(\mathtt{v}^{(\alpha_j)}_{j^*+1,A}\Big)\Big) && \text{(Lemma 4.6, Item 3)}\\
&\leq \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(l^*)}\big)\big) - \mathfrak{C}\Big(\mathtt{d}\Big(\mathtt{v}^{(\alpha_j)}_{j^*+1,A}\Big)\Big) && \text{(Claim 4.19, Item 2)}\\
&\leq \Big(\mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(l^*)}\big) + 1\big) - \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(l^*)}\big)\big)\Big) \cdot \left(\frac{c}{c-1}\right)^c && \text{(Lemma 4.3)}\\
&\leq \Big(\mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(l^*)}\big)\big) - \mathfrak{C}\big(\mathtt{d}\big(\mathtt{v}^{(l^*)}\big) - 1\big)\Big) \cdot c \cdot \left(\frac{c}{c-1}\right)^c \\
& && \text{(Equation (12) and the fact that } \mathtt{d}\big(\mathtt{v}^{(l^*)}\big) > 0)\\
&\leq c \cdot \left(\frac{c}{c-1}\right)^c \cdot \sum_{l \in [\alpha_j, l^*]} \mathbb{1}\big(v'^{(l-1)} \in V'_A \wedge v'^{(l)} \neq v_r'^{(l-1)}\big) && \text{(Claim 4.20)}\\
&\leq c \cdot \left(\frac{c}{c-1}\right)^c \cdot \sum_{l=\alpha_j}^{\beta_j} \mathbb{1}\big(v'^{(l-1)} \in V'_A \wedge v'^{(l)} \neq v_r'^{(l-1)}\big).
\end{aligned}$$

□

Claim 4.22. *We have:*

$$\sum_{l=\alpha_j+1}^{\beta_j} \mathbb{1}\left(v'^{(l-1)} \in V'_B\right) \cdot \left(\mathtt{cnf}^{(l)} - \mathtt{cnf}^{(l-1)}\right) \leq c \cdot \sum_{l=\alpha_j}^{\beta_j} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge v'^{(l)} \neq v'^{(l-1)}_r\right).$$

Proof. To start, define the set $\mathsf{Bob} = \left\{l \in (\alpha_j, \beta_j] \mid v'^{(l-1)} \in V'_B\right\}$. We partition the set Bob into disjoint intervals as follows. Let $Z \geq 0$ and:

$$\alpha_j - 1 = \nu_0 < \mu_1 < \nu_1 < \cdots < \mu_Z < \nu_Z < \mu_{Z+1} = \beta_j + 1, \tag{31}$$

be such that $\mathsf{Bob} = \bigcup_{z\in[Z]}(\mu_z, \nu_z]$. We get that for all $z \in [Z]$, we have $\mu_z + 1 \in \mathsf{Bob}$ and $\mu_z \notin \mathsf{Bob}$. For $z > 1 \in [Z]$, we also get $\nu_{z-1}+1 \notin \mathsf{Bob}$ and $\nu_{z-1} \in \mathsf{Bob}$. Using $\alpha_j < \mu_1$ (which follows from Claim 4.18), we get that for all $z \in [Z]$, we have $v'^{(\mu_z)} \in V'_B$ and $v'^{(\mu_z-1)} \notin V'_B$ and if $z > 1$, also have $v'^{(\nu_{z-1}-1)} \in V'_B$ and $v'^{(\nu_{z-1})} \notin V'_B$.

From Equations (17) to (20) and Item 2 of Lemma 4.6, we get that this implies that for all $z \in [Z]$, we have that $\mathfrak{C}\left(\mathtt{d}\left(\mathtt{v}^{(\mu_z)}\right)\right) = \mathtt{cnf}^{(\mu_z)}$ and $\mathfrak{C}\left(\mathtt{d}\left(\mathtt{v}^{(\nu_{z-1})}\right)\right) = \mathtt{cnf}^{(\nu_{z-1})}$, where the latter for $z = 1$ follows from Claim 4.18. We also get that that $\mathtt{d}\left(\mathtt{v}^{(\mu_z)}\right), \mathtt{d}\left(\mathtt{v}^{(\nu_{z-1})}\right) > 0$ for all $z \in [Z]$. As the protocol Π is alternating and as we have Item 3 of Lemma 4.9, for all $z \in [Z]$, we get that $\mathtt{d}\left(\mathtt{v}^{(\nu_z-1)}\right) \leq \mathtt{d}\left(\mathtt{v}^{(\mu_z)}\right) = \mathtt{d}\left(\mathtt{v}^{(\mu_z-1)}\right) + 1 \leq \mathtt{d}\left(\mathtt{v}^{(\nu_{z-1})}\right) + 1$. This gives:

$$\begin{aligned}
\sum_{l=\alpha_j+1}^{\beta_j} \mathbb{1}\left(v'^{(l-1)} \in V'_B\right) \cdot \left(\mathtt{cnf}^{(l)} - \mathtt{cnf}^{(l-1)}\right) &= \sum_{z=1}^{Z} \sum_{l\in(\mu_z,\nu_z]} \left(\mathtt{cnf}^{(l)} - \mathtt{cnf}^{(l-1)}\right) \\
&= \sum_{z=1}^{Z} \left(\mathtt{cnf}^{(\nu_z)} - \mathtt{cnf}^{(\mu_z)}\right) \\
&\leq \sum_{z=1}^{Z} \left(\mathtt{cnf}^{(\nu_z-1)} + 1 - \mathfrak{C}\left(\mathtt{d}\left(\mathtt{v}^{(\mu_z)}\right)\right)\right) \\
&\leq \sum_{z=1}^{Z} \left(\mathfrak{C}\left(\mathtt{d}\left(\mathtt{v}^{(\mu_z)}\right) + 1\right) - \mathfrak{C}\left(\mathtt{d}\left(\mathtt{v}^{(\mu_z)}\right)\right)\right) \\
&\qquad \text{(Lemma 4.6, Item 2)} \\
&\leq c \cdot \sum_{z=1}^{Z} \left(\mathfrak{C}\left(\mathtt{d}\left(\mathtt{v}^{(\mu_z)}\right)\right) - \mathfrak{C}\left(\mathtt{d}\left(\mathtt{v}^{(\mu_z)}\right) - 1\right)\right) \\
&\qquad \text{(Equation (12) and the fact that } \mathtt{d}\left(\mathtt{v}^{(\mu_z)}\right) > 0\text{)} \\
&\leq c \cdot \sum_{z=1}^{Z} \sum_{l\in(\nu_{z-1},\mu_z]} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge v'^{(l)} \neq v'^{(l-1)}_r\right) \\
&\qquad \text{(Claim 4.20 and the fact that } \mathtt{d}\left(\mathtt{v}^{(\mu_z-1)}\right) \leq \mathtt{d}\left(\mathtt{v}^{(\nu_{z-1})}\right)\text{)} \\
&\leq c \cdot \sum_{l=\alpha_j}^{\beta_j} \mathbb{1}\left(v'^{(l-1)} \in V'_A \wedge v'^{(l)} \neq v'^{(l-1)}_r\right).
\end{aligned}$$

□

□

References

[BB94] Maria Luisa Bonet and Samuel R. Buss. Size-depth tradeoffs for boolean fomulae. *Information Processing Letters*, 49(3):151–155, 1994. 4

[BCD$^+$23a] Mirza Ahad Baig, Suvradip Chakraborty, Stefan Dziembowski, Małgorzata Gałązka, Tomasz Lizurej, and Krzysztof Pietrzak. Efficiently testable circuits. In *14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*, 2023. 3

[BCD$^+$23b] Mirza Ahad Baig, Suvradip Chakraborty, Stefan Dziembowski, Małgorzata Gałązka, Tomasz Lizurej, and Krzysztof Pietrzak. Efficiently testable circuits without conductivity. In *Theory of Cryptography Conference*, 2023. 3

[BEGY19] Mark Braverman, Klim Efremenko, Ran Gelles, and Michael A. Yitayew. Optimal short-circuit resilient formulas. In *Computational Complexity Conference (CCC)*, volume 137, pages 10:1–10:22, 2019. 1, 2, 3, 5

[CLPS20] T.-H. Hubert Chan, Zhibin Liang, Antigoni Polychroniadou, and Elaine Shi. Small memory robust simulation of client-server interactive protocols over oblivious noisy channels. In *Symposium on Discrete Algorithms (SODA)*, pages 2349–2365, 2020. 4

[DO77] Roland L'vovich Dobrushin and SI Ortyukov. Upper bound on the redundancy of self-correcting arrangements of unreliable functional elements. *Problemy Peredachi Informatsii*, 13(3):56–76, 1977. 3

[EHK$^+$22] Klim Efremenko, Bernhard Haeupler, Yael Tauman Kalai, Pritish Kamath, Gillat Kol, Nicolas Resch, and Raghuvansh R. Saxena. Circuits resilient to short-circuit errors. In *54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, 2022. 1, 2, 3, 4, 5, 6, 10, 12, 15

[EP98] William S. Evans and Nicholas Pippenger. On the maximum tolerable noise for reliable computation by formulas. *IEEE Transactions on Information Theory*, 44(3):1299–1305, 1998. 3

[ES99] William S. Evans and Leonard J. Schulman. Signal propagation and noisy circuits. *IEEE Transactions on Information Theory*, 45(7):2367–2373, 1999. 3

[ES03] William S. Evans and Leonard J. Schulman. On the maximum tolerable noise of k-input gates for reliable computation by formulas. *IEEE Transactions on Information Theory*, 49:3094–3098, 2003. 3

[Fed89] Tomás Feder. Reliable computation by networks in the presence of noise. *IEEE Transactions on Information Theory*, 35(3):569–571, 1989. 3

[Gál91] Anna Gál. Lower bounds for the complexity of reliable boolean circuits with noisy gates. In *Foundations of Computer Science (FOCS)*, pages 594–601, 1991. 3

[Gel17] Ran Gelles. Coding for interactive communication: A survey. *Foundations and Trends in Theoretical Computer Science*, 13(1–2):1–157, 2017. 4

[GG94] Péter Gács and Anna Gál. Lower bounds for the complexity of reliable boolean circuits with noisy gates. *IEEE Transactions on Information Theory*, 40(2):579–583, 1994. 3

[GGKS18] Ankit Garg, Mika Göös, Pritish Kamath, and Dmitry Sokolov. Monotone circuit lower bounds from resolution. In *Symposium on Theory of Computing (STOC)*, pages 902–911, 2018. 4

[GS95] Anna Gál and Mario Szegedy. Fault tolerant circuits and probabilistically checkable proofs. In *Proceedings of Structure in Complexity Theory. Tenth Annual IEEE Conference*, pages 65–73, 1995. 3

[HW91] Bruce E. Hajek and Timothy Weller. On the maximum tolerable noise for reliable computation by formulas. *IEEE Transactions on Information Theory*, 37(2):388–391, 1991. 3

[KLM97] Daniel J. Kleitman, Frank Thomson Leighton, and Yuan Ma. On the design of reliable boolean circuits that contain partially unreliable gates. *Journal of Computer and System Sciences*, 55(3):385–401, 1997. 1, 3

[KLR12] Yael Tauman Kalai, Allison B. Lewko, and Anup Rao. Formulas resilient to short-circuit errors. In *Foundations of Computer Science (FOCS)*, pages 490–499, 2012. 1, 2, 3, 4, 5

[Kra97] Jan Krajícek. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *Journal of Symbolic Logic*, 62(2):457–486, 1997. 4

[KW88] Mauricio Karchmer and Avi Wigderson. Monotone circuits for connectivity require super-logarithmic depth. In *Symposium on Theory of Computing (STOC)*, pages 539–550, 1988. 2, 4, 5

[Pip85] Nicholas Pippenger. On networks of noisy gates. In *Foundations of Computer Science (FOCS)*, pages 30–38, 1985. 3

[Pip88] Nicholas Pippenger. Reliable computation by formulas in the presence of noise. *IEEE Transactions on Information Theory*, 34(2):194–197, 1988. 3

[Pud10] Pavel Pudlák. On extracting computations from propositional proofs (a survey). In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8, pages 30–41, 2010. 4

[Raz95] Alexander Razborov. Unprovability of lower bounds on circuit size in certain fragments of bounded arithmetic. *Izvestiya of the RAN*, pages 201–224, 1995. 2, 3, 4, 12

[Sch92] Leonard J Schulman. Communication on noisy channels: A coding theorem for computation. In *Foundations of Computer Science (FOCS)*, pages 724–733. IEEE, 1992. 3, 5

[Sch93] Leonard J. Schulman. Deterministic coding for interactive communication. In *Symposium on Theory of Computing (STOC)*, pages 747–756, 1993. 3, 5

[Sch96] Leonard J Schulman. Coding for interactive communication. *IEEE Transactions on Information Theory*, 42(6):1745–1756, 1996. 3, 5

[Sok17] Dmitry Sokolov. Dag-like communication and its applications. In *Proceedings of the 12th Computer Science Symposium in Russia (CSR)*, pages 294–307. Springer, 2017. 2, 3, 4, 12

[vN56a] John von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies*, pages 43–98, 1956. 1

[vN56b] John von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies*, pages 43–98, 1956. 3

A Trimming DAG-protocols

The goal of this appendix to define a notion of empty edges for a DAG-protocol and show that they can be trimmed away, without affecting the DAG-protocol. We start with an easy technical lemma, showing that the rectangle at a given node does not depend on the errors after the node.

Lemma A.1. *Let Π be a* DAG*-protocol with inputs sets $\mathcal{X}$, $\mathcal{Y}$, and output set $\mathcal{O}$, and let Ξ be a rectangular set of error patterns for Π. Let $w \in V$, $x \in \mathcal{X}$, and $\xi_A, \xi'_A \in \Xi_A$ be error* *patterns that only differ on vertices $v \in V_A$ that are reachable from w. Then, if $\{R_{\Xi,v}\}_{v \in V}$ be the rectangles associated with Π_Ξ according to Definition 3.1, we have:*

$$(x, \xi_A) \in (R_{\Xi,w})_A \iff (x, \xi'_A) \in (R_{\Xi,w})_A.$$

An analogous statement holds with the roles of Alice and Bob reversed.

Proof. We show that $\implies$ direction as the other direction is symmetric. Assume that $(x, \xi_A) \in (R_{\Xi,w})_A$. This means that there exists a pair (y, ξ_B) such that $((x, \xi_A), (y, \xi_B)) \in R_{\Xi,w}$. Applying Observation 3.2 repeatedly, we get that there exists an $\ell \geq 0$ and a sequence of vertices $\mathsf{rt} = w^{(0)}, w^{(1)}, \dots, w^{(\ell)} = w$ such that for all $l \in [\ell]$, we have $\left(w^{(l-1)}, w^{(l)}\right) \in E$ and:

$$\exists \Big((x, \xi_A), \left(y^{(l-1)}, \xi_B^{(l-1)}\right)\Big) \in R_{\Xi,w^{(l-1)}}: \qquad h_{\Xi,w^{(l-1)}}\Big((x, \xi_A), \left(y^{(l-1)}, \xi_B^{(l-1)}\right)\Big) = w^{(l)}. \quad (32)$$

We define $\left(y^{(\ell)}, \xi_B^{(\ell)}\right) = (y, \xi_B)$ for convenience. To prove the lemma, we prove by induction that, for all $0 \leq l \leq \ell$, we have $\Big((x, \xi'_A), \left(y^{(l)}, \xi_B^{(l)}\right)\Big) \in R_{\Xi,w^{(l)}}$. The lemma then follows by setting $l = \ell$. For the base case $l = 0$, this is because $R_{\Xi,w^{(0)}} = (\mathcal{X} \times \Xi_A) \times (\mathcal{Y} \times \Xi_B)$ by Definition 3.1. We show the result for $l > 0$ assuming it holds for $l - 1$. By our induction hypothesis, we have $\Big((x, \xi'_A), \left(y^{(l-1)}, \xi_B^{(l-1)}\right)\Big) \in R_{\Xi,w^{(l-1)}}$. We now claim that $h_{\Xi,w^{(l-1)}}\Big((x, \xi'_A), \left(y^{(l-1)}, \xi_B^{(l-1)}\right)\Big) = w^{(l)}$. If $w^{(l-1)} \in V_B$, this follows from Equation (32) and the fact that the message function for nodes in V_B only depends on the second argument. Otherwise, it is because of Equation (3) and the fact that ξ_A and ξ'_A match on $w^{(l-1)}$ as $w^{(l-1)}$ is not reachable from w. It follows that $\Big((x, \xi'_A), \left(y^{(l-1)}, \xi_B^{(l-1)}\right)\Big) \in R_{\Xi,w^{(l)}}$. As $\Big((x, \xi_A), \left(y^{(l)}, \xi_B^{(l)}\right)\Big) \in R_{\Xi,w^{(l)}}$ (see Equation (32)) and $R_{\Xi,w^{(l)}}$ is a combinatorial rectangle, we have that $\Big((x, \xi'_A), \left(y^{(l)}, \xi_B^{(l)}\right)\Big) \in R_{\Xi,w^{(l)}}$, as desired. □

Let Π be a DAG-protocol with inputs sets $\mathcal{X}$, $\mathcal{Y}$, and output set $\mathcal{O}$ and let $\Theta \geq 0$. Let $\{R_{\Theta,v}\}_{v \in V}$ be the rectangles associated with Π_Θ according to Definition 3.1 and $\{h_{\Theta,v}\}_{v \in V_A \cup V_B}$ be the message functions.

Definition A.2. *We say that an edge $e = (u, v) \in E$ is empty if $R_{\Theta,u} \neq \emptyset$ and for all $((x, \xi_A), (y, \xi_B)) \in R_{\Theta,u}$, we have:*

$$h_{\Theta,u}((x, \xi_A), (y, \xi_B)) \neq v.$$

Let $e = (u, v)$ be an empty edge in Π. We define a DAG-protocol Π^{-e} with the same input and output sets that does not have this edge. The graph G^{-e} for Π^{-e} is the same as G except that the edge e is removed. The output functions are unchanged and so are the message functions for all vertices other than u. For the vertex u, we have to ensure that the new message function h_u^{-e} does not map anything to v. For this, we first claim that u must have another out neighbor $v' \neq v$, as otherwise the condition in Definition A.2 is impossible. We define:

$$h_u^{-e}(x,y) = \begin{cases} h_u(x,y), & \text{if } h_u(x,y) \neq v \\ v', & \text{if } h_u(x,y) = v \end{cases}. \tag{33}$$

Let $\left\{R_{\Theta,v}^{-e}\right\}_{v\in V}$ be the rectangles associated with Π_Θ^{-e} according to Definition 3.1 and $\left\{h_{\Theta,v}^{-e}\right\}_{v\in V_A\cup V_B}$ be the message functions. Lemma A.5 is our main result about Π^{-e}, which says that the rectangles $\left\{R_{\Theta,v}^{-e}\right\}_{v\in V}$ and $\{R_{\Theta,v}\}_{v\in V}$ have the same "projection" on the inputs in the sets $\mathcal{X}$ and $\mathcal{Y}$. Before stating it, we need the following definition:

Definition A.3. *Let $C \in \{A,B\}$ and $\xi_C^- : V_C \to V \cup \{*\}$. Let W be a path in G that starts from rt. Define the error pattern $\mathsf{curb}_{C,\xi_C^-}(W)$ to match ξ_C^- on all vertices in $V_C \cap W$ and equal $*$ on all vertices in $V_C \setminus W$.*

Lemma A.4. *Let $e=(u,v)$ be an empty edge, $C \in \{A,B\}$. For all $\xi_C^- \in (\Xi_\Theta(\Pi^{-e}))_C$ and all paths W in G^{-e} starting from rt, we have $\mathsf{curb}_{C,\xi_C^-}(W) \in (\Xi_\Theta(\Pi))_C$.*

Proof. Define $\xi_C = \mathsf{curb}_{C,\xi_C^-}(W)$ for convenience. For any path P in G that starts at rt and ends at a node in V_O, all the nodes $w \in V_C \cap P$ that satisfy $\xi_C(w) \neq *$ must also be in W. Thus, the number of such nodes in P is at most the number on W which is at most Θ, by the fact that W is a path in G^{-e} and $\xi_C^- \in (\Xi_\Theta(\Pi^{-e}))_C$. The lemma follows. $\square$

Lemma A.5. *Let $e=(u,v)$ be an empty edge and $w \in V$. We have:*

1. *Let $x \in \mathcal{X}$ be given. For all $\xi_A^- \in (\Xi_\Theta(\Pi^{-e}))_A$ such that $(x,\xi_A^-) \in \left(R_{\Theta,w}^{-e}\right)_A$, there exists a path W in G^{-e} starting from rt and ending at w such that $\left(x, \mathsf{curb}_{A,\xi_A^-}(W)\right) \in (R_{\Theta,w})_A$.*

2. *Let $y \in \mathcal{Y}$ be given. For all $\xi_B^- \in (\Xi_\Theta(\Pi^{-e}))_B$ such that $(y,\xi_B^-) \in \left(R_{\Theta,w}^{-e}\right)_B$, there exists a path W in G^{-e} starting from rt and ending at w such that $\left(y, \mathsf{curb}_{B,\xi_B^-}(W)\right) \in (R_{\Theta,w})_B$.*

Proof. If w is not reachable from rt in the graph G^{-e}, and we have from Definition 3.1 that $R_{\Theta,w}^{-e} = \emptyset$, and the lemma is true vacuously. For vertices w that are reachable from rt in the graph G^{-e}, we prove by induction. For the base case, note that $w = \mathsf{rt}$ implies from Definition 3.1 that $R_{\Theta,w} = (\mathcal{X} \times (\Xi_\Theta(\Pi))_A) \times (\mathcal{Y} \times (\Xi_\Theta(\Pi))_B)$. The result now follows from Lemma A.4.

We now consider the case $w \neq \mathsf{rt}$. We only show Item 1 as the proof of Item 2 is analogous. Fix an arbitrary $\xi_A^- \in (\Xi_\Theta(\Pi^{-e}))_A$ such that $(x,\xi_A^-) \in \left(R_{\Theta,w}^{-e}\right)_A$. By Observation 3.2, there $w' \in V$ such that (w',w) is an edge in G^{-e} and $(y,\xi_B^-) \in \mathcal{Y} \times (\Xi_\Theta(\Pi^{-e}))_B$ such that $((x,\xi_A^-),(y,\xi_B^-)) \in R_{\Theta,w'}^{-e}$ and $h_{\Theta,w'}^{-e}((x,\xi_A^-),(y,\xi_B^-)) = w$. Consider the following cases:

- **When $w' \in V_A$:** By Item 1 of the induction hypothesis on w' to get that there exists a path W' in G^{-e} starting from rt and ending at w' such that $\left(x, \mathsf{curb}_{A,\xi_A^-}(W')\right) \in (R_{\Theta,w'})_A$. As $e=(u,v)$ is an empty edge, we get from Definition A.2 that either $w' \neq u$ or $h_{\Theta,w'}\left(x, \mathsf{curb}_{A,\xi_A^-}(W')\right) \neq v$.

Define the path W to be W' appended with the edge (w', w). To show the lemma, it suffices to show that $\big(x, \mathsf{curb}_{A,\xi^{-}_A}(W)\big) \in (R_{\Theta,w})_A$. Observe from Definition A.3 and Lemma A.4 that the error patterns $\mathsf{curb}_{A,\xi^{-}_A}(W')$ and $\mathsf{curb}_{A,\xi^{-}_A}(W)$ are in $(\Xi_\Theta(\Pi))_A$ and only differ on vertices $v \in V_A$ that are reachable from w (in fact, they can only differ on the vertex w itself). By Lemma A.1 this means that showing $\big(x, \mathsf{curb}_{A,\xi^{-}_A}(W)\big) \in (R_{\Theta,w})_A$ is the same as showing $\big(x, \mathsf{curb}_{A,\xi^{-}_A}(W')\big) \in (R_{\Theta,w})_A$, which we do next.

By Definition 3.1 and the fact that $w' \in V_A$ implies that $h_{\Theta,w'}$ is only a function of the first argument, this follows if we show that $h_{\Theta,w'}\big(x, \mathsf{curb}_{A,\xi^{-}_A}(W')\big) = w$. Owing to the definition of w', this is the same as showing $h_{\Theta,w'}\big(x, \mathsf{curb}_{A,\xi^{-}_A}(W')\big) = h^{-e}_{\Theta,w'}(x, \xi^{-}_A)$. If $\xi^{-}_A(w') \neq *$, then, as $\mathsf{curb}_{A,\xi^{-}_A}(W')$ matches ξ^{-}_A on w', this follows from Equation (3). Otherwise, Equation (3) says that $h_{\Theta,w'}\big(x, \mathsf{curb}_{A,\xi^{-}_A}(W')\big) = h_{w'}(x)$ and $h^{-e}_{\Theta,w'}(x, \xi^{-}_A) = h^{-e}_{w'}(x)$ implying that this is the same as showing $h_{w'}(x) = h^{-e}_{w'}(x)$, which is true by Equation (33) as either $w' \neq u$ or $h_{w'}(x) = h_{\Theta,w'}\big(x, \mathsf{curb}_{A,\xi^{-}_A}(W')\big) \neq v$.

- **When $w' \in V_B$:** By Items 1 and 2 of the induction hypothesis on w', we get that there exist paths W', W'' in G^{-e}, both starting from rt and ending at w' such that $\big(x, \mathsf{curb}_{A,\xi^{-}_A}(W')\big) \in (R_{\Theta,w'})_A$ and $\big(y, \mathsf{curb}_{B,\xi^{-}_B}(W'')\big) \in (R_{\Theta,w'})_B$. As $e = (u, v)$ is an empty edge, we get from Definition A.2 that either $w' \neq u$ or $h_{\Theta,w'}\big(y, \mathsf{curb}_{B,\xi^{-}_B}(W'')\big) \neq v$. Moreover, as $R_{\Theta,w'}$ is a combinatorial rectangle, we conclude that:

$$\Big(\big(x, \mathsf{curb}_{A,\xi^{-}_A}(W')\big), \big(y, \mathsf{curb}_{B,\xi^{-}_B}(W'')\big)\Big) \in R_{\Theta,w'}. \tag{34}$$

  We now claim that $h_{\Theta,w'}\big(y, \mathsf{curb}_{B,\xi^{-}_B}(W'')\big) = h^{-e}_{\Theta,w'}(y, \xi^{-}_B)$, which also equals w by the definition of w' (recall that $w' \in V_B$ implies that the message function is only a function of the second argument). If $\xi^{-}_B(w') \neq *$, then as $\mathsf{curb}_{B,\xi^{-}_B}(W'')$ matches ξ^{-}_B on w', this follows from Equation (3). Otherwise, Equation (3) says that $h_{\Theta,w'}\big(y, \mathsf{curb}_{B,\xi^{-}_B}(W'')\big) = h_{w'}(y)$ and $h^{-e}_{\Theta,w'}(y, \xi^{-}_B) = h^{-e}_{w'}(y)$ implying that this is the same as showing that $h_{w'}(y) = h^{-e}_{w'}(y)$, which is true by Equation (33) as either $w' \neq u$ or $h_{w'}(y) = h_{\Theta,w'}\big(y, \mathsf{curb}_{B,\xi^{-}_B}(W'')\big) \neq v$. From Equation (34) and $h_{\Theta,w'}\big(y, \mathsf{curb}_{B,\xi^{-}_B}(W'')\big) = w$, conclude using Definition 3.1 that:

$$\Big(\big(x, \mathsf{curb}_{A,\xi^{-}_A}(W')\big), \big(y, \mathsf{curb}_{B,\xi^{-}_B}(W'')\big)\Big) \in R_{\Theta,w}.$$

  It follows that $\big(x, \mathsf{curb}_{A,\xi^{-}_A}(W')\big) \in (R_{\Theta,w})_A$. Define the path W to be W' appended with the edge (w', w). As $w' \in V_B$, we have by Definition A.3 that $\mathsf{curb}_{A,\xi^{-}_A}(W') = \mathsf{curb}_{A,\xi^{-}_A}(W)$ implying that $\big(x, \mathsf{curb}_{A,\xi^{-}_A}(W)\big) \in (R_{\Theta,w})_A$, as desired.

$\square$

The above lemma implies the main theorem of this section, showing that one can remove empty edges without affecting correctness.

Theorem A.6. *Let* Π *be a* DAG*-protocol that solves a search problem* S *with rectangular correctness despite* $\Xi_\Theta(\Pi)$*. Let* e *be an empty edge in* Π *and let the* DAG*-protocol* $\Pi^{\text{-}e}$ *be as define above. Then,* $\Pi^{\text{-}e}$ *solves* S *with rectangular correctness despite* $\Xi_\Theta(\Pi^{\text{-}e})$*.*

Proof. We have to show that $\Pi^{\text{-}e}_\Theta$ solves $S_{\Xi_\Theta(\Pi^{\text{-}e})}$ with rectangular correctness, where $S_{\Xi_\Theta(\Pi^{\text{-}e})} \subseteq (\mathcal{X} \times (\Xi_\Theta(\Pi^{\text{-}e}))_A) \times (\mathcal{Y} \times (\Xi_\Theta(\Pi^{\text{-}e}))_B) \times \mathcal{O}$ is the search problem satisfying $((x, \xi^{\text{-}}_A), (y, \xi^{\text{-}}_B), o) \in S_{\Xi_\Theta(\Pi^{\text{-}e})} \iff (x, y, o) \in S$ for all values of $x, y, o, \xi^{\text{-}}_A, \xi^{\text{-}}_B$. By Definition 3.3, have to show that for all $v \in V_O$ and all $((x, \xi^{\text{-}}_A), (y, \xi^{\text{-}}_B)) \in R^{\text{-}e}_{\Theta,v}$, we have $(x, y, o_v) \in S$.

Fix an arbitrary $v \in V_O$ and an arbitrary $((x, \xi^{\text{-}}_A), (y, \xi^{\text{-}}_B)) \in R^{\text{-}e}_{\Theta,v}$. By Lemma A.5, we get that there exists paths W', W'', both starting from rt and ending at v such that $\big(x, \mathsf{curb}_{A,\xi^{\text{-}}_A}(W')\big) \in (R_{\Theta,v})_A$ and $\big(y, \mathsf{curb}_{B,\xi^{\text{-}}_B}(W'')\big) \in (R_{\Theta,v})_B$. As $R_{\Theta,v}$ is a combinatorial rectangle, it follows that:

$$\big(\big(x, \mathsf{curb}_{A,\xi^{\text{-}}_A}(W')\big), \big(y, \mathsf{curb}_{B,\xi^{\text{-}}_B}(W'')\big)\big) \in R_{\Theta,v}.$$

Now, as Π solves S with rectangular correctness despite $\Xi_\Theta(\Pi)$, we get that Π_Θ solves $S_{\Xi_\Theta(\Pi)}$ with rectangular correctness, where $S_{\Xi_\Theta(\Pi)} \subseteq (\mathcal{X} \times (\Xi_\Theta(\Pi))_A) \times (\mathcal{Y} \times (\Xi_\Theta(\Pi))_B) \times \mathcal{O}$ is the search problem satisfying $((x, \xi_A), (y, \xi_B), o) \in S_{\Xi_\Theta(\Pi)} \iff (x, y, o) \in S$ for all values of x, y, o, ξ_A, ξ_B. By Definition 3.3, it follows that $(x, y, o_v) \in S$, as desired. □