Electronic Colloquium on Computational Complexity, Report No. 55 (2017)

Round Complexity Versus Randomness Complexity in Interactive Proofs

Maya Leshkowitz*

March 26, 2017

Abstract

Consider an interactive proof system for some set S that has randomness complexity $r(n)$ for instances of length n, and arbitrary round complexity. We show a public-coin interactive proof system for S of round complexity $O(r(n)/\log n)$. Furthermore, the randomness complexity is preserved up to a constant factor, and the resulting interactive proof system has perfect completeness.

*Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel. Email: maya.leshkowitz@weizmann.ac.il

ISSN 1433-8092

Contents

1 Introduction

The notion of interactive proof systems, put forward by Goldwasser, Micali, and Rackoff [8], and the demonstration of their power by Lund, Fortnow, Karloff, Nisan [11] and Shamir [13] are among the most celebrated achievements of complexity theory.

Loosely speaking, interactive proof systems capture the most general way in which one party can efficiently verify claims made by another, more powerful, party. The definition of interactive proof systems generalizes the traditional notion of a proof system (indeed, an NP-proof system), by allowing both *interaction* and *randomness*.

It is well known that both interaction and randomness are inherent to the power of interactive proof systems, where here we mean the extra power above that of NP-proof systems. Interactive proofs with no randomness can be easily transformed into NP-proof systems, whereas randomized non-interactive proofs, captured by the class $\mathcal{MA}$ (defined by Babai [1]), are merely the randomized version of $\mathcal{NP}$ (e.g., loosely speaking, if $\mathcal{BPP} = \mathcal{P}$, then $\mathcal{MA} = \mathcal{NP}$).

Both *interaction* and *randomness* are quantitative notions; that is, one talks of the "amount of interaction", which is commonly associated with the (total) number of messages exchanged (a.k.a number of *rounds*), and of the amount of randomness (a.k.a *randomness complexity*). While the previous paragraph refers to the qualitative question and asserts that both interaction and randomness are essential, a finer study of the quantitative question is called for; that is, it is natural to ask about the necessary amount of interaction and randomness in various interactive proof systems.

The study of the round-complexity aspect of interactive proof systems is well known: Babai and Moran showed that the round complexity of any public-coin interactive proof system (a.k.a Arthur-Merlin proofs) can be reduced by a constant factor [2], whereas the transformation of Goldwasser and Sipser [9] (which essentially preserves the number of rounds) extends this result to general interactive proof systems. It is also known that a stronger round reduction is quite unlikely, since it would place `SAT` in co-AM-time$(2^{o(n)})$, whereas AM-time(T) may equal Ntime(poly(T)). (This is the case due to a combination of results reviewed below (in the paragraph titled "conditional tightness").)

In contrast, we are only aware of one study that focuses on the randomness complexity of interactive proof systems [1]. Specifically, Bellare *et al.* [3] studied the randomness complexity of *error reduction* in the context of interactive proof systems. (We mention that the randomness complexity is also of interest in [5], which provides an alternative transformation of general interactive proof systems to public-coin ones.)

1.1 Round complexity versus randomness complexity

A natural question, which to the best of our knowledge was not considered before, is what is the relation between the two foregoing complexity measures. We do suspect that the randomness complexity of interactive proof systems *may* be much higher than the number of rounds, since constant-round interactive proof systems seem more powerful than NP-proof systems (see, e.g., the Graph Non-Isomorphism proof of [6]), whereas a logarithmic amount of randomness is clearly useless. But *can the randomness complexity be smaller than the round complexity?*

The answer is definitely negative if we consider *public-coin* interactive proof systems. Recall that in these proof systems, in each round, the verifier sends the outcome of fresh coins

[1]We refer to unconditional results and not to the long line of research of randomness versus hardness trade-off that rely on uniform or non-uniform assumptions, see, e.g. [10],[12].

that it has tossed at the beginning of the current round, and so by definition the number of coin tosses is at least as large as the number of rounds.[2] However, it is not clear what happens in case of general interactive proofs. (In particular, the transformation of [9] significantly increases the randomness complexity by a factor depending on the number of rounds.)

Recall that in a general interactive proof system, the verifier may toss all coins at the very beginning, but its message in each round may be a complex function of the outcome of these coins (and the messages it has received from the prover). In particular, the verifier message may have very little information contents (from the prover's point of view), and so we may have many more rounds than the number of coins tossed. Furthermore, it is not clear how to collapse rounds that yield verifier messages of low information contents. These are the issues we deal with when showing that also in general interactive proof systems, randomness complexity $r(n)$ yields round complexity $O(r(n)/\log n)$.

Theorem 1 *(Main Theorem). Suppose that S has an interactive proof system of randomness complexity $r(n)$ for instances of length n. Then, S has a public-coin interactive proof system of round complexity $O(r(n)/\log n)$ and randomness complexity $O(r(n))$. Furthermore, the resulting interactive proof system has perfect completeness.*

Note that, in addition to obtaining the public-coin feature, we obtain perfect completeness for free. That is, even if the original system does not have perfect completeness, the new one has this feature.

We note that it is easier to prove a weaker version of the main theorem, which does not obtain the public-coin and perfect completeness features. The proof of this weaker result, outlined in Section 1.2, and given in detail in Appendix C, illustrates one of the ideas that underlies the proof of Theorem 1.

Conditional tightness: The round-complexity obtained by Theorem 1 is the best one may hope for at this time, since a result asserting round complexity $o(r(n)/\log n)$ for any set that has an interactive proof system of randomness complexity $r(n)$ would yield an unexpected result that conflicts with common beliefs and seems currently out of reach. Specifically, it would place SAT in co-AM-time$(2^{o(n)})$, which does contradict common beliefs. The full reasoning is as follows:

1. A variant of the celebrated interactive proof system for $\overline{\text{SAT}}$ yields an interactive proof system of randomness complexity $O(n)$ for unsatisfiable CNFs with n variables. (This interactive proof consists of $n/\log n$ rounds such that in each round we strip a single variable in the sum-check that sums over $n/\log n$ variables with values in $[n]$, while using a finite field of size poly(n).)[3]

[2]This presumes that the definition requires the verifier to send a non-empty message in each round. But otherwise (i.e., if the definition allows empty messages), rounds in which the verifier sends nothing can be collapsed.

[3]This is done by packing a sequence of $\log n$ bits of the boolean variables into a symbol of $H = [n] \subseteq F$ where F is some field. For $i \in \ell$, where $\ell = \log n$, denote by $f_i(x) : F \to F$ the polynomial of degree $n-1$ that maps each element in H to the value of its ith boolean variable. Now, take the standard arithmetization of the CNF and replace each occurrence of the variable indexed $j \cdot \ell + i$ by the polynomial $f_i(x_j)$, where x_j is the F variable that represents the jth block of boolean variables. The resulting polynomial is a n/ℓ variable polynomial of total degree $O(m \cdot n)$, where m is the number of clauses. Finally, the number of satisfying assignments is given by the sum over all $(y_1, ..., y_{n/\ell}) \in H^{n/\ell}$ of the polynomial derived above. Furthermore, we do not execute the sum-check protocol over an exponentially large finite field but rather over a finite field of prime cardinality $p = \text{poly}(n)$, where p is selected by the verifier at random among such primes.

2. On the other hand, any set having an m-round interactive proof system is in AM-time($n^{O(m)}$), see [7, Apdx B]. Hence, if unsatisfiable CNFs have an interactive proof of round complexity $o(n/\log n)$, then such instances can be refuted in AM-time($2^{o(n)}$), whereas AM-time(T) may equal Ntime(poly(T)).

A different perspective: Theorem 1 may be viewed as an alternative transformation of general interactive proof systems into public-coin ones. Recall that the transformation of Goldwasser and Sipser [9] preserves the round-complexity of the original system (up to an additive constant), but increases the randomness complexity (i.e., raising it to a constant power). The same holds in the variant of that transformation presented in [5]. In contrast, Theorem 1 preserves the randomness complexity of the original system (up to a constant factor), but does not preserve the round-complexity. Taking this perspective, the fact that the round-complexity is bounded in terms of the randomness complexity is a consequence of the fact that the resulting scheme is of the public-coin type.

1.2 On the proof of Theorem 1

We start by giving an overview of a proof of a weaker result, in which we show how to transform any interactive proof, of randomness complexity $r(n)$, to a *private-coin* interactive proof for the same set that uses $O(r(n)/\log n)$ rounds, while maintaining the randomness complexity of $r(n)$. This proof gives a flavor of the proof of the main theorem, but is significantly simpler.

1.2.1 Private-coin Emulation Protocol

The idea of the emulation protocol is that, in every iteration, we would like the prover to send possible continuations of the current transcript (describing execution segments of possibly different number of rounds) that reveal much information about the verifier's random coins. Hence, the prover sends partial transcripts of *maximal* length such that each account for a large fraction of the residual probability mass[4]. The verifier then checks if one of these transcripts is consistent with the strategy determined by the values of its random coins, which were tossed upfront. If so, the verifier picks the maximal transcript consistent with its strategy and the verifier and prover proceed their interaction from that point. Otherwise, the verifier sends its next message (based on the aforementioned coins) without using the continuations suggested by the prover. We stress that the only source of the verifier's randomness is its private coins tossed upfront, which are used to determine the continuation of the transcript in each subsequent iteration.

We wish to elaborate on how the prover determines the continuations of the transcripts. Fixing an iteration, we denote the current transcript by γ and its residual probability mass by $p(\gamma)$. Each transcript the prover sends on this iteration is a possible continuation of γ of *maximal* length that is a "heavy continuation". By a heavy continuation γ', we mean that γ' has probability mass greater than $p(\gamma)/n$, when subtracting from it the probability mass of the continuations of γ that were either sent by the prover in previous iterations, or determined in this one.

This conditioning allows the prover to send several continuations of the transcript that are also continuations of each other. Consider for example the case that the prover sends

[4]Note that in the eyes of an observer, a verifier that samples its random coins at the beginning of the interaction and proceeds accordingly, is equivalent to a verifier that on each round samples a message with probability proportional to its residual probability mass.

$\gamma\alpha_1\beta_1\alpha_2\beta_2$ and $\gamma\alpha_1\beta_1$. In this case if the verifier chooses $\gamma\alpha_1\beta_1$ it means that in the next iteration the continuation of this transcript cannot begin with $\alpha_2\beta_2$.

The benefit of this method of determining transcript-continuations is that we guarantee that in the *next* iteration the probability mass of the new transcript is *lower* than $p(\gamma)/n$. The reasoning is as follows. If the verifier chooses one of the transcripts suggested by the prover, then on the *next* iteration the residual probability mass of each of its continuations is lower than $p(\gamma)/n$, otherwise this continuation should have suggested by the prover on the previous iteration. If the verifier did not choose any of the transcripts, and instead continued the transcript with its own message $\widetilde{\alpha_1}$, then it follows that the residual probability mass of the transcript $\gamma\widetilde{\alpha_1}$ (under the conditioning of the appropriate events) is also lower than $p(\gamma)/n$, otherwise the continuation $\gamma\widetilde{\alpha_1}$ should have been suggested by the prover.

It follows that after $O(r(n)/\log n)$ rounds of interaction the probability of the transcript generated is at most $(1/n)^{r(n)/\log n} = 1/2^{r(n)}$, which means that there is a unique value of the coin tosses consistent with the transcript. Hence, a complete transcript is generated and the verifier can reject or accept at this point. It is easy to show that if the prover follows the prescribed emulation, then the verifier accepts with the same probability as in the original interactive proof system, and hence completeness is maintained.

Note that the above emulation per se does not suffice. It is essential to include validation checks that guarantee that the transcripts provided by the prover are consistent with some prover strategy for the original protocol. This means that if the prover provides two transcripts that share a prefix, this common prefix must end with a prover's message. This implies that the prover answers in the same way to the same verifier messages, which means that the prover's strategy is consistent with some prover of the original emulation, and so the soundness of the original proof system is maintained.

1.2.2 Public-coin Emulation Protocol

The simplified private-coin emulation protocol captures one of the key ideas of our public-coin emulation protocol. The difficulty that we face when seeking a public-coin emulation is that we cannot rely on hidden coins tossed upfront by the verifier. Thus, when presented with a list of heavy continuations, it is unclear how the verifier should select one at random, since the selection probability should be determined by the residual probability masses that are unknown to it. Our solution is to have the prover provide these probabilities, but this raises the need to verify these claimed values. (Needless to say, the verifier rejects upfront if the sum of these probabilities does not match the claimed probability of the transcript as determined before the current round.) In the last iteration, a complete transcript is sampled, containing the verifier's private coins, hence the validity of the transcript and the claim can be checked.

The foregoing description raises a few issues. Firstly, the prover should find a way to communicate all the transcripts to the verifier, and not only the ones with high residual probability mass as before. Second, it is not clear what happens when the prover provides wrong values for the residual probabilities. As for the second issue, note that maliciously raising the probability of a transcript does contribute towards having the sum of probabilities meet the prior claim, but it makes the probability that this transcript is selected higher, and so puts the prover in greater problem in the next round. Indeed, a careful analysis shows that actually the prover gains nothing by such behavior, since when the transcript is complete, false claims about its residual probability are easily detected.

Turning back to the first issue, we note that the issue is that there may be too many short transcripts that each account for a small fraction of the residual probability mass. To deal with this case, we pack many transcripts into a single auxiliary message, which means that we use a succinct representation of a sequence that contains many of the transcripts but not all of them (since otherwise we would have made no progress at the current round). The succinct representation should support the verification that the corresponding sequences are disjoint. Now, each such "pack" of transcripts will be assigned the corresponding probability mass, and be treated as if it were an actual transcript.

Needless to say, the foregoing is but a very rough sketch of the structure of the derived proof system. The actual proof system uses a carefully designed verification procedure that ensures that its executions can be mapped to executions in the original proof system.

We note that while the above description of the public-coin emulation refers to the probability that various transcripts appear in the original proof system (when the prover uses an optimal strategy), our actual construction refers only to accepting transcripts (i.e., transcripts that lead the original verifier to accept). Consequently, we obtain a proof system of perfect completeness, even if the original proof system had two-sided error probability.

1.3 Organization

Towards proving the main theorem we shall show how to emulate an existing interactive proof system with a public coin emulation protocol that has $O(r(n)/\log n)$ rounds. We begin by introducing the notion of "protocol trees" in Section 3, which we use to describe the interaction of the verifier and prover of the original interactive proof system. In Section 4, we shall show how to transform the protocol tree into an "emulation tree", that contains the continuations of the transcripts that the prover sends on each iteration along with their probability masses. Using this emulation tree, we then turn to describing the public-coin emulation protocol for the new prover and verifier, in Section 5. The analysis of the emulation, which is given in Section 6, is partitioned into completeness (Subsection 6.1) and soundness (Subsection 6.2).

In Appendix C, we shall show how to emulate the existing interactive proof system with a private-coin emulation protocol that has $O(r(n)/\log n)$ rounds. The organization of this section is similar to the organization of the main part of the paper, although most sections of it are less involved. Appendix C is written so it can be read independently of the rest of the paper, and the decision if to read it before or after the other parts of the paper is left to the reader.

We provide notes that point out the main differences and similarities between the private and public-coin emulation protocols. These notes are typeset as this one.

2 Preliminaries

Let us start by formally defining interactive proof systems, where the completeness and soundness bounds are parameters.

Definition 2 ***[Interactive Proof Systems]*** *Let $c, s : \mathbb{N} \to [0,1]$ such that $c(|x|) \geq s(|x|) + \frac{1}{poly(|x|)}$. An interactive proof system for a set S is a two party game, between a verifier executing a probabilistic polynomial time strategy, denoted V, and a prover executing a* (computationally unbounded) *strategy satisfying the following two conditions:*

- *Completeness with bound c: For every $x \in S$, the verifier V accepts after interacting with the prover P on common input x with probability at least $c(|x|)$.*

- *Soundness with bound s: For every $x \notin S$ and every prover strategy $\widetilde{P}$, the verifier V accepts after interacting with $\widetilde{P}$ on common input x with probability at most $s(|x|)$.*

When c and s are not specified, we mean $c \equiv 2/3$ and $s \equiv 1/3$. We denote by $\mathcal{IP}$ the class of sets having interactive proof systems. When $c \equiv 1$, we say that the system has perfect completeness.

3 The protocol tree of the original proof system

Note that the protocol tree for the private-coin emulation described in Section C.1 is similar to the one described here, except for the definition of weights.

Fixing an interactive proof and an instance x of length n, we describe the possible prover-verifier interactions of the system on common input x using a tree whose height corresponds to the number of rounds of interaction. For some $\ell = \ell(n)$, we assume without loss of generality that in each round the verifier sends a message $\alpha \in \{0,1\}^\ell$, and the prover's responds with a message $\beta \in \{0,1\}^\ell$. We can also assume, without loss of generality, that the prover's strategy is deterministic and fixed. Each node v in level j represents a possible prover-verifier transcript for the first j rounds of the interaction. The branching of the tree represents the possible ways to extend the transcript to the next round. The number of ways to extend the transcript depends only on the verifier's message, since we fixed the prover's strategy. Hence, each node has *at most* $d := 2^\ell$ children, corresponding to the 2^ℓ possible verifier messages for the next round. The prover's response to each such message is included in the **description** of the corresponding node.

The description of a node u on level j contains the partial **transcript** $\gamma(u) = \alpha_1\beta_1, \ldots, \alpha_j\beta_j$ of the interaction up to the j'th round. The root (at level zero) has an empty transcript, whereas a leaf of the tree represents a complete prover-verifier interaction. We can assume, without loss of generality, that the verifier sends its private coins on the last round, and hence every leaf is associated with a sequence of coin tosses which either leads the verifier to accept or to reject. Hence, we can represent the possible interactions generated by the interactive proof system using a tree of height m which has $2^{r(n)}$ leaves, where m is the number of rounds and $r(n)$ is the number of coin tosses. Using a constant number of parallel repetitions, we can assume that the interactive proof system has completeness parameter $\frac{9}{10}$ and soundness parameter $\frac{1}{10}$. Note that this blows up the randomness complexity only by a constant factor (as compared to our interactive proof for the standard $\frac{1}{3}$, $\frac{2}{3}$ parameters). Therefore, if x is a yes-instance then at least $\frac{9}{10} \cdot 2^{r(n)}$ of its leaves represent accepting runs, and if x is a no-instance then at most $\frac{1}{10} \cdot 2^{r(n)}$ of its leaves represent accepting runs.

The description of a node also contains its **weight**, denoted $w(u)$. The weight of the node is the number of coin sequences that are consistent with the node and lead the verifier to accept at the end of the interaction. That is,

Definition 3 ***(Weight of a leaf)*** *Let u be a leaf with transcript $\gamma(u)$ which corresponds to the full transcript of the interaction of P and V on input x, when V uses coins ρ; that is,*

$$\gamma(u) = (\alpha_1, \beta_1, \ldots, \alpha_m, \beta_m, (\rho, \sigma)) \tag{1}$$

where $\sigma = V(x, \rho, \beta_1, \ldots, \beta_m) \in \{0,1\}$ is V's final verdict and for every $i = 1, \ldots, m$ it holds that $\alpha_i = V(x, \rho, \beta_1, \ldots, \beta_{i-1})$ and $\beta_i = P(x, \alpha_1, \ldots, \alpha_i)$. We define the weight $w(u)$ of u to be V's final verdict σ.

Definition 4 ***(Weight of a node)*** *The weight of a node u in the protocol tree is the sum of the weights of the leaves that are descendants of u.*

Note that $w(u)$ is proportional the probability that $\gamma(u)$ is generated and the verifier accepts at the end of the interaction.

In the private-coin emulation the weight of all the leaves is defined as 1. Hence, in the private-coin emulation the weight of a node is the number of coin sequences that are consistent with the corresponding transcript.

4 The emulation tree

4.1 Overview

So far we explained how to represent the possible executions of a m-round interactive proof system on some instance x, where the protocol utilizes $r(n)$ coins. This resulted in a protocol tree of height m with $2^{r(n)}$ leaves. Our goal is to transform this protocol tree to an emulation tree that defines a prover strategy for a $O(r(n)/\log n)$-round public-coin emulation protocol. This transformation is done using the Build_Tree procedure. First we describe a very restricted case where the protocol tree is already suitable for our proposed public-coin emulation and the Build_Tree procedure is not required. Next, we explain how the transformation works in a restricted case when the degree of the protocol tree is bounded by poly(n), and finally in the case of a general protocol tree.

The procedure for constructing the private-coin emulation tree described in Section C.2 is similar to the one described here for the restricted case that the degree of the protocol tree is bounded by *poly*(n). Those familiar with the construction of the emulation tree for the private-coin protocol can skip to the "general case" paragraph.

The protocol tree is of height $O(r(n)/\log n)$ and degree poly(n). In order to convince the verifier that x is a yes-instance the prover makes an initial claim that the weight of the root of the protocol tree is at least $c \cdot 2^{r(n)}$. The emulation is initiated at the root of the protocol tree and on each round of the emulation the prover assists the verifier at progressing one step down the protocol tree. (This assistance is required because the verifier does not have access to the protocol tree.) Each round consists of the prover providing the verifier with the descriptions of the current node (i.e., the node u sampled on the previous round), where these descriptions contain the weights of the various children. The verifier performs validations to check that according to the descriptions these are legal children of u, and that their weights sum up to $w(u)$. Then, the verifier samples a child with probability that is approximately proportional to its weight, up to a multiplicative factor of $1 + \frac{1}{n}$, using $O(\log n)$ public coins. On the last round, a leaf is sampled, whose description contains the complete prover-verifier interaction along with the coins tossed by the verifier. The new verifier accepts if and only if the transcript sampled is consistent with the original verifier's strategy and leads the original verifier to accept.

To see why this is indeed an interactive proof system for the original language, note that an honest prover can always convince the verifier of the correctness of a true claim using this emulation. Hence the interactive proof system we described has perfect completeness. On the other hand, for no-instances, a prover that wants to make the verifier accept must make an initial claim that the weight of the empty transcript is much larger than its real weight. Namely, the real weight of the empty transcript is at most $s \cdot 2^{r(n)}$, whereas the prover claims that the weight is at least $c \cdot 2^{r(n)}$, where $c > s$ are the completeness and soundness parameters of the original interactive proof system. Thus, there is a multiplicative gap of $\frac{s}{c}$ between the real weight and the one claimed. We can show that, in expectation, this gap is maintained throughout the emulation, up to a factor of $(1 + \frac{1}{n})^n$ that comes from the approximation factor. Therefore, the probability that a leaf that corresponds to an accepting run is sampled on the last round (and hence the verifier accepts) is at most $\frac{s}{c}(1 + \frac{1}{n})^n$, which is smaller than $\frac{1}{3}$ for a suitable choice of s and c.

The degree of the protocol tree is bounded by $\mathrm{poly}(n)$**.** In this case the height of the protocol tree may be asymptotically larger than $\frac{r(n)}{\log n}$. We create a new tree of height $O(r(n)/\log n)$ to guide the prover's strategy, which we use in a way similar to how we used the protocol tree in the previous paragraph. We call this tree the **emulation tree**. The nodes in the emulation tree are nodes from the protocol tree, however the children of a node u in the emulation tree may be non-immediate descendants of u in the protocol tree.

We start with a protocol tree T rooted at r whose weight is $w(r)$, and on each step we modify this tree towards creating an emulation tree. We define a **heavy descendant** of r to be a node in T whose weight is at least $\frac{w(r)}{n}$, and the weight of each of its children is smaller than $\frac{w(r)}{n}$. Note that there are at most n such nodes.

We modify T so that the children of r in the emulation tree are its original children as well as the heavy descendants that we lift upwards to make them new children of r. This modification is performed using the Build_Tree(r) procedure, which when invoked on a node r identifies the nodes that will be children of r in the new tree, sets them as children of r, and then initiates recursive invocations on the (original and new) children of r, creating the new emulation tree rooted at r. Details follow.

Let T_r denote the temporary tree after the stage that we identify and set the children of r. We start by identifying the heavy descendants of r (they can also be children of r in T), which will become **heavy children** of r in T_r. We then proceed to the non-heavy children of r in T, which will also be children of r in T_r. After we identified the children of r in the new emulation tree, we call the Build_Tree procedure on each child of r in T_r, which creates an emulation tree rooted at that node.

Observe that in the final emulation tree, for each node u, the weight of each grandchild of u is at most $\frac{w(u)}{n}$. This is because if v is an heavy child of u in the emulation tree, then the weight of the descendants of v in T_u is at most $\frac{w(u)}{n}$. Since the children of v in the emulation tree are descendants of v in T_u the claim holds. Otherwise, v is non-heavy child of u, so its weight is smaller than $\frac{w(u)}{n}$ and hence the weight of the children of v is also smaller than $\frac{w(u)}{n}$.

It follows that the height of the final emulation tree is $O(r(n)/\log n)$. The number of children of each node is at most $\mathrm{poly}(n)$, because we add at most n heavy children to the original children of each node. Hence the emulation tree has properties similar to the protocol tree in the previous paragraph, so it is suitable for our public-coin emulation.

The general case. In general, the degree of the protocol tree is unbounded, and hence it may be exponential in n. Lifting the heavy children as we did in the case of unbounded height guarantees that the height of the new emulation tree is $O(r(n)/\log n)$, but its degree may be super-polynomial in n (due to the original children). Hence, in the case that $|x| = \mathrm{poly}(n)$

we will not be able to perform the emulation, since the verifier must run in time $\text{poly}(|x|)$. Thus, we also need to make sure the degree of the new tree is $\text{poly}(n)$.

In order to reduce the degree when it is too large, we group the non-heavy children of r under new children, which we call **interval children**. This is done in addition to handling the heavy children of r as before. In general, children of r in T may become non-immediate descendants in T_r, and non-immediate descendants of r may become immediate children of r in T_r (due to lifting).

Determining the children of r in T_r is done in two steps, as part of the Build_Tree(r) procedure. The first step is identifying the heavy descendants of r and lifting them to be children of r, creating heavy children in T_r. Next, we unite the non-heavy children of r into groups. We unite the children by lexicographic order of their transcript field, such that the weight of each group is larger than $\frac{w(r)}{n}$ and at most $\frac{2w(r)}{n}$ (except for, possibly, the last group which is only required to have weight smaller than $\frac{2w(r)}{n}$). We create a new **interval child** v for each such group, where the children of v are the nodes in the group.

After this step, the children of r in the final emulation tree are exactly the children of r in T_r. The number of children r has is at most n, since the weight of the heavy children and the interval children (except for possibly the last interval child) is at most n. Next, the procedure is called recursively on the children of r in T_r in order to create the final emulation tree.

The description of a node u in the emulation tree is composed of the **transcript** field $\gamma(u) = \alpha_1\beta_1 \dots \alpha_i\beta_i$ and a **weight** field $w(u)$ as in the original protocol tree, with an additional **range** field $R(u)$. The range of a node represents the possible range of its children's transcripts. After determining the heavy children of u, and before grouping the non-heavy children under interval children, the non-heavy children of u are all children of u in the original tree. Hence, the transcripts of the children of each interval child are the same up to the last verifier's message on which they differ, which corresponds to the branching of the protocol tree for the next round. Thus, we can label the range $R(u) = [s, e]$ where $s < e \in \{0,1\}^\ell$ according to the range of the last verifier message in the transcript field of the children of u. Heavy children have full range $\left[0^\ell, 1^\ell\right]$, whereas the range of interval children is a subinterval of $\left[0^\ell, 1^\ell\right]$ such that this subinterval corresponds to the transcripts of the nodes that are grouped under this node.

We show in the analysis that the height of the final emulation tree is $O(r(n)/\log n)$. Recall that the degree of nodes in the final emulation tree is at most n, hence it is suitable for public-coin emulation like in the previous paragraph.

(The running time of this algorithm is at least the size of the protocol tree, which is exponential in $r(n)$, and thus it may be exponential in $|x|$. However, the prover is the one that runs this algorithm and the the prover is computationally unbounded. Therefore the running time is not an issue.)

4.2 The Build Tree Procedure

Denote the designated prover and verifier of the original interactive proof system by P_0 and V_0 respectively, and the protocol tree of P_0 and V_0 for a yes-instance x by T_{P_0,V_0}. The Build_Tree procedure is a recursive procedure that reads and updates a global tree T, which is initially set to equal the protocol tree T_{P_0,V_0}until obtaining the final emulation tree, denoted by E_{P_0,V_0}. When invoked on a node u in T, the procedure determines the children of u, updates the global tree and invokes the procedure recursively on the children of u.
We denote by $T(u)$ the subtree of T rooted at u.

Initialization. The tree T is initialized to be the original protocol tree, where each node has a description that contains the weight and transcript like in the original tree, and an additional range field which is initially left empty. We set the range of the root, denoted r, to be full range $R(r) = [0^\ell, 1^\ell]$. If the weight of r is zero we terminate the process. Otherwise, we invoke the Build_Tree procedure on r.

The main procedure: Build_Tree. If u is a leaf, the procedure returns without updating the global tree T. Otherwise, the Build_Tree procedure invokes two sub-procedures, Build_Heavy(u) and Build_Interval(u), in order to identify and update the children of u in T. Finally, the Build_Tree procedure is invoked recursively on all the children of u in T.

Build_Heavy. The Build_Heavy procedure identifies the **heavy descendants** of u in $T(u)$, which are descendants of large weight that have no children of large weight, and modifies the tree by lifting them to become **heavy children** of u.

Definition 5 ***(Heavy descendants)*** *We call v a heavy descendant of u if v is a descendant of u in T and the following conditions hold:*

1. $w(v) \geq \frac{w(u)}{n}$
2. *Either v is a leaf, or for each child z of v it holds that $w(z) < \frac{w(u)}{n}$.*

For each heavy descendant, v, of u we perform the following process:

1. Update v's description: Set the range field of v to be full range $R(v) = [0^\ell, 1^\ell]$.
2. Modify the protocol tree if v is not already a child of u:
   (a) Subtract $w(v)$ from the weight of the ancestors of v in $T(u)$, except for u whose weight stays the same.
   (b) Move v (along with the subtree rooted at v) to be directly under u.

See Figure 1.

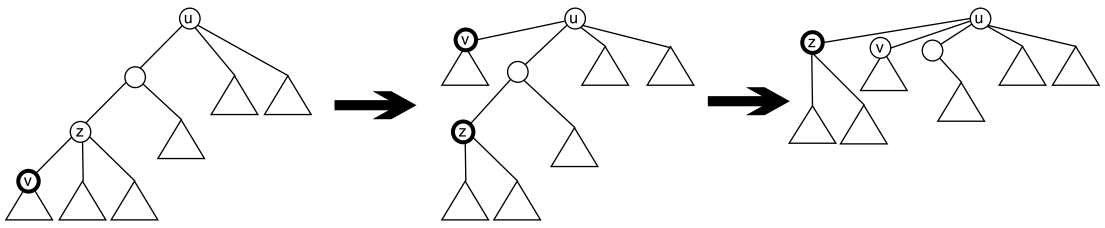

Figure 1: Build_Heavy: In the first step, v is identified as a heavy descendant of u and moved to be a heavy child of u. In the second step, z is identified and moved to be a heavy child of u. The triangles represent subtrees of the original tree.

After we finish identifying and moving the heavy children of u we perform a clean up stage where we erase all the nodes in T with weight zero.

Build_Interval(u). This procedure groups the non-heavy children of u under **interval children**. Denote the range field of u by $R(u) = [s(u), e(u)]$. (Note that $s(u) = 0^\ell$ and $e(u) = 1^\ell$ unless u is an interval node, in which case its range is partial.) We partition the range of u into a sequence of consecutive intervals, each one representing the range of a new child of u. As long as we have not partitioned all of the range $[s(u), e(u)]$ we perform the following procedure.

1. Determine s', the starting point of the interval child's range: Initially, for the first interval child of u we set $s' = s(u)$. For the next interval children, if the end of the range of the previously created interval child is $\tilde{e}$, then we set $s' = \tilde{e} + 1$.
2. Determine e', the ending point of the interval child's range: For each $e \in \{0,1\}^\ell$, denote by *non_heavy*(s', e) the set of children of u in $T(u)$ whose weights are smaller than $\frac{w(u)}{n}$ and their last verifier message α (in the transcript field) is in the range $[s', e]$. Note that when $[s', e] \neq [s(u), e(u)]$ the set *non_heavy*(s', e) can be a proper subset of set of non-heavy children of u. We define the weight of the set *non_heavy*(s', e), which we denote by $W(s', e)$, as the sum of the weights of nodes in *non_heavy*(s', e).
   We set e', to be the minimal $e \in \{0,1\}^\ell$ that satisfies $W(s', e) \geq \frac{w(u)}{n}$. If no such e exists and $W(s', e(u)) > 0$, we set $e' = e(u)$. If $W(s', e(u)) = 0$ there is no need to create another interval child so we return to the Build_Tree procedure. (This guarantees that the weight of an interval child is at least 1).
3. Create a new node v: We set the transcript of v to be like the transcript of u, $\gamma(v) = \gamma(u)$, its range to be $R(v) = [s', e']$ and its weight to be $w(v) = W(s', e')$.
4. Place v in the tree: disconnect u from the nodes in *non_heavy*(s', e'). Set u as a parent of v and let v be the parent of all nodes that are in *non_heavy*(s', e').

See Figure 2. Note that the weight of an interval child of u is at most $\frac{2w(u)}{n}$ and at least $\frac{w(u)}{n}$, except possibly for the last interval child, whose weight is at least 1.

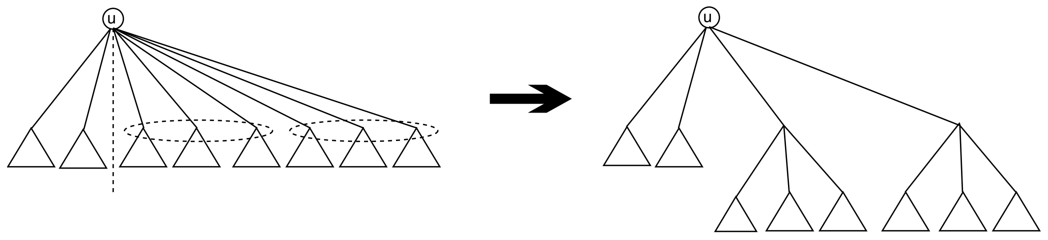

Figure 2: Build_Interval: The left diagram represents the tree before the Build_Interval procedure. The nodes to the left of the dashed line are heavy children of u. The group of nodes inside each dashed circle are united under an interval node. The tree on the right is the result of applying the Build_Interval procedure.

4.3 Properties of the emulation tree

Recall that in the original protocol tree, v was a child of u if and only if $\gamma(v) = \gamma(u)\alpha\beta$ where $\alpha, \beta \in \{0,1\}^\ell$ denote the next verifier message and the prover's response to it. However, in the new emulation tree, E_{P_0,V_0}, this is not the case. Namely, if v is a child of u in E_{P_0,V_0}, then it could be that v is an heavy child of u and hence $\gamma(v) = \gamma(u)\alpha_1\beta_1, \ldots, \alpha_k\beta_k$ for some $\alpha_1, \beta_1, \ldots, \alpha_k, \beta_k \in \{0,1\}^\ell$, or v is an interval child hence $\gamma(v) = \gamma(u)$ and $R(v) \subseteq R(u)$. Nevertheless, the following properties of the final emulation tree E_{P_0,V_0} are readily verified.

Claim 6 ***(Node degree)*** *Each node u in the final emulation tree E_{P_0,V_0} has at most n children.*

Proof. Note that we call Build_Tree on every node in E_{P_0,V_0}. By definition, the weight of the heavy children of u is at least $\frac{w(u)}{n}$. The weight of the interval children is also at least $\frac{w(u)}{n}$ except for possibly the last interval child whose weight is non zero. Therefore, the number of children is at most n. (If there were $n+1$ children or more, then the n first children would have weight of at least $\frac{w(u)}{n}$, and the last child has positive weight, which means that in total the sum of the weights of the children is greater than $w(u)$, in contradiction.) ■

Claim 7 ***(Weight reduction)*** *For every node u in the final emulation tree E_{P_0,V_0} the weight of each grandchild of u in E_{P_0,V_0} is at most $\frac{2w(u)}{n}$.*

Proof. Let v be a child of u in the emulation tree. First consider the case that v is a heavy child of u. Denote by T_u the temporary tree in the process of construction, after we determine the new children of u and before the recursive invocations of the procedure on the children of u. By the definition of heavy children, the weight of the children of v in T_u is at most $\frac{w(u)}{n}$. Thus, the weights of the ancestors of v in T_u is also at most $\frac{w(u)}{n}$, because the weight of a node is at most the weight of its ancestors. Now, if z is a heavy child of v (i.e. heavy with respect to $w(v)$) in the final emulation tree E_{P_0,V_0}, then it is a descendant of v in T_u, so its weight is at most $\frac{w(u)}{n}$. Otherwise, z is an interval child of v so its weight is at most $\frac{2w(v)}{n}$, which is at most $\frac{2w(u)}{n}$.

In case v is an interval child of u, its weight is at most $\frac{2w(u)}{n}$. Hence the weight of the grandchildren of u, which are children of v, is also at most $\frac{2w(u)}{n}$. ■

Corollary 8 ***(Corollary to Claim 7)*** *The height of the final emulation tree E_{P_0,V_0} is $O(r(n)/\log n)$.*

Proof. The weight of the root of the protocol tree is at most the number of leaves in the tree, which is $2^{r(n)}$. When we start constructing the emulation tree, the weight of the root is the same as in the protocol tree. Moreover, the weight of a node does not change from the point that we call Build_Tree on it. Hence the weight of the root in E_{P_0,V_0} is also at most $2^{r(n)}$. By Claim 7, the weight of a node in level $2i$ of the emulation tree is at most $\frac{2^{r(n)}}{(\frac{n}{2})^i}$. Taking $i = \lceil \frac{r(n)}{\log(\frac{n}{2})} \rceil$ we get that the weight of a node in level $2i$ of the emulation tree is at most 1. Lastly note that a node with weight 1 cannot have grandchildren, else by Claim 7 their weight is smaller than 1. This cannot happen since the weights of nodes in the emulation tree are positive integers. We conclude that the height of the emulation tree is at most $2 \cdot \lceil \frac{r(n)}{\log(n)-2} \rceil + 1$. ■

Claim 9 ***(Leaves)*** *The leaves of E_{P_0,V_0} are exactly the leaves of protocol tree T_{P_0,V_0} whose weights are 1.*

Proof. The construction does not create nodes whose weights are zero. Hence all the leaves in E_{P_0,V_0} have positive weight. Following the construction of the emulation tree we can see that, during each step, the weight of a leaf from the original protocol tree stays the same, whereas the weight of a non-leaf is the sum of the weights of the leaves that are descendants of it. Hence a leaf in E_{P_0,V_0} must be a leaf in T_{P_0,V_0}whose is 1.

On the other hand, if v is a leaf whose weight is 1 in T_{P_0,V_0}, then v appears in the emulation tree. This is because the only way nodes from the protocol tree are erased throughout the construction is if their weight is 0 (possibly after truncations in the middle of the construction). A leaf from T_{P_0,V_0} that appears in E_{P_0,V_0} must appear as a leaf. This is because the only way we add descendants to a node is when we add interval children, but we do not invoke Build_Interval on a leaf. ■

5 Public-coin emulation

Next, we describe the strategy of the designated prover P and verifier V in the new ("emulation") protocol. The strategy of the designated prover P for a yes-instance x uses the emulation tree E_{P_0,V_0} of x constructed in the previous section. The prover assists the verifier V in progressing down the emulation tree. On each iteration, the prover provides the descriptions of the children $v_1, \ldots, v_d$ of the current node u, which was sampled in the previous iteration. The verifier preforms validations on the list supplied by the prover (to be detailed below), and then samples one of the children for the next iteration according to its weight. The verifier does not have access to the emulation tree, and its validations consist of structural requirements on the emulation tree. On the last round the verifier checks that the full transcript, along with the sequence of coin tosses, leads the original verifier V_0 to accept.

The main difference between the public-coin emulation and the private-coin one is in the way a child of a node is chosen in each iteration. In the private-coin emulation the values of the verifier's private coin tosses determine which child is chosen. In contrast, in the public coin emulation V does not have private coins, hence it must choose a continuation based on the transcripts and the probability distributions suggested by P.

One of the structural validations that the verifier makes is that the nodes provided by the prover may be children of u in the emulation tree. For nodes in the original protocol tree, v is a child of u if and only if the transcript of v extends the transcript of u by one pair of messages, and thus v is a descendant of u if and only if the transcript of u is a proper prefix of the transcript of v. For nodes in the emulation tree the situation is more complex. If v is an interval child of u, then the transcript of v equals the transcript of u, and the range of v is a partial range of the range of u. If v is a heavy child of u, then the transcript of u is a proper prefix of the transcript of v. Furthermore, if $\gamma(u) = \alpha_1^u\beta_1^u, \ldots, \alpha_i^u\beta_i^u$, then α_{i+1}^v (the $i+1$ verifier's message in the transcript of v) should be in the range of u.

With these two cases in mind, we define the conditions required of the descriptions of two nodes u and v in order for v to be a descendant (not necessarily a child) of u in the emulation tree. We say that v is a **transcript descendant** of u if these required conditions hold.

Definition 10 ***(Transcript Descendant)*** *Denote by u and v nodes in the emulation tree with transcripts $\gamma(u) = \alpha_1^u\beta_1^u, \ldots, \alpha_i^u\beta_i^u$ and $\gamma(v) = \alpha_1^v\beta_1^v \ldots \alpha_j^v\beta_j^v$ and with range field $R(u)$*

and $R(v)$, respectively. We say that v is a transcript descendant of u if one of the following conditions hold:

i $\gamma(v) = \gamma(u)$ *and* $R(v) \subseteq R(u)$

ii $\gamma(u)$ *is a proper prefix of* $\gamma(v)$ *and* $\alpha_{i+1}^{v} \in R(u)$

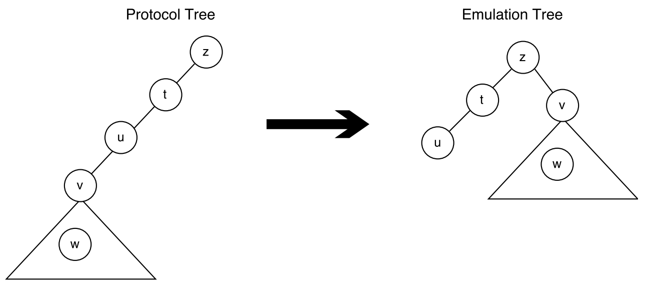

Figure 3: Truncation during Build_Tree. The node v is identified as a heavy descendant of z, so it is moved (along with the subtree that is rooted at it) to be a heavy child of z.

Note that it is possible that v is a descendant of u in the emulation tree and the description of u is equal to the description of v. For example, this can happen when v is the only interval child of u.

It is easy to show that for every node u in E_{P_0,V_0}, every descendant u is a transcript descendant of u. The proof is similar to the proof of (c) in the Completeness part of the analysis.

We state the following claim regarding the transitivity property of transcript descendancy, which we use in the analysis of the emulation.

Claim 11 ***(Transitivity)*** *For nodes u, v and z in the emulation tree such that z is a transcript descendant of v and v is a transcript descendant of u, then z is a transcript descendant of u.*

The proof of the claim is given in Appendix A. It follows by case analysis of the different transcript descendancy types between u, v and z.

The condition of v being a transcript descendant of u is not sufficient to guarantee that v is a descendant of u in the emulation tree E_{P_0,V_0}. For example, suppose that v was a child of u in T_{P_0,V_0}, and is a heavy descendant of a node z that is an ancestor of u in E_{P_0,V_0}. Then, v becomes a heavy child of z in E_{P_0,V_0}, which means that the subtree rooted at v was truncated and moved up to be a direct descendant of z. Therefore, in order to check if v may be a legal descendant of u in the emulation tree, the verifier needs to check that v does not belong to a part of the tree that was truncated and moved to a different part of the emulation tree (such as nodes v and w in Figure 3). For this reason the verifier keeps a list S of nodes that were seen during the emulation, and updates the list at every iteration with the new nodes seen. Note that the nodes in S, which the verifier sees up to some iteration, are a subtree of the emulation tree that is composed of a path from the root of the tree to the current input node, augmented with the children of the nodes in the path. See Figure 4.

Another structural validation requires the transcripts that the verifier sees to be consistent with a deterministic prover strategy.

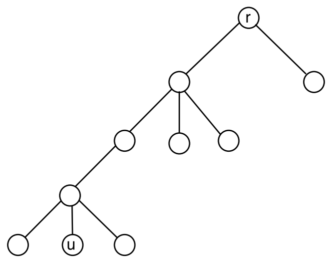

Figure 4: The nodes in the list of seen nodes, S, in the iteration that the input node is u.

Definition 12 *(Prover consistent) We say that two transcripts $\gamma(u)$ and $\gamma(v)$ are* ***prover consistent*** *if the maximal prefix they agree on is either empty or ends with a prover's message. That is, the prover should respond in the same way on the same prefix of the transcripts (i.e., for every j smaller than the length of the shorter transcript, if $\alpha_1^u\beta_1^u,\ldots,\alpha_j^u = \alpha_1^v\beta_1^v,\ldots,\alpha_j^v$ then $\beta_j^u = \beta_j^v$).*

This condition will allow for extracting a prover's strategy for the original protocol from the transcripts in E_{P_0,V_0}, and then to claim that since the original prover cannot fool the verifier with high probability, the new prover cannot either.

Initially, for the first iteration, the transcript of the root r is the empty transcript and the range is full range, $[0^\ell, 1^\ell]$. The prover provides the weight of the root r and the verifier checks that the claimed weight is at least $\frac{9}{10}\cdot 2^{r(n)}$. The verifier adds the description of r to the list of seen nodes S. The rest of the first iteration, as well as subsequent iterations, proceed as follows.

Construction 13 *(the ith iteration) On input a non-leaf node u and list S.* [5]

1. The prover provides the descriptions of the children $v_1,\ldots,v_d$ of u:
$$\big(\gamma(v_1),R(v_1),w'(v_1)\big),\ldots,\big(\gamma(v_d),R(v_d),w'(v_d)\big)$$
2. The verifier preforms the following validations and rejects if any of them fails:
   (a) The verifier checks that all nodes are different (according to their descriptions) that is, for each distinct $i,j\in[d]$, if $\gamma(v_i)=\gamma(v_j)$, then $R(v_i)\neq R(v_j)$.
   (b) The verifier checks that the weights of the children of u sum up to $w'(u)$; that is,
   $$w'(u)=\sum_{j=1}^{d} w'(v_j) \qquad (2)$$
   (c) For each $j\in[d]$, the verifier checks that v_j is a transcript descendant of u.
   (d) For each interval child v_j, the verifier checks that $\gamma(v_j)=\gamma(u)$; that is, if $R(v_j)\neq[0^\ell,1^\ell]$, then $\gamma(v_j)=\gamma(u)$ must hold.
   (e) For each $j\in[d]$ the verifier checks that v_j is not in a part of the emulation tree that was truncated. (See discussion following Definition 10.) Specifically, let $v\in S(u)$ such that v is not an ancestor of u. If v is a transcript descendant of u, then v_j

[5] Because of Step 4, the input node u will always be a non-leaf node.

should not be a transcript descendant of v.
For illustration consider Figure 3, where $u, t, z, v \in S$. In that case, the verifier checks that v_j is not a transcript descendant of v (where v is a transcript descendant of u since it was a descendant of u in the original protocol tree).
(Note that it can be that v_j is a transcript descendant of some node in S that is not a transcript descendant of u, and this is not considered a violation. For example, all the nodes are transcript descendants of the root r, which is in S.)

(f) The verifier checks that the ranges of all the interval children are disjoint; that is, for every two interval children v_j and v_k, the verifier checks that $R(v_j) \cap R(v_k) = \emptyset$.

(g) For each $j \in [d]$ the verifier checks that $\gamma(v_j)$ is **prover consistent** (see Definition 12) with respect to the other transcripts of nodes in S and with regarding to the transcripts of the other children $\gamma(v_k)$, where $k \neq j$.

3. The verifier chooses a child according to the probability distribution J that assigns each $j \in [d]$ probability approximately proportional to $w'(v_j)$ using $O(\log n)$ coin tosses. That is,

$$\mathbf{Pr}[J = j] \leq \frac{w'(v_j)}{\sum_{i=1}^{d} w'(v_i)} \cdot \left(1 + \frac{1}{n}\right) \tag{5.1}$$

We can only afford to use $O(\log n)$ public-coins per round, and hence we compromise on sampling each child with probability proportional to $w'(v_j)$, and instead sample with approximate probability. See explanation for approximate sampling in Appendix B .

4. The verifier adds all the children of u to the list S; that is $S \leftarrow S \cup \{v_1, \ldots, v_d\}$. Unless $\gamma(v_j)$ is the complete transcript (which contains the last message), the next iteration will start with node v_j and the set S. Otherwise, we proceed to the final checks.

By our conventions, the last message the verifier sends, denoted α_m, contains the outcomes $\rho \in \{0,1\}^{r(n)}$ of the $r(n)$ coins tossed. Thus, if the last node chosen is v, then ρ can be easily extracted from $\gamma(v) = \alpha_1\beta_1, \ldots, \alpha_m\beta_m$. After the last iteration the verifier performs final checks and accepts if all of them hold:

(i) Check that ρ is accepting for $\gamma(v)$ and consistent with it: It checks that $V_0(x, \rho, \beta_1, \ldots, \beta_m) = 1$, and that for every $i = 1, \ldots, m$ it holds that $\alpha_i = V_0(x, \rho, \beta_1, \ldots, \beta_{i-1})$. Note that the verifier needs ρ in order to verify these conditions, so this check can only be done after the last iteration. Also note that if these checks pass then $w(v) = 1$ (rather than $w(v) = 0$).

(ii) Check that $w'(v) = 1$; in other words the prover's last claim should be that the weight of the last node chosen is 1 (and not more than 1).

Clearly, the number of rounds of the emulation is $O(r(n)/\log n)$ because the height of the emulation tree is $O(r(n)/\log n)$, and the prover and verifier proceed one step down the tree on each round. Since the verifier uses $O(\log n)$ public coins on each round, the randomness complexity of the emulation is $O(r(n))$.

6 Analysis of the emulation

We show that the interactive proof system is transformed by the emulation protocol of Construction 13, which uses the emulation tree E_{P_0,V_0} constructed in Section 4.2, into a public-coin interactive proof system with perfect completeness and soundness $\frac{1}{3}$.

6.1 Completeness

We claim that the emulation protocol of Construction 13 has perfect completeness. That is, if x is a yes-instance, then V will accept at the end of the interaction with P.

Recall that P builds an emulation tree E_{P_0,V_0} from the protocol tree T_{P_0,V_0}. Since x is a yes-instance at least $\frac{9}{10} \cdot 2^{r(n)}$ of the coin tosses lead the verifier to accept and hence the weight of the root of T_{P_0,V_0} is at least $\frac{9}{10} \cdot 2^{r(n)}$. The weight of the root does not change during the construction of the emulation tree. Thus, the weight of the root of E_{P_0,V_0} is at least $\frac{9}{10} \cdot 2^{r(n)}$ as well. Hence, P can make a valid initial claim of weight at least $\frac{9}{10} \cdot 2^{r(n)}$

Next, we wish to show that the validations on Step 2 are satisfied for every iteration. This is equivalent to showing that validations are satisfied for every node in the final emulation tree E_{P_0,V_0}.

The general framework of the proof consists of going over every validation performed and showing that the property being checked holds for every node in the original protocol tree T_{P_0,V_0}, and continues to hold with every modification of the global tree T as part of the Build Tree procedure. Thus, the property also holds for every node in the final emulation tree E_{P_0,V_0}, and hence the validations are satisfied.

When we say that a validation passes relative to a (possibly intermediate) tree T and node u in T, we mean that if the tree T had been used as an emulation tree then, in the iteration on which u is the input node, the validation would have passed. Recall that the nodes in T on which we did not invoke the Build_Tree procedure yet do not have a range field. We regard the nodes that do not have a range field as having a full range $[0^\ell, 1^\ell]$. The children of u that are considered in the validation are the children of u in T, and the list of the seen nodes S consists of the ancestors of u and their children in T.

Let T be the global tree at some point in the construction, and let z be a node that the Build_Tree procedure is currently invoked on. We assume that the validation we are currently checking holds for every node in T, and show that it also holds in the next step of the construction. Recall that the next step can either be identifying a heavy child for z as part of the Build_Heavy procedure, or creating an interval child for z as part of the Build_Interval procedure. Denote the child being created or identified by v, and the global tree after this modification by T_v.

Note that it is not sufficient to show that after the creation or identification of v the property being checked is maintained for v. This is because the procedure might affect the descendants and ancestors of v in T_v, as well as nodes whose list of seen nodes changes. Exactly which nodes are effected depends on the validation.

Remark 14 *Let v be a node in T that the Build_Tree procedure has not been invoked on yet. Recall that the children of v in T are children of the node v' in T_{P_0,V_0} that satisfies $\gamma(v') = \gamma(v)$. Hence, like in the tree T_{P_0,V_0}, the transcripts of the children of v in T extend the transcript of v by one pair of messages. Furthermore, if we did not invoke Build_Tree on v yet, then we also did not invoke it on the descendants of v in T. Thus the subtree of T rooted at v, denoted by $T(v)$, is a subtree of T_{P_0,V_0}.*

Now, we go over the validations in Step 2, which are stated for a node u and its children $v_1, \ldots, v_d$ provided by the prover as part of the emulation. The validations are numbered as in Construction 13. We shall prove that these validations hold for every node in E_{P_0,V_0}.

Showing that the validations in Step 2 of the public-coin emulation are satisfied is similar in spirit to the proof that the validations in Step 2 of the private-coin emulation are satisfied, which is given in C.4.1.

(a) In this validation, the verifier checks that the descriptions of all the children of u are distinct, i.e. if v_i and v_j are two children of u provided by the prover and $\gamma(v_i) = \gamma(v_j)$ then $R(v_i) \neq R(v_j)$.
First note that in T_{P_0,V_0} all the nodes have distinct transcripts. Denote by T the global tree before we invoke Build_Tree on a node z. By Remark 14, the subtree rooted at z, denoted by $T(z)$, is a subtree of T_{P_0,V_0}. Hence, each node in $T(z)$ has a different description. When we determine a heavy child of z we are lifting descendants of z in $T(z)$ to be children of z. Hence, before invoking Build_Interval(z) all the children of z are different. When we determine interval children for z the transcripts of the interval children are equal to the transcript of z and their range fields are disjoint, so in particular the descriptions of the interval children of z are all distinct.

(b) In this validation the verifier checks that the weights of the children $v_1, \ldots, v_d$ of u sum up to $w'(u)$; that is,

$$w'(u) = \sum_{j=1}^{d} w'(v_j) \tag{6.1}$$

We shall show that this property holds in every step of the construction of the emulation tree E_{P_0,V_0}, for every node in the tree. Starting from the protocol tree T_{P_0,V_0}, we know that by definition it satisfies the property that the weight of each node is the sum of the weights of its children. (Recall that z is a node that we are invoking the Build_Tree procedure on, and v a node that we determine as a child for z.) Denote the tree before creation of v by T, and after by T_v. Assuming that Eq. (6.1) holds for every node in T, we shall show that it holds for every node in T_v as well. We consider to two cases, according to if v is a heavy child or interval of z. (Recall that these are the only cases since after invoking Build_Tree(z) every child of z is either a heavy or interval child.)

- If v is a new heavy child determined for z, denote by $z, z_1, \ldots, z_k = v$ the path from z to v in T before v is moved to be a child of z. In this case, after moving v we subtract $w(v)$ off the weight of $z_1, \ldots, z_{k-1}$. Hence, Eq. (6.1) holds for these nodes (we subtract $w(v)$ both from their weight and from the weight of one of their children). Eq. (6.1) also holds for z, since its weight stays the same, whereas we subtract $w(v)$ off the weight of z_1, but we add an additional child v with weight $w(v)$. For the other nodes in T_v we neither change their weight nor the weights of their children.
- If v is a new interval child of z, then the weight of v is the sum of the weights of its children. Eq. (6.1) also holds for z since its weight stays the same and the sum of the weights of its children also stays the same.

Lastly, we note that the clean-up stage, in which we remove nodes whose weights are 0, does not affect this validation.

(c) In this validation, for each $j \in [d]$, the verifier checks that the child v_j provided by the prover is a transcript descendant of u.
Recall that we consider nodes in the protocol tree that do not have a range field (yet) as having a full range. In the protocol tree T_{P_0,V_0} the transcript of each node is a proper prefix of the transcript of its children. Hence every node in the protocol tree is a transcript descendant of type (ii) (see Definition 10) of its parent. As before, we shall assume that every node in T maintains the property that it is a transcript descendant of its parent, and show that this property is maintained in the tree T_v, after the creation of v. We consider two cases:

- If v is a heavy child of z, then v was a descendant of z in T that we move to be a child of z (along with its descendants) and set its range to be full range. By our hypothesis, each node in the path from z to v in T is a transcript descendant of its parent, so by transitivity (see Claim 11), v is a transcript descendant of z.
- If v is an interval child of z then after the creation of v the transcript of v is equal to the transcript of z, and by the construction we know that $R(u) \subseteq R(z)$. Thus, v is a transcript descendant of z of type (i).
In addition, the children of v are transcript descendants of v. This is because the children of v are all children of z in T, such that their next verifier message is in the range of v. Hence the children of v are transcript descendants of of type (ii) of v . The rest of the nodes in T_v are nodes in T with the same children as in T_v. Hence from our hypothesis on T the property holds for these nodes in T_v.

(d) In this validation the verifier checks, for each interval child v_j of u, that the transcript of v_j is equal to the transcript of u; that is, if $R(v_j) \neq [0^\ell, 1^\ell]$ then $\gamma(v_j) = \gamma(u)$ must hold.
This holds because the only case where $R(v_j) \neq [0^\ell, 1^\ell]$ is if v_j is an interval child of u, and in this case $\gamma(v_j) = \gamma(u)$.

(e) In this validation the verifier checks, for each child v_j of u, that v_j is not in a part of the emulation tree that was truncated. Recall that $S(u)$ is the set of nodes the prover provided up to the iteration that is executed on input node u. Let $y \in S(u)$ such that y is not an ancestor of u. Then the verifier checks that y is a transcript descendant of u, then v_j should not be a transcript descendant of y.
In order to show that the validation is satisfied, we shall prove that the following claim holds for every node in the emulation tree E_{P_0,V_0}.

Claim. *Let $u \in E_{P_0,V_0}$ and $y \in S(u)$ that is not an ancestor of u in E_{P_0,V_0}. If y is a transcript descendant of u, then each* descendant *v_j of u in E_{P_0,V_0} is not a transcript descendant of y.*

Note this is a stronger claim then what we need to show because we only need to show it for the children of u in E_{P_0,V_0} and not for each descendant of u.

Recall that when consider a tree T that is not the final emulation tree, and some node $v \in T$, then we regard the set $S(v)$ as the set containing the nodes that are ancestors of v, augmented with their children.

First, we shall show that the claim holds in the initial protocol tree T_{P_0,V_0}. Each node is T_{P_0,V_0} only a transcript descendant of its ancestors in the tree. However, each node u is not an ancestor of any node in $S(u)$, so the nodes in $S(u)$ cannot be transcript descendants of u. Next, we shall assume that the claim holds for the tree T before creating a child v of z, and we show that it holds in the tree T_v after the identification v as a heavy child of u, or creation of v as an interval child.

- If v is identified as a heavy descendant of z, then v is moved to be a child of z along with the subtree under it. In order to show that the claim holds in T_v, it is enough to consider the nodes $c \in T_v$ who have new descendants or new nodes in $S(c)$ relative to the ones they had in T. The descendants of the nodes in T_v are descendants of it in T.

  The only nodes in T_v that have new nodes in their seen list are the descendants of z, since now v is in their seen list, whereas v may not have been in their seen list before the move. Since determining the heavy descendants of z is done bottom up, v is only a transcript descendant of its ancestors in T. Thus, if c is a node in T_v such that $v \in S(c)$ and v is a transcript descendant of c, then c is an ancestor of v in T. It is left to check that the descendants of c in T_v are not transcript descendants of v. From Note 14, it follows that the subtree of T_v rooted at c, denoted bt $T_v(c)$, is a subtree of T_{P_0,V_0}. Thus, because $v \notin T_v(c)$ (recall that v was lifted to be a heavy child of z, and c is a descendant of z) it follows that the descendants of c are not transcript descendants of v. (See Figure 5).

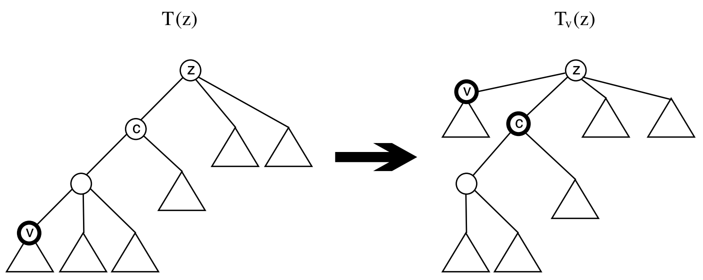

Figure 5: v is identified as a heavy child of z, and c is a descendant of z.

- Let v be an interval child created for z. We shall assume that the claim holds in the tree T before the creation of v, and show that it holds in T_v. It is enough show that the claim holds when the node u from the claim is one of the following: the new interval child v, a node in T_v that has new descendants or has new nodes in their seen list (relative to the ones in T). (For the other nodes in T_v the claim follows from our hypothesis regarding T.)
  The only nodes in T_v that have new descendants that they did not have in T are the ancestors of v in T_v, which now have v as their descendant. Let d be some ancestor of v in T_v, $c \in S(d)$ a transcript descendant of d. We need to show that v is not a transcript descendant of c. Let t be a child of v in T_v (See Figure 6). Assume in contradiction that v is a transcript descendant of c. Hence, by transitivity, t is also transcript descendants of c. In addition, t is a descendant of d in T. This is in contradiction to our hypothesis that the claim holds in T.

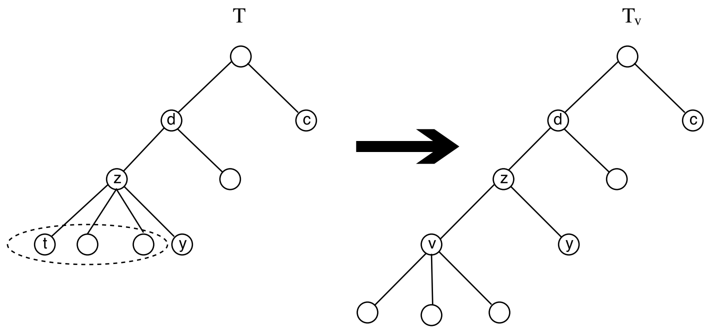

Figure 6: v is an interval child created by uniting 3 children of z.

The nodes whose list of seen nodes S increases in T_v relative to T are the descendants of z, because v is added to their list. However, v cannot be a transcript descendant of any descendant of z. This is because the heavy children of z have longer transcripts than v, so v cannot be a transcript descendant of them, or of their descendants. The other interval children of z have a range that is disjoint from the range of v. Thus, they and their descendants cannot be transcript descendants of v.
Lastly, we show that the claim holds for v as well. Let there be a some node $b \in S(v)$ which is a transcript descendant of v. We need to show that the descendants of v in T_v are not transcript descendants of b. There are two different options.

– Either $b \in S(z)$ in T, like node c in Figure 6. The node b is a transcript descendant of v which is a transcript descendant of z, so by transitivity b is a transcript descendant of z. Every descendant of v in T_v is a descendant of z in T. Hence from the fact that the claim is true for z in T we know that the descendants of v in T_v are not transcript descendants of b.
– If $b \in S(v)$ but $b \notin S(z)$ then b must be a child of z, like node y in Figure 6. Since b is a transcript descendant of v then b must be a heavy child of z. (If b is an interval child of z, like v is, then b cannot be a transcript descendant of v since their ranges are disjoint.) We did not invoke Build_Tree on v yet, so by Note 14, it follows that $T_v(v)$ is a subtree of T_{P_0,V_0}. Hence, from the fact that b is not in $T_v(v)$ it follows that the transcript descendants of b are not in $T_v(v)$ either. In other words, the descendants of v in T_v are not transcript descendants of b.

(f) In this validation, the verifier checks that the ranges of all the interval children of u are disjoint; that is, for every two interval children v_j and v_k of u, the verifier checks that $R(v_j) \cap R(v_k) = \emptyset$.
By the way we create the interval children it holds that the start of the range of each interval child is after the end of the range of the previous child created.

(g) In this validation, the verifier checks, for each child v_j of u, that $\gamma(v_j)$ is *prover consistent* (see Definition 12) with respect to the other transcripts of nodes in $S(u)$ and with regarding to the transcripts of the other children of u.
In the original protocol tree, T_{P_0,V_0}, every two nodes are prover consistent since P_0 is deterministic. (If there were two partial transcripts in T_{P_0,V_0} whose maximal common

prefix ends with a verifier message its would mean that the prover can responds in different ways to the same partial transcripts). The transcripts of the nodes in E_{P_0,V_0} all appear in T_{P_0,V_0}, so every two transcripts in E_{P_0,V_0} are prover consistent as well.

The final checks are satisfied because according to Claim 9 the leaves of the emulation tree E_{P_0,V_0} are the leaves of the protocol tree whose weights are 1. Hence, $V_0(x, \rho, \alpha_1, \ldots, \alpha_m) = 1$, and for every $i = 1, \ldots, m$ it holds that $\alpha_i = V_0(x, \rho, \beta_1, \ldots, \beta_{i-1})$. In addition $w'(v) = w(v) = 1$.

6.2 Soundness

We show that if x is a no-instance, then when interacting with any prover $\widetilde{P}$ for the public-coin emulation protocol, the new public-coin verifier V accepts with probability at most $\frac{1}{3}$. We do so by showing that on each iteration there is a gap (in expectation) between the weight of the node claimed by the prover, and the actual weight of the node. Starting from the root, if x is a no-instance, then the initial prover's claim is that the weight of the node sampled should be at least $\frac{9}{10} \cdot 2^{r(n)}$ (or else the verifier rejects upfront), but the number of coin sequences that lead the verifier to accept at the end of the interaction, and hence the real weight of the node, is at most $\frac{1}{10} \cdot 2^{r(n)}$. We want to show that this gap is maintained with high probability until the last iteration, and hence the verifier rejects.

In order to proceed with this analysis we need to define the notion of "real weight of a node" in the emulation tree. We do this by considering the weight relative to the protocol tree of the original interactive proof system. The verifier of the original protocol system, which we refer to, is of course V_0, the verifier of the original interactive proof system being emulated. However, choosing the prover of the original system is less straight forward. We shall show that a prover's strategy for the emulation protocol yields a prover strategy for the original protocol.

Subsection 6.2.1 is a more involved version of the private-coin proof of soundness in Section C.4.2. The other two components of the soundness analysis of the public-coin system (which are defining the notion of real weight of a node, and the actual proof of soundness) are not required for the private-coin soundness analysis.

6.2.1 Deriving a prover strategy for the original proof system

We can assume without loss of generality that $\widetilde{P}$ is deterministic since for every probabilistic prover there is a deterministic prover for which the verifier's rejection probability is at least as high (recall that we want to show that the verifier rejects with high probability). We show that we can extract a deterministic strategy $\widetilde{P_0}$ for the original prover using the strategy of $\widetilde{P}$. Denote by $E_{\widetilde{P}}$ the emulation tree of the prover $\widetilde{P}$. We define a strategy for $\widetilde{P_0}$ by using the transcripts in $E_{\widetilde{P}}$. That is, for each $u \in E_{\widetilde{P}}$ with transcript $\gamma(u) = \alpha_1\beta_1 \ldots, \alpha_j\beta_j$ we define $\widetilde{P_0}(x, \alpha_1, \ldots, \alpha_i) := \beta_i$ for all $i \le j$. We extend $\widetilde{P_0}$'s strategy to transcripts that do not appear in $E_{\widetilde{P}}$ in an arbitrary way.

In order to show that the strategy of $\widetilde{P_0}$ is well defined we need to show that no two nodes $u, v \in E_{\widetilde{P}}$ share the prefix of prover-verifier interaction but differ on the prover's response. That is, we show that every two nodes $u, v \in E_{\widetilde{P}}$ are **prover consistent** (see Definition 12).

For a node $u \in E_{\widetilde{P}}$ provided during the emulation, denote by $S(u)$ the list of seen nodes from $E_{\widetilde{P}}$ at the beginning of the iteration in which u was the input node (i.e. was the node handled on that iteration). That is, the nodes in $S(u)$ are the ancestors of u in $E_{\widetilde{P}}$ and their children. We denote by $S'(u)$ the list of seen nodes of the parent of u. That is, if v is the parent of u in $E_{\widetilde{P}}$ then $S'(u) = S(v)$.

We can assume, without loss of generality, that the strategy of $\widetilde{P}$ is such that the verifier does not abort until the final checks. This is because every prover strategy in which the verifier aborts in one of the intermediate checks can be modified to a prover strategy such that the verifier does not abort until the final checks and the verifier's acceptance probability is at least as large.

To show that every two nodes $u, v \in E_{\widetilde{P}}$ are prover consistent, as well as for other parts of the soundness proof, we rely on the following claim.

Claim 15 ***(Transcript descendancy forms a tree)*** *Let $w \in E_{\widetilde{P}}$ and $u, z \in S(w)$. Assume that the strategy of $\widetilde{P}$ is such that verifier does not abort until the final checks. Let ℓ be a node that is a transcript descendant of both z and u. Then either u is a transcript descendant of z or z is a transcript descendant of u.*

Proof. First note that this claim is not true if the prover's strategy is not one in which the verifier does not abort until the final checks. For example, if ℓ is a transcript descendant of type (i) of both u and z, then $\gamma(u) = \gamma(z) = \gamma(\ell)$, and $R(\ell)$ is contained in both $R(u)$ and in $R(z)$. However, $R(u)$ and $R(z)$ may not be contained one in the other, and hence u is not a transcript descendant of z and z is not a transcript descendant of u.

Since ℓ is a transcript descendant of u and of z, then both $\gamma(u)$ and $\gamma(z)$ are prefixes of $\gamma(\ell)$. Assume, without loss of generality, that $|\gamma(z)| \geq |\gamma(u)|$. It follows that $\gamma(u)$ is a prefix of $\gamma(z)$. (See Figure 7).

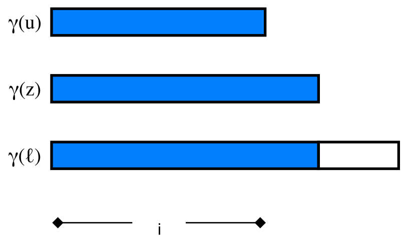

Figure 7: The transcripts of u,z and ℓ when $|\gamma(z)| \geq |\gamma(u)|$

The first case is that $\gamma(u)$ is a strict prefix of $\gamma(z)$. In this case $\gamma(u)$ is also a strict prefix of $\gamma(\ell)$ and hence ℓ is a transcript descendant of type (ii) of u . Denote the transcripts of u and z by $\gamma(u) = \alpha_1\beta_1 \dots \alpha_i\beta_i$ and $\gamma(z) = \alpha_1\beta_1 \dots \alpha_j\beta_j$, for $j > i$. Because $\gamma(z)$ is a prefix of $\gamma(\ell)$ it follows that $\alpha^z_{i+1} = \alpha^\ell_{i+1}$. From the fact that ℓ is a transcript descendant of type (ii) of u we know that $\alpha^\ell_{i+1} \in R(u)$. Thus, $\alpha^z_{i+1} \in R(u)$ and $\gamma(u)$ is a strict prefix of $\gamma(z)$, so z is a transcript descendant of u (of type (ii)).
The second case is that the transcripts of u and z are equal; that is, $\gamma(u) = \gamma(z) = \alpha_1\beta_1 \dots \alpha_i\beta_i$. In this case we need to show that $R(u) \subseteq R(z)$ or $R(z) \subseteq R(u)$. If $R(u) = [0^\ell, 1^\ell]$ then $R(z) \subseteq R(u)$, so z is a transcript descendant of u of type (i). Similarly, if $R(z) = [0^\ell, 1^\ell]$ then u is a transcript descendant of z of type (i).

We are left with the case that $\gamma(u) = \gamma(z)$ and both u and z do not have a full range. In this part we consider the relation between u and z in $E_{\widetilde{P}}$. Recall that $S(w)$ contains the nodes in the path from the root to w, augmented with their children. Denote the least common ancestor of u and z in $E_{\widetilde{P}}$ by v. If $v = u$ then there is a path from v to z. Since the validations pass, each node in the path is a transcript descendant of its predecessor, so from transitivity z is a transcript descendant of v. Similarly, if $v = z$ then u is a transcript descendant of z.

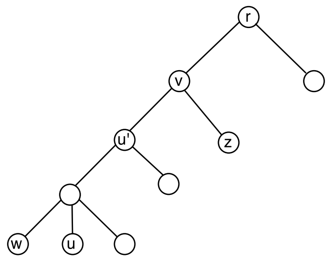

Figure 8: z and u are two nodes in $S(w)$ whose least common ancestor is v.

Otherwise, v is neither equal to u nor to z. However, at least one of the nodes u or z is a child of v in $E_{\widetilde{P}}$ (because of the structure of $S(w)$, see Figure 8 for illustration). Assume, without loss of generality, that z is a child of v. The range of z is not full so z must be an interval child of v and $\gamma(v) = \gamma(z)$. It follows that $\gamma(u) = \gamma(v)$ so u is either another interval child of v or a descendant of an interval child of v. Thus, the ranges of u and z are disjoint because they belong to different branches of interval children of v, and the ranges of the interval children are disjoint from validation 2f. This is in contradiction to the assumption in the claim that $\gamma(\ell)$ is a transcript descendant of both $\gamma(u)$ and $\gamma(z)$. (If $\gamma(\ell) = \gamma(u) = \gamma(z)$ then ℓ is a transcript descendant of type (i) of both u and z and hence $R(\ell) \subseteq R(u)$ and $R(\ell) \subseteq R(z)$, and so $R(u)$ and $R(z)$ cannot be disjoint. If $\gamma(u) = \gamma(z)$ are strict prefixes of $\gamma(\ell)$ then ℓ is a transcript descendant of u and z of type (ii). Hence $\alpha^{\ell}_{i+1} \in R(u)$ and $\alpha^{\ell}_{i+1} \in R(z)$, so also in this case $R(u)$ and $R(z)$ cannot be disjoint.) ■

We are ready to prove the prover consistency property of the emulation tree.

$\boxed{\text{Note that the following proof is a more involved version of the proof of Lemma 30 in Section C.4.}}$

Lemma 16 *(Prover consistency of the emulation tree). If $\widetilde{P}$ is a prover strategy for the new emulation such that the verifier V does not abort until the final checks then every two transcripts of nodes in the emulation tree $E_{\widetilde{P}}$ are prover consistent.*

Proof. Let u and v two nodes in the emulation tree $E_{\widetilde{P}}$, we wish to show that their transcripts $\gamma(u)$ and $\gamma(v)$ are prover-consistent. If one of the nodes is a descendant of the other node in the emulation tree, with out loss of generality, we assume that v is a descendant of u. In this case $u \in S'(v)$ and validation 2g is satisfied so $\gamma(u)$ and $\gamma(v)$ are prover-consistent. Otherwise, denote by z the least common ancestor of u and v, and a and b the children of z that are ancestors of u and v respectively. (It is possible that $a = u$ or $b = v$.) See Figure 9 for illustration.

Consider the case that at least one of the transcripts of u and v equals the transcript of a or b, respectively (this also covers the case that u or v are children of z). Assume, without loss

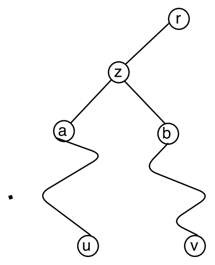

Figure 9: The subtree of E_{P_0,V_0} that contains u and v

of generality, that $\gamma(a) = \gamma(u)$. We know that $\gamma(v)$ and $\gamma(a)$ are prover consistent, because $a \in S'(v)$ and validation 2g is satisfied. Hence, $\gamma(v)$ and $\gamma(u)$ are also prover consistent.

Otherwise, $\gamma(a)$ is a proper prefix of $\gamma(u)$, and $\gamma(b)$ is a proper prefix of $\gamma(v)$. We consider three cases according to the relation between $\gamma(a)$ and $\gamma(b)$.

First, consider the case that $\gamma(a)$ is not a prefix of $\gamma(b)$ and $\gamma(b)$ is not a prefix of $\gamma(a)$. The fact that $\gamma(a)$ is a proper prefix of $\gamma(u)$ implies that u is a descendant of a in $E_{\widetilde{P}}$. By validation 2c we know that each node in this path from u to a is a transcript descendant of its parent. Thus, by transitivity (Claim 11), u is a transcript descendant of a, so $\gamma(a)$ is a prefix of $\gamma(u)$. Similarly, $\gamma(b)$ is a prefix of $\gamma(v)$. Recall that we are in the case that $\gamma(a)$ is not a prefix of $\gamma(b)$ and $\gamma(b)$ is not a prefix of $\gamma(a)$, and so the maximal prefix on which $\gamma(a)$ and $\gamma(b)$ agree upon is a proper prefix of both. This common prefix equals the maximal prefix on which $\gamma(u)$ and $\gamma(v)$ agree. We know that $\gamma(a)$ and $\gamma(b)$ are prover-consistent because the prover provides a along with b as children of z and we assume that validation 2g is satisfied. Since the maximal prefix that $\gamma(u)$ and $\gamma(v)$ agree on is equal to the maximal prefix that $\gamma(a)$ and $\gamma(b)$ agree on, it follows that $\gamma(v)$ and $\gamma(u)$ are also prover-consistent. See Figure 10 for illustration.

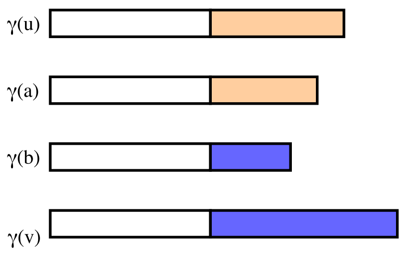

Figure 10: $\gamma(a)$ and $\gamma(b)$ agree on the prefix in white, while $\gamma(a)$ is a prefix of $\gamma(u)$ and $\gamma(b)$ is a prefix of $\gamma(v)$.

Next, consider the case when $\gamma(a) = \gamma(b)$, and assume that the transcript of a and b contain messages from i rounds. It this case both a and b are interval children of z and from validation 2f it follows that $R(a) \cap R(b) = \emptyset$. Recall that u is a transcript descendant of a, and v is a transcript descendant of b, and hence $\alpha^v_{i+1} \in R(b)$ and $\alpha^u_{i+1} \in R(a)$. It follows that $\alpha^v_{i+1} \neq \alpha^u_{i+1}$, which means that the maximal prefix that $\gamma(u)$ and $\gamma(v)$ agree on is equal to $\gamma(a) = \gamma(b)$. Hence, the maximal prefix ends with a prover message and so $\gamma(u)$ and $\gamma(v)$ are prover consistent.

We are left with the case that one of the transcripts $\gamma(a)$ and $\gamma(b)$ is a proper prefix of the other, and assume, without loss of generality, that $\gamma(a)$ is a proper prefix of $\gamma(b)$. Denote the transcript of b by $\gamma(b) = \alpha_1\beta_1, \ldots, \alpha_k\beta_k$. Since $\gamma(a)$ is a proper prefix of $\gamma(b)$, and $\gamma(z)$

is a prefix of both $\gamma(b)$ and $\gamma(a)$ (since they are transcript descendants of z) it follows that $\gamma(b) \neq \gamma(z)$ and so b is not an interval child of z and hence $R(b) = [0^\ell, 1^\ell]$.

We claim that u is not a transcript descendant of b. Assume, in contradiction, that u is a transcript descendant of b. Note that a is not a transcript descendant of b, since we assumed that $\gamma(a)$ is a proper prefix of $\gamma(b)$. Denote the nodes in the path from a to u in $E_{\widetilde{P}}$ by $a = a_0, a_1, \ldots, a_k = u$ and by a_j the first node in the path such that a_j is not a transcript descendant of b and a_{j+1} is a transcript descendant of b. Since a_{j+1} is a transcript descendant of both a_j and b, which are in $S(a_{j+1})$, and since a_j is not a transcript descendant of b, it follows by Claim 15 that b is a transcript descendant of a_j (see Figure 11). Hence, in the iteration where the prover provides node a_{j+1} as a child of a_j there is a violation to validation 2e, because a_{j+1} is a transcript descendant of $b \in S(a_j)$ and b is a transcript descendant of a_j. Put differently, a_{j+1} is part of a truncation from a_j. Hence, we reached a contradiction to the hypothesis that V does not abort in the intermediate validations, and so u cannot be a transcript descendant of b.

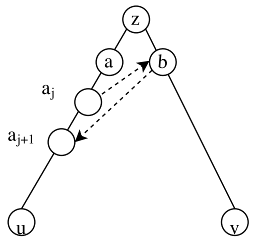

Figure 11: The dashed arrow pointing from b to a_{j+1} represent the fact that a_{j+1} is a transcript descendant of b, and similarly that b is a transcript descendant of a_j.

We showed that u is not a transcript descendant of b and $R(b) = [0^\ell, 1^\ell]$, so $\gamma(b)$ is not a prefix of $\gamma(u)$. (If $\gamma(b)$ is a proper prefix of $\gamma(u)$ then u is a transcript descendant of b of type (ii) since $R(b) = [0^\ell, 1^\ell]$, whereas if $\gamma(b) = \gamma(u)$ then $R(u) \subseteq [0^\ell, 1^\ell] = R(b)$, which means that u is a transcript descendant of type (i) of b.) Let u' be the parent of u in E_{P_0,V_0}. We know that $b \in S(u')$ because b is a child of z, which is an ancestor of u'. Hence by validation 2g the transcript of u, which is a child of u' and the transcript of $b \in S(u')$ are prover consistent. It follows that the maximal common prefix of $\gamma(u)$ and $\gamma(b)$ is a proper prefix of $\gamma(b)$ that ends with a prover message.

Lastly, note that from validation 2c each node in the path from b to v is a transcript descendant of its parent, so from transitivity v is a transcript descendant of b. Thus, $\gamma(b)$ is a prefix of $\gamma(v)$. It follows that the maximal common prefix of $\gamma(v)$ and $\gamma(u)$ is contained in the maximal common prefix of $\gamma(u)$ and $\gamma(b)$, so it ends with a prover message (See Figure 12). ∎

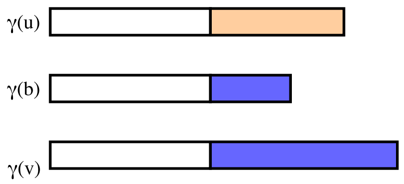

Figure 12: The maximal common prefix between $\gamma(u)$ and $\gamma(b)$ appears in white. Since $\gamma(b)$ is a prefix of $\gamma(v)$ the maximal common prefix of $\gamma(u)$ and $\gamma(v)$ is the same as the former.

6.2.2 Real weight of a node in $E_{\widetilde{P}}$

We want to introduce the notion of the "real weight" of a node in $E_{\widetilde{P}}$. This "real weight' should capture the probability that the node represents a partial interaction of the verifier V_0 with the prover $\widetilde{P_0}$, which was extracted in the previous subsection, that leads V_0 to accept. Hence, it is natural to use the weights in $T_{\widetilde{P_0},V_0}$ for this value. However, the weights in $T_{\widetilde{P_0},V_0}$ do not reflect the truncations the prover $\widetilde{P}$ makes. Thus, we also need to use the tree $E_{\widetilde{P}}$, which contains the information about the truncations. Another difficulty that arises is that the nodes in $T_{\widetilde{P_0},V_0}$ are only interval nodes. Hence, instead of using the weight of a node in $T_{\widetilde{P_0},V_0}$ we consider the leaves whose weights are 1 and are transcript descendants of it.

Definition 17 ***(Real weight of*** $u \in E_{\widetilde{P}}$ ***relative to protocol tree*** $T_{\widetilde{P_0},V_0}$***)*** *For a node* $u \in E_{\widetilde{P}}$ *denote by* $L^1(u)$ *the set of leaves in* $T_{\widetilde{P_0},V_0}$ *that are transcript descendants of* u *and whose weight is* 1. *(The definition of transcript descendants relates to nodes that have a transcript and range field, so we consider the nodes in* $T_{\widetilde{P_0},V_0}$ *as if they have full range* $[0^\ell, 1^\ell]$.*)*
Denote by $Trunc(u)$ *the non-ancestors of* u *in* $S(u)$ *(relative to the emulation tree* E_{P_0,V_0}*) that are transcript descendants of* u.
We define $L^{\widetilde{P}}(u)$ *as*

$$L^{\widetilde{P}}(u) = \left\{ L^1(u) \Big\backslash \bigcup_{z \in Tranc(u)} L^1(z) \right\}$$

We define the real weight of u*, denoted by* $W^{\widetilde{P}}(u)$*, to be the size of the set* $L^{\widetilde{P}}(u)$*. That is,*

$$W^{\widetilde{P}}(u) = \left| L^{\widetilde{P}}(u) \right|$$

For the soundness proof we use the following claim regarding the real weight of a node in the emulation tree $E_{\widetilde{P}}$ which asserts that the real weight of a node is at least as large as the sum of the weights of its children.

Claim 18 *Assume that the strategy of the prover* $\widetilde{P}$ *is such that the verifier does not abort until the final checks. Let* $u \in E_{\widetilde{P}}$ *and* $v_1, \ldots, v_d$ *children of* u *in* $E_{\widetilde{P}}$*. Then*

$$\sum_{i=1}^{d} W^{\widetilde{P}}(v_i) \leq W^{\widetilde{P}}(u) .$$

Proof. The proof follows from the following two facts:

Fact 19 *For each distinct* $i, j \in [d]$*, it holds that* $L^{\widetilde{P}}(v_j) \cap L^{\widetilde{P}}(v_i) = \emptyset$.

Fact 20 *For each* $j \in [d]$*, it holds that* $L^{\widetilde{P}}(v_j) \subseteq L^{\widetilde{P}}(u)$.

We start with the proof of Fact 19. Consider two different cases. The first case is if v_i and v_j are not transcript descendants of each other (v_i is not a transcript descendant of v_j and vice versa). If $\ell \in L^{\widetilde{P}}(v_j)$, then ℓ is a transcript descendant of v_j. Assume in contradiction that $\ell \in L^{\widetilde{P}}(v_i)$. Hence, ℓ is also a transcript descendant of v_i, and v_i and v_j are in $S(v_j)$.

From Claim 15 one of v_i and v_j is a transcript descendant of the other, in contradiction to the our assumption for this case.

The second case is that v_j (respectively v_i) is a transcript descendant of v_i (respectively v_j). Without loss of generality, assume that v_j is a transcript descendant of v_i. In this case $v_j \in Tranc(v_i)$ since $v_j \in S(v_i)$ and v_j is not an ancestor of v_i. From the fact that $\ell \in L^{\widetilde{P}}(v_j)$ it follows that $\ell \in L^1(v_j)$. By the definition of $L^{\widetilde{P}}$ the leaves in $L^1(v_j)$ are removed from the set $L^{\widetilde{P}}(v_i)$, and hence $\ell \notin L^{\widetilde{P}}(v_i)$. This completes the proof of Fact 19.

Turning to the proof of Fact 20, let $\ell \in L^{\widetilde{P}}(v_j)$. Therefore, the leaf ℓ is a transcript descendant of v_j. From validation 2c it follows that v_j is a transcript descendant of u. Therefore, by transitivity, ℓ is a transcript descendant of u so $\ell \in L^1(u)$. Assume in contradiction that there exists $z \in Tranc(u)$ such that $\ell \in L^1(z)$ and hence $\ell \notin L^{\widetilde{P}}(u)$. Recall that $z \in Tranc(u)$ means that z is a transcript descendant of u and $z \in S(u)$ (See Figure 13). This also means that $z \in S(v_j)$ since $S(u) \subseteq S(v_j)$. It follows that ℓ is a transcript descendant of both z and v_j which are in $S(v_j)$. Hence, from Claim 15 there are two options:

- The first option is that v_j is a transcript descendant of z. However, this cannot happen since in the iteration where u is the input node, by validation 2e it cannot be that v_j is a transcript descendant of $z \in S(u)$ and z is a transcript descendant of u.
- The second option is that z is a transcript descendant of v_j. This cannot be the case because $z \in S(u)$ implies that $z \in S(v_j)$ and so $z \in Tranc(v_j)$. But because $\ell \in L^1(z)$ we get that $\ell \notin L^{\widetilde{P}}(v_j)$, in contradiction to our hypothesis.

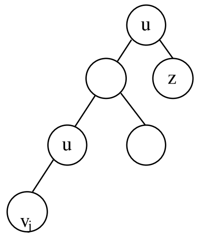

Figure 13: The transcripts of u,z and ℓ when $|\gamma(z)| \geq |\gamma(u)|$

Thus we reach a contradiction in both options, so in particular there does not exist $z \in Tranc(u)$ such that $\ell \in L^1(z)$. Hence, $\ell \in L^{\widetilde{P}}(u)$. ■

6.2.3 The actual proof of soundness

Next, we define the gap between the real weight and the claimed weight, which we use for the analysis.

Definition 21 ***(Gaps) The gap for vertex*** *$u \in S$ denoted by $g(u)$, is the ratio between $W^{\widetilde{P}}(u)$, the real weight of node u according to the strategy of $\widetilde{P}$, and the claimed weight $w'(u)$.*

$$g(u) = \frac{W^{\widetilde{P}}(u)}{w'(u)}$$

Note that we can assume without loss of generality that $w'(u) > 0$ since the prover can omit the nodes with claimed weight equal to 0.

If u is the note chosen on the ith iteration we denote the gap on the ith iteration by $g_i = g(u)$, and by g_0 the gap of the root r.

We consider a m' round emulation protocol defined in Construction 13, and fix an iteration $i \in [m']$ as well as the values of the coin tosses, denoted by $r_1, \ldots, r_{i-1}$, obtained during the emulation of the first $i-1$ iterations. Denote by u the node sampled on iteration $i-1$. If $i = 1$ then u is the root of the emulation tree. The description of the node u and $g_{i-1} = g(u)$ are fixed. Denote by G_i the random variable that represents g_i at the end of the ith iteration, which depends on the child of u chosen on the ith iteration. Towards proving the claim, we analyze the change in the gap on the ith iteration, and show that it does not increase too much in expectation.

Claim 22 *Consider a prover strategy for the proposed public-coin emulation in which the verifier does not abort until the final checks. For any sequence of values of the coin tosses* $r_1, \ldots, r_{i-1}$, *it holds that*

$$\mathbb{E}_{r_i}[G_i |\, r_1, \ldots, r_{i-1}] \leq g_{i-1} \cdot \left(1 + \frac{1}{n}\right)$$

Proof. Let $v_1, \ldots, v_d$ be the children of u, that were provided by the prover in the emulation. Since G_i is the gap after the ith iteration, its value dependents on the child of u that was chosen

$$\mathbb{E}[G_i |\, r_1, \ldots, r_{i-1}] = \sum_{j=1}^{d} \mathbf{Pr}[v_j \text{ chosen}] \cdot g(v_j) \ . \tag{6.2}$$

By the definition of the gap for node v_j,

$$g(v_j) = \frac{W^{\widetilde{P}}(v_j)}{w'(v_j)} \ . \tag{6.3}$$

According to step 3 of the emulation protocol,

$$\mathbf{Pr}[v_j \text{ chosen}] \leq \frac{w'(v_j)}{\sum_{i=1}^{d} w'(v_i)} \cdot \left(1 + \frac{1}{n}\right) . \tag{6.4}$$

Plugging in Eq. (6.3) and Eq. (6.4) in Eq. (6.2) we have

$$\begin{aligned} \mathbb{E}[G_i |\, r_1, \ldots, r_{i-1}] &\leq \sum_{j=1}^{d} \frac{w'(v_j)}{\sum_{i=1}^{d} w'(v_i)} \cdot \left(1 + \frac{1}{n}\right) \cdot \frac{W^{\widetilde{P}}(v_j)}{w'(v_j)} \\ &= \sum_{j=1}^{d} \frac{W^{\widetilde{P}}(v_j)}{\sum_{i=1}^{d} w'(v_i)} \cdot \left(1 + \frac{1}{n}\right) . \end{aligned} \tag{6.5}$$

From validation 2b it holds that

$$w'(u) = \sum_{i=1}^{d} w'(v_i) \ . \tag{6.6}$$

Hence, combining Eq. (6.6) and Claim 18 in Eq. (6.5) we get

$$\begin{aligned}\mathbb{E}[G_i|\, r_1,\ldots,r_{i-1}] &\le \frac{W^{\widetilde{P}}(u)}{w'(u)}\cdot\left(1+\frac{1}{n}\right)\\ &= g(u)\cdot\left(1+\frac{1}{n}\right) = g_{i-1}\cdot\left(1+\frac{1}{n}\right) .\end{aligned}$$

The claim follows. ■

Completing the proof. Denote by r the root of the protocol tree. Recall that $W^{\widetilde{P}}(u)$ is defined based on the protocol tree $T_{\widetilde{P_0},V_0}$ of a prover $\widetilde{P_0}$ and verifier V_0 for the original interactive proof of x. Hence for the root of the protocol tree r we know that $W^{\widetilde{P}}(r)$ is bounded above by the number of leaves with weight 1 in $T_{\widetilde{P_0},V_0}$, which is the number of sequences of coin tosses that lead the verifier to accept x. Thus if x is a no-instance, then $W^{\widetilde{P}}(r) \le \frac{1}{10}\cdot 2^{r(n)}$. Hence if the prover claims that some no-instance is a yes-instance, then at the beginning of the emulation $w'(r) \ge \frac{9}{10}\cdot 2^{r(n)}$ whereas $W^{\widetilde{P}}(r) \le \frac{1}{10}\cdot 2^{r(n)}$, thus $g_0 \le \frac{1}{9}$. Denote by v the leaf sampled at the end of the emulation. If the verifier accepts the complete emulation, then (in particular) the final checks pass and $W^{\widetilde{P}}(v) = w'(v) = 1$ and so $g_{m'} = g(v) = \frac{W^{\widetilde{P}}(v)}{w'(v)} = 1$.

Therefore, in order to upper bound the probability the verifier accepts, it suffices to upper bound the probability that the gap after the last round, $G_{m'}$, is greater than or equal to 1. Clearly,

$$\mathbf{Pr}[G_{m'} \ge 1] \le \mathbb{E}[G_{m'}] .$$

From Claim 22 we know that for every sequence of coin tosses $r_1,\ldots,r_{i-1}$ that determine g_{i-1}

$$\mathbb{E}_{r_i}[G_i|\, r_1,\ldots,r_{i-1}] \le g_{i-1}\cdot\left(1+\frac{1}{n}\right) .$$

Hence,

$$\mathbb{E}_{r_1,\ldots,r_i}[G_i] \le \left(1+\frac{1}{n}\right)\cdot\mathbb{E}_{r_1,\ldots,r_{i-1}}[G_{i-1}] . \tag{6.7}$$

Applying the bound from Eq. (6.7) iteratively it follows that

$$\mathbb{E}_{r_1,\ldots,r_{m'}}[G_{m'}] \le \left(1+\frac{1}{n}\right)^{m'}\cdot g_0 .$$

From property 8, the height of the emulation tree and hence the number of iterations of the new emulation is at most $m' = \lceil\frac{2r(n)}{\log n}\rceil + 1$. For $n \ge 8$ the value of m' is at most n and thus,

$$\begin{aligned}\mathbb{E}_{r_1,\ldots,r_{m'}}[G_{m'}] &\le \left(1+\frac{1}{n}\right)^{n}\cdot g_0\\ &\le e\cdot g_0 < \frac{1}{3} .\end{aligned}$$

where the last equality follows from the fact that $g_0 \le \frac{1}{9}$. Hence the verifier accepts with probability of at most $\frac{1}{3}$.

Appendix A Proof of transitivity - Claim 11

Denote the length of $\gamma(u)$ by i. Recall that if v is a transcript descendant of u then one of the following conditions hold:

1. $\gamma(v) = \gamma(u)$ and $R(v) \subseteq R(u)$, in this case we say that v is a transcript descendant of u of type (i).
2. $\gamma(u)$ is a proper prefix of $\gamma(v)$ and $\alpha^v_{i+1} \in R(u)$, in this case we say that v is a transcript descendant of u of type (ii).

We proceed by case analysis according to the descendancy types between u, v and z and show that in each case z is a transcript descendant of u.

- If z is a transcript descendant of v of type (i) and v is a transcript descendant of u of type (i) then $\gamma(u) = \gamma(v) = \gamma(z)$, and $R(z) \subseteq R(v) \subseteq R(u)$, so z is a transcript descendant of u of type (i).
- If z is a transcript descendant of v of type (i) and v is a transcript descendant of u of type (ii) then $\gamma(u)$ is a proper prefix of $\gamma(v) = \gamma(z)$. Since the transcripts of v and z are equal it follows that $\alpha^z_{i+1} = \alpha^v_{i+1}$. Because $\alpha^v_{i+1} \in R(u)$ we get that $\alpha^z_{i+1} \in R(u)$, so z is a transcript descendant of u of type (ii).
- If z is a transcript descendant of v of type (ii) and v is a transcript descendant of u of type (i) then $\gamma(v)$ is a proper prefix of $\gamma(z)$ and $\gamma(u) = \gamma(v)$, so $\gamma(u)$ is a proper prefix of $\gamma(z)$. Furthermore, $\alpha^z_{i+1} \in R(v) \subseteq R(u)$ so z is a transcript descendant of u of type (ii).
- If z is a transcript descendant of v of type (ii) and v is a transcript descendant of u of type (ii) then $\gamma(u)$ is a proper prefix of $\gamma(z)$. Furthermore, because the transcript of v is a prefix of the transcript of z then $\alpha^z_{i+1} = \alpha^v_{i+1}$. Since v is a transcript descendant of u of type (ii), we know that $\alpha^v_{i+1} \in R(u)$ and so $\alpha^z_{i+1} \in R(u)$. Hence, z is a transcript descendant of u of type (ii).

Appendix B Approximate Sampling

Let $D = (p_1, \ldots, p_d)$ be the probability distribution that assigns each $j \in [d]$ probability proportional to $w'(v_j)$; that is, $p_j = \frac{w'(v_j)}{\sum_{i=1}^d w'(v_i)}$. Our goal is to approximate the probability distribution D with a probability distribution $J = (p'_1, \ldots, p'_d)$ in the sense that for each $j \in [d]$ it holds that $p'_j \le p_j \cdot (1 + \frac{1}{n})$. Moreover, the probability distribution J should be one that the verifier can sample from using $k = O(\log n)$ coin tosses. Note that our method of approximation also satisfies $p'_j > p_j - 1/n^3$ for every $j \in [d]$, although the lower bound is not used in our work.

Let $k \in \mathbb{N}$ such that $2^{k-1} < n^3 \le 2^k$. Assume, without loss of generality, that p_d is the largest probability and thus $p_d \ge \frac{1}{d} \ge \frac{1}{n}$. For $j < d$ define p'_j by rounding down p_j to the closest fraction of 2^k, whereas we add the residual probability mass to p'_d. That is,

$$p'_j = \begin{cases} \frac{\lfloor p_j \cdot 2^k \rfloor}{2^k} & \text{for } j < d \\ 1 - \sum_{i=1}^{d-1} \frac{\lfloor p_i \cdot 2^k \rfloor}{2^k} & \text{for } j = d \end{cases}$$

Clearly for $j < d$ it holds that $p'_j \leq p_j$. For $j = d$,

$$\begin{aligned} p'_d &= 1 - \sum_{i=1}^{d-1} \frac{\lfloor p_i \cdot 2^k \rfloor}{2^k} \\ &\leq 1 - \sum_{i=1}^{d-1} \frac{p_i \cdot 2^k - 1}{2^k} \\ &= 1 - \sum_{i=1}^{d-1} p_i + (d-1) \cdot 2^{-k} \,. \end{aligned} \tag{B.1}$$

Recall that the number of children the prover supplies, d, is upper bounded by n, and that $2^{-k} \leq n^{-3}$. Thus,

$$(d-1) \cdot 2^{-k} \leq n \cdot n^{-3} = n^{-2} \,. \tag{B.2}$$

Because D is a probability distribution we get that $p_d = 1 - \sum_{i=1}^{d-1} p_i$ and so plugging Eq. (B.2) in Eq. (B.1) we get that $p'_d \leq p_d + \frac{1}{n^2}$.

Using the fact that $p_d \geq \frac{1}{n}$ it follows that $\frac{1}{n^2} \leq \frac{p_d}{n}$ and hence $p'_d \leq p_d \cdot (1 + \frac{1}{n})$ as required.

Appendix C Private-coin emulation

In the following appendix we prove a weaker version of the main theorem, which gives an upper-bound on the round complexity in terms of the randomness complexity, for *private-coin* interactive proof systems. The point in doing so is that the proof is significantly simpler

Theorem 23 *Suppose that S has an interactive proof system of randomness complexity $r(n)$ for instances of length n. Then, S has a private-coin interactive proof system of round complexity $O(r(n)/\log n)$ and randomness complexity $r(n)$. Furthermore, the soundness and completeness of the original interactive proof system are preserved.*

This appendix can also be read independently from the proof of the main theorem. The general structure of the proof is similar to the one of the main theorem. In Section C.1 we describe the protocol tree of the original proof system. The protocol tree is used to construct an emulation tree in Section C.2. In Section C.3 we describe the private-coin emulation, which uses the emulation tree constructed in the previous section. The analysis of the private-coin emulation is performed in Section C.4.

We provide notes that point out the main differences and similarities between the private and public-coin emulation protocols. These notes are typeset as this one.

C.1 The protocol tree of the original proof system

Note that the protocol tree for the public-coin emulation described in Section 3 is similar to the one described here, except for the definition of weights.

Fixing an interactive proof and an instance x of length n, we describe the possible prover-verifier interactions of the system on common input x using a tree whose height corresponds to the number of rounds of interaction. For some $\ell = \ell(n)$, we assume without loss of generality that in each round the verifier sends a message $\alpha \in \{0,1\}^{\ell}$, and the prover's responds with a message $\beta \in \{0,1\}^{\ell}$. We can also assume, without loss of generality, that the prover's strategy is deterministic and fixed. Each node v in level j represents a possible prover-verifier transcript for the first j rounds of the interaction. The branching of the tree represents the possible ways to extend the transcript to the next round. The number of ways to extend the transcript depends only on the verifier's message, since we fixed the prover's strategy. Hence, each node has *at most* $d := 2^{\ell}$ children, corresponding to the 2^{ℓ} possible verifier messages for the next round. The prover's response to each such message is included in the **description** of the corresponding node.

The description of a node u on level j contains the partial **transcript** $\gamma(u) = \alpha_1\beta_1, \ldots, \alpha_j\beta_j$ of the interaction up to the j'th round. The root (at level zero) has an empty transcript, whereas a leaf of the tree represents a complete prover-verifier interaction. We can assume, without loss of generality, that the verifier sends its private coins on the last round, and hence every leaf is associated with a sequence of coin tosses which either leads the verifier to accept or to reject. Hence, we can represent the possible interactions generated by the interactive proof system using a tree of height m that has $2^{r(n)}$ leaves, where m is the number of rounds and $r(n)$ is the number of coin tosses.

The description of a node also contains its **weight**, denoted $w(u)$. The weight of the node is the number of coin sequences that are consistent with the node and lead the verifier to accept at the end of the interaction. That is,

Definition 24 ***(Weight of a leaf)*** *The weight of a leaf is defined to be* 1. *Recall that a leaf u corresponds to a full transcript of the interaction of P and V on input x, when V uses a sequence of coin tosses ρ.*

Definition 25 ***(Weight of a node)*** *The weight of a node u in the protocol tree is the sum of the weights of the leaves that are descendants of u.*

Note that the weight of node in the protocol tree, which corresponds to a possibly partial transcript, is proportional to the probability that a sequence of coin tosses is consistent with that transcript.

In the public-coin emulation, the weight of a leaf is defined as 1 if the verifier accepts at the end of the interaction, otherwise the weight is 0. Hence, in the public-coin emulation the weight of the node is the number of coin sequences that are consistent with the corresponding transcript *and* lead the verifier to accept at the end of the interaction.

C.2 The emulation tree

Using the protocol tree, we create a new tree of height $O(r(n)/\log n)$ to guide the prover's strategy. We call this tree the **emulation tree**. The nodes in the emulation tree are nodes from the protocol tree, however the children of a node u in the emulation tree may be non-immediate descendants of u in the protocol tree.

The procedure for constructing the private-coin emulation tree is similar to the one described for the the main public-coin emulation, provided that the degree of the protocol tree is $poly(n)$. In the foregoing private-coin emulation we only care about the height of the emulation tree, and we do not mind if the degree of the tree is not polynomially bounded. Hence, unlike in the public-coin emulation, we do not group the nodes under interval children in order to reduce the degree.

We start with a protocol tree T rooted at u whose weight is $w(u) = 2^{r(n)}$, and on each step we modify this tree towards creating an emulation tree. We define a **heavy descendant** of u to be a node in T whose weight is at least $\frac{w(u)}{n}$, such that this descendant is either a leaf, or the weight of each of its children is smaller than $\frac{w(u)}{n}$. Note that there are at most n such nodes.

We modify T so that the children of u in the emulation tree are its original children as well as the heavy descendants that we lift upwards to make them new children of u. This modification is performed using the Build_Tree(u) procedure, which when invoked on a node u identifies the nodes that will be children of u in the new tree, sets them as children of u, and then initiates recursive invocations on the (original and new) children of u, creating the new emulation tree rooted at u. Details follow.

Let T_u denote the temporary tree after the stage that we identify the heavy descendants of u and raise them to be children of u. We start by identifying a heavy descendant v of u (v can also be a child of u in T), which will become a **heavy child** of u in T_u. We move v to be a direct child of u, along with the subtree rooted at v (see Figure 14). We update the weights of the ancestors of v that are descendants of u by subtracting $w(v)$ off their weight. We then proceed to the next heavy descendant of u. When we finish identifying all the heavy children, the children of u in T_u consist of the heavy children of u along with the original children of u in T. Next, we erase any nodes whose weights are 0 from the tree. The weight of a descendant of u may become zero, for example, if all its children were identified as heavy descendants of u.

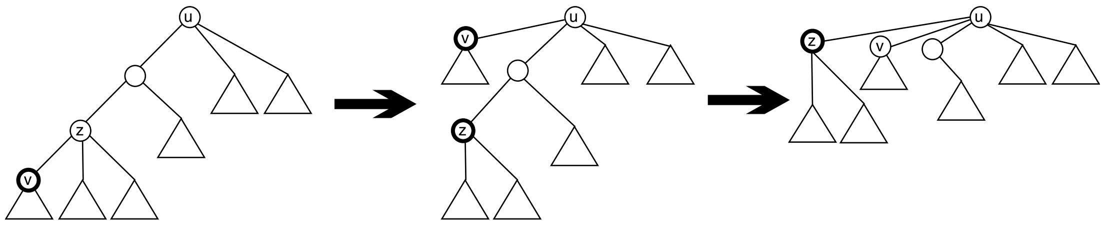

Figure 14: In the first step, v is identified as a heavy descendant of u and moved to be a heavy child of u. In the second step, z is identified and moved to be a heavy child of u. The triangles represent subtrees of the original tree.

Lastly, we invoke the Build_Tree procedure on each child of u in T_u, which creates an emulation tree rooted at that node.

Observe that in the final emulation tree, for each node u, the weight of each grandchild of u is at most $\frac{w(u)}{n}$. If v is a heavy child of u in the emulation tree, then the weight of the descendants of v in T_u is at most $\frac{w(u)}{n}$. Since the children of v in the emulation tree are descendants of v in T_u the claim holds. Otherwise, v is non-heavy child of u, so its weight is smaller than $\frac{w(u)}{n}$ and hence the weight of the children of v is also smaller than $\frac{w(u)}{n}$.

It follows that the weight of a node in level $2r(n)/\log n$ is at most $\frac{2^{r(n)}}{n^{r(n)/\log n}} = 1$. Recall that we perform a clean up stage to delete nodes with weight equal to zero, and hence we guarantee that the weights of all the nodes in the emulation tree are positive integers. It follows that the height of the final emulation tree is $O(r(n)/\log n)$. We note that the number of heavy children of each node is at most n, whereas the number of non-heavy children is unbounded.

C.3 Emulation protocol

Next, we describe the strategy of the designated prover P and verifier V in the new ("emulation") protocol. Denote the designated prover and verifier of the original proof system by P_0 and V_0 respectively, and by E_{P_0,V_0} the emulation tree constructed in the previous subsection. The verifier V does not have access to the emulation tree, but it has access to the original verifier's strategy V_0. The emulation starts with the verifier sampling private coins $\rho \in \{0,1\}^{r(n)}$. Starting from the root of the emulation tree, in each iteration, the prover and the verifier progress one step down the emulation tree, until reaching a leaf that represents the complete transcript of the original interaction.

The main difference between the public-coin emulation and the private-coin one is in the way a child of a node is chosen in each iteration. In the public coin emulation V does not have private coins, hence it must choose a continuation based on the transcripts and the probability distributions suggested by P. In contrast, in the foregoing emulation the values of the verifier's private coin tosses determine which child is chosen.

In each iteration, the prover provides the transcripts of the *heavy children* $v_1, \ldots, v_d$ of the current node u, which was reached in the previous iteration. The verifier preforms validations on the list supplied by the prover (to be detailed below), and aborts if any of these validations fail. If one of the transcripts provided by the prover is consistent with the verifier's private coins, then the verifier chooses this transcript and the next iteration proceeds from this heavy child. Otherwise, the verifier sends its next message, according to the strategy of V_0 and to the values of its private coins ρ. The prover then answers with its response to the verifier's message. In this case, the continuation of the transcript corresponds to one of the non-heavy children of u in the emulation tree. Towards the next iteration, the prover and verifier proceed from the new node. On the last iteration the verifier checks that the full transcript, along with the sequence of coin tosses, leads the original verifier V_0 to accept.

The validations that the verifier performs are meant to ensure that the transcripts that the prover provides for the new emulation are consistent with a deterministic prover strategy for the original interactive proof system. In such a case, we can claim that, since the original prover cannot fool the verifier with high probability, the new prover cannot do so either.

Definition 26 *(Prover consistent) We say that two transcripts $\gamma(u)$ and $\gamma(v)$ are **prover consistent** if the maximal prefix they agree on is either empty or ends with a prover's message. That is, the prover should respond in the same way on the same prefix of the transcripts.*

The verifier is able to check prover consistency only between previous transcripts seen so far in the emulation. For this reason the verifier keeps a list S of the nodes that were seen during the emulation, and at each iteration, it checks prover consistency between the new transcripts and the transcripts of the nodes in S.

Note that the nodes in S, which the verifier has seen up to some iteration, are a subtree of the emulation tree that consists of a path from the root of the tree to the current input node, augmented with the children of the nodes on the path. See Figure 15.

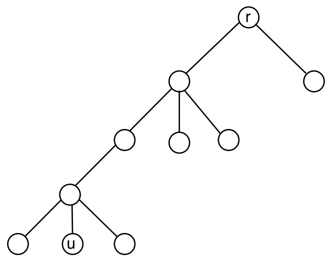

Figure 15: The nodes in the list of seen nodes, S, in the iteration that the input node is u.

Recall that we want to claim that all the transcripts in the emulation tree $E_{\widetilde{P}}$ of some untrusted prover are prover consistent, whereas the verifier can only check the prover consistency between the transcripts of the seen nodes, which consist of a very partial portion of the emulation tree. In order to guarantee that consistency between the nodes in S is enough to ensure that all the transcripts in the emulation tree are prover consistent, we add additional validations that check the structure of the emulation tree. The goal of these checks is to make sure that the emulation tree was constructed using a protocol tree where the only changes done to the protocol tree are "raising" parts of the tree (has done with the heavy children in the designated construction) [6].

Initially, for the first iteration, the verifier samples its private coins $\rho \in \{0,1\}^{r(n)}$. The input node is the root. The verifier sets the set S of seen nodes to contain the transcript of the root, which is the empty string. The rest of the first iteration, as well as subsequent iterations, proceed as follows.

Construction 27 *(the ith iteration) On input a node u and list of seen nodes S.*

1. The prover provides the transcripts of the heavy children $v_1, \ldots, v_d$ of u: $\gamma(v_1), \ldots, \gamma(v_d)$.
2. The verifier preforms the following validations and rejects if any of them fails:
   (a) For each $j \in [d]$ the verifier checks that $\gamma(u)$ is a proper prefix of $\gamma(v_j)$.
   (b) For each $j \in [d]$ the verifier checks that the transcript $\gamma(v_j)$ is prover consistent with the other transcripts in S and with the other transcripts $\gamma(v_k)$.
   (c) For each $j \in [d]$ the verifier checks that $\gamma(v_j)$ is not part of the emulation tree that was truncated from $\gamma(u)$; that is, if $\gamma(u)$ is a proper prefix of a transcript $\widetilde{\gamma} \in S$, then $\widetilde{\gamma}$ should not be a prefix of $\gamma(v_j)$. (Note that this also implies that $\gamma(v_j)$ is different from the other transcripts in S.)
   (d) The verifier checks that all the nodes are different (according to their transcripts), that is, for each distinct $i, j \in [d]$ the verifier checks that $\gamma(v_i) \neq \gamma(v_j)$.

[6]We note that these validations (i.e. the validations other than the prover consistency check) are performed in order to aid the analysis. An alternative way to perform the analysis is to claim that all the transcripts that are feasible, in the sense that the verifier might choose them in some invocation of the emulation, are consistent with some deterministic prover strategy. (This means that the transcripts that fail the structural validations are ones that the verifier never chooses anyways, so we can ignore them when defining a deterministic strategy for the original prover.)

3. The verifier checks if any of the transcripts the prover provided are consistent with the values of its private coins ρ, where a transcript $\gamma_j = \alpha_1\beta_1, \ldots, \alpha_k\beta_k$ is consistent with a sequence of private coins ρ, if for every $i \in [k]$ it holds that $V_0(x, \rho, \beta_1, \ldots, \beta_{i-1}) = \alpha_i$. There are two options according to whether or not there exists a suggested transcript that is consistent with ρ.

   (a) If there exists a transcript that is consistent with ρ, the verifier sends the prover the maximal transcript $\gamma(v_j)$ that is consistent with its coins. The prover and the verifier update the input node u for the next iteration to be v_j. The verifier updates the set S of seen transcripts $S \leftarrow S \cup \{\gamma(v_1), \ldots, \gamma(v_d)\}$.

   (b) Otherwise, the verifier sends its next message α according to the value of its private coins ρ. That is, if the current transcript is $\gamma(u) = \alpha_1\beta_1, \ldots, \alpha_k\beta_k$, then the verifier sends α such that $V_0(x, \rho, \beta_1, \ldots, \beta_k) = \alpha$. The prover answers with a message β such that $\gamma(u)\alpha\beta$ is a transcript of a child of u in the emulation tree. (The assumption that there exists such a child in the emulation tree is justified in the completeness part of the analysis.) The verifier updates the set S of seen transcripts; that is, $S \leftarrow S \cup \{\gamma(v_1), \ldots, \gamma(v_d), \gamma(u)\alpha\beta\}$. Towards the next iteration the prover and the verifier update the input node for the next iteration to be the node in the emulation tree whose transcript is $\gamma(u)\alpha\beta$.

Unless $\gamma(u)$ is the complete transcript (which contains the last message), the next iteration will start with transcript $\gamma(u)$ and the set S. Otherwise, we proceed to the final checks.

Final check. After the complete transcript $\gamma = \alpha_1\beta_1, \ldots, \alpha_m\beta_m$ has been determined, the verifier V accepts if and only if ρ is accepting for γ; that is, if $V_0(x, \rho, \beta_1, \ldots, \beta_m) = 1$.

Number of rounds. In the proposed emulation each iteration consists of a prover message in Step 1, followed by a verifier message in Step 3, and then possibly another prover message in Step 3b. Hence each iteration takes two rounds of communication. However, we can augment the last prover message in Step 3b with the prover message in Step 1 of the next iteration. Thus, the number of rounds of communication is the number of iterations of the emulation protocol plus one.

C.4 Proof of correctness

C.4.1 Completeness

We claim that if x is a yes-instance, then the new verifier V accepts with probability greater than $c(n)$, the completeness parameter of the original interactive proof system. The proof of completeness is partitioned into three parts. First, we shall show that the prover's strategy P, as specified in subsection C.3, is indeed well defined; specifically, we shall show that if the verifier does not choose one of the continuations suggested by the prover, but rather sends its next message according to the value of its coin tosses, then there is an unique node in the emulation tree that corresponds to the new transcript. Next, we show that the strategy of P is such that V does not abort until the final check. Finally, we show that this implies that the probability that V accepts at the end of the interaction with P is equal to the probability that V_0 accepts at the end of the interaction with P_0, which is at least $c(n)$.

The prover's strategy is well defined. We begin by showing that the the strategy of P in Step 3b is well defined. That is, we have to show that if the verifier does not choose one of the suggested continuations of u provided by the verifier, but rather sends its next message α in Step 3b according to the value of its coin tosses ρ, then u has an unique child in the emulation tree E_{P_0,V_0} whose transcript is $\gamma(u)\alpha\beta$ for some $\beta \in \{0,1\}^\ell$.

Let β be the message P_0 sends in response to the transcript $\gamma(u)\alpha$. All the nodes in E_{P_0,V_0} appear in T_{P_0,V_0}, and thus all the transcripts must be consistent with the strategy of P_0. Thus, every transcript in E_{P_0,V_0} whose prefix is $\gamma(u)\alpha$ must proceed with β. Note that u cannot have two children whose transcripts are $\gamma(u)\alpha\beta$ since all the transcripts in T_{P_0,V_0} are distinct, and the same must hold in E_{P_0,V_0}. The reason that u has a child whose transcript is $\gamma(u)\alpha\beta$ is as follows. If continuations of $\gamma(u)$ that are consistent with the values of V's private coin tosses were not suggested by the prover P in this iteration or a previous one, then this implies that $\gamma(u)\alpha\beta$ was not raised to be a heavy child of an ancestor of u, and hence it is a child of u in E_{P_0,V_0}. Similarly, the leaf whose transcript is the complete interaction with private coins ρ was not not raised to be a heavy child of u or of its ancestors, and hence it is a descendant of $\gamma(u)\alpha\beta$ in E_{P_0,V_0}. Thus, the weight of the node whose transcript is $\gamma(u)\alpha\beta$ is non-zero, so it was not erased from the tree, and it is a child of u.

The validations in Step 2 are satisfied. We shall show that the validations in Step 2 are satisfied in every iteration. This is equivalent to showing that validations are satisfied for every non-leaf node u in the final emulation tree E_{P_0,V_0}, in the iteration that u was the input node (i.e. the node handled on that iteration).

Showing that the validations in Step 2 are satisfied is a simplified version of the completeness proof of the public-coin protocol, which is given in Subsection 6.1.

The general outline of the proof consists of going over every validation performed and showing that the property being checked holds for every node in the original protocol tree T_{P_0,V_0}, and continues to hold with every modification of the global tree as part of the Build_Tree procedure. Thus, the property also holds for every node in the final emulation tree E_{P_0,V_0}, and hence the validations are satisfied.

When we say that a validation passes relative to a (possibly intermediate) tree T and node u in T, we mean that if the tree T had been used as an emulation tree, then in the iteration on which u is the input node, the validation would have passed. The children of u that are considered in the validation are the children of u in T, and the list of the seen nodes S consists of the ancestors of u and their children in T.

Let T be the global tree at some point in the construction, and let z be a node in T that the Build_Tree procedure is currently invoked on. We assume that the validation we are currently checking holds for every node in T, and show that it also holds after the next modification of the tree, which is identifying a heavy descendant for z as part of the Build_Heavy procedure and moving it to be a child of z. Denote the child identified by v, and the global tree after this modification by T_v.

Note that it is not sufficient to show that the property being checked holds for v in T_v. This is because the procedure might affect the descendants and ancestors of v in T_v, as well as nodes whose list of seen nodes changes. Exactly which nodes are effected depends on the validation.

Remark 28 *Let v be a node in T that the Build_Tree procedure has not been invoked on yet. Recall that the children of v in T are children of the node v in T_{P_0,V_0}. Hence, like in* *the tree T_{P_0,V_0}, the transcripts of the children of v in T extend the transcript of v by one pair of messages. Furthermore, if we did not invoke Build_ Tree on v yet, then we also did not invoke it on the descendants of v in T. Thus, the subtree of T rooted at v, denoted by $T(v)$, is a subtree of T_{P_0,V_0}.*

Now, we go over the validations in Step 2, which are stated for a node u and its children $v_1, \ldots, v_d$ provided by the prover as part of the emulation. The validations are numbered as in Construction 27. We shall prove that these validations pass for every node in E_{P_0,V_0}.

(a) In this validation, for each $j \in [d]$ the verifier checks that $\gamma(u)$ is a proper prefix of $\gamma(v_j)$. In the original protocol tree T_{P_0,V_0} the transcript of each node extends the transcript of its parent by a pair of messages and thus the property holds. Assume that every node in some temporary tree T maintains the property that its parent's transcript is a proper prefix of its transcript. Let v be a heavy descendant identified for z and moved to be a child of z along with the subtree under it. The only node in T_v that has a child it did not have in T is z, which now has v as a child. Since v is a descendant of z in T, and the transcript of every node in T is a proper prefix of the transcripts of its children, then the transcript of z is a proper prefix of the transcript of v. Hence, in the new tree T_v, the transcript of each node is a proper prefix of the transcript of its children as well.

(b) In this validation, the verifier checks, for each child v_j of u, that $\gamma(v_j)$ is prover consistent with respect to the other transcripts of nodes in S and with respect to the transcripts of the other children of u.
In the original protocol tree, T_{P_0,V_0}, every two nodes are prover consistent since P_0 is deterministic. (If there were two partial transcript in T_{P_0,V_0} whose maximal common prefix ends with a verifier message it would mean that that the prover can respond in different ways to the same partial transcript). The transcripts of the nodes in E_{P_0,V_0} all appear in T_{P_0,V_0}, so every two transcripts in E_{P_0,V_0} are prover consistent as well.

(c) In this validation the verifier checks, for each child v_j of u, that v_j is not part of the emulation tree that was truncated from $\gamma(u)$; that is, if $\gamma(u)$ is a prefix of a transcript $\widetilde{\gamma} \in S$, then $\widetilde{\gamma}$ should not be a prefix of $\gamma(v_j)$. (Note that this also implies that for each transcript $\widetilde{\gamma} \in S$ that $\widetilde{\gamma} \neq \gamma(v_j)$.)

The main idea is that the changes we make to the emulation tree in every step of the construction are identifying a heavy descendant and moving it to be a direct child along with the subtree under it. Hence, if some node v whose transcript is $\widetilde{\gamma} = \gamma(v)$ and is a descendant of u, is identified as a heavy child of an ancestor of u, denoted by z, then v is moved, along with the subtree rooted at v, to be a descendant of z. Hence, all the nodes that $\gamma(v)$ is a prefix of *and* are descendants of v cannot be descendants of u, and in particular they cannot be children of u after the move. However, it is not clear that after moving v to be a heavy child of z the only potential nodes that we need to check that the claim holds for are the ancestors of v (note that above we assumed that u is an ancestor of v). Moreover, it is not true that all the nodes that $\gamma(v)$ is a prefix of are descendants of v in the emulation tree, since some of these nodes might have been lifted to be heavy children of ancestors of v in a previous iteration. Hence, a detailed proof follows.

When we consider a temporary tree T, which may not be the final emulation tree, and some node $v \in T$, then we denote by $S(v)$ the set of seen nodes S in the beginning of the iteration where v is the input node handled. That is, $S(v)$ is the set containing the nodes that are ancestors of v in T, augmented with their children.

Let $u \in E_{P_0,V_0}$ and $\widetilde{\gamma} \in S(u)$ such that $\gamma(u)$ is a prefix of $\widetilde{\gamma}$ we prove that for each *descendant* b of u in E_{P_0,V_0} the transcript $\widetilde{\gamma}$ is not a prefix of $\gamma(b)$.

Note this is a stronger claim than what we need to show, because we only need to show it for the children of u in E_{P_0,V_0} and not for each descendant of u.

First, we shall show that the claim holds in the initial protocol tree T_{P_0,V_0}. The transcript of each node is T_{P_0,V_0} is a prefix only of its descendants in the tree. However, u is not an ancestor of any node in $S(u)$, so $\gamma(u)$ cannot be a prefix of any $\widetilde{\gamma} \in S(u)$. Next, we shall assume that the claim holds in the global tree T before creating a child v of z, and we show that it holds in the tree T_v after the identification v as a heavy child of u.

When v is identified as a heavy descendant of z, node v is moved to be a child of z along with the subtree under it. In order to show that the claim holds in T_v, it is enough to consider the nodes $u \in T_v$ that have new descendants or new nodes in $S(u)$ relative to the ones they had in T. The descendants of every node in T_v are all descendants of it in T.

The only nodes in T_v that have new nodes in their seen list are the descendants of z, since now $\gamma(v)$ is in their seen list, whereas $\gamma(v)$ may not have been in their seen list before the move. By Remark 28, before the invocation of Build_Tree(z) the subtree rooted at z is a subtree of T_{P_0,V_0}. Let u be a descendant of z in T_v. Since determining the heavy descendants of z is done bottom up, if $\gamma(u)$ is a prefix of $\gamma(v)$ then u is an ancestor of v in T. It is left to check that $\gamma(v)$ is not a prefix of the transcripts of the descendants of u in T_v. By Remark 28, it follows that the subtree of T_v rooted at u, denoted by $T_v(u)$, is a subtree of T_{P_0,V_0}. Thus, because $v \notin T_v(u)$ (recall that v was lifted to be a heavy child of z, and u is a descendant of z) it follows that $\gamma(v)$ is not a prefix of the transcripts of the nodes in $T_v(u)$, which are the descendants of u in T_v. (See Figure 16.)

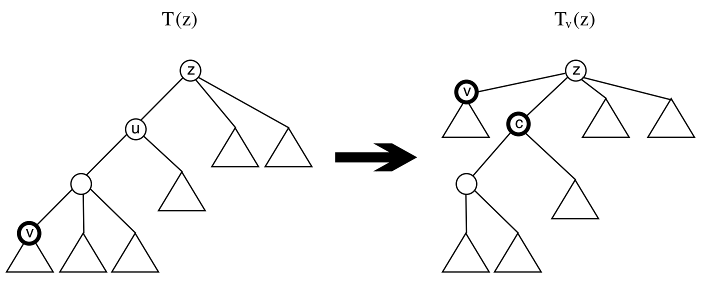

Figure 16: v is identified as a heavy child of z, and u is a descendant of z, which was an ancestor of v in T.

(d) In this check the verifier checks that all the children of u have different transcripts; that is, for every two distinct children v_i and v_j of u, the verifier checks that $\gamma(v_i) \neq \gamma(v_j)$. In the original protocol tree T_{P_0,V_0}every two nodes have different transcripts. The nodes in E_{P_0,V_0} all appear in T_{P_0,V_0}, so every two nodes in E_{P_0,V_0} also have different transcripts, and in particular every two children of u have different transcripts.

Concluding the proof of completeness. In order to conclude the proof of completeness we use the following claim, which shows that the probability that V accepts at the end of the interaction with P is equal to the probability that V_0 accepts at the end of the interaction with P_0. The claim is stated in a more general way than needed for the proof of completeness, so that we shall also be able to use it in the soundness part of the analysis.

Claim 29 ***(Relating leaves in the two trees)*** *Let $\widetilde{P}$ be a prover strategy for the emulation of input x, with emulation tree $E_{\widetilde{P}}$ such that the verifier V does not abort until the final checks. Let $\widetilde{P_0}$ be a prover strategy for the original interactive proof system that is consistent with all the transcripts in $E_{\widetilde{P}}$; that is, for every $u \in E_{\widetilde{P}}$ with transcript $\gamma(u) = \alpha_1\beta_1, \dots, \alpha_j\beta_j$, the strategy of $\widetilde{P_0}$ satisfies $\widetilde{P_0}(x, \alpha_1, \dots, \alpha_i) = \beta_i$ for every $i \leq j$. Then, the transcript γ created by the end of the emulation of $\widetilde{P}$ and V when using coins ρ is equal to the transcript interacted between $\widetilde{P_0}$ and V_0 when using coins ρ. Thus, the probability that V accepts at the end of the interaction with $\widetilde{P}$ is equal to the probability that V_0 accepts at the end of the interaction with $\widetilde{P_0}$.*

The proof of completeness follows by using in Claim 29, P as $\widetilde{P}$ and P_0 as $\widetilde{P_0}$. Indeed, we can do so because as showed previously it holds that V does not abort until the final checks when interacting with P. In addition, since the emulation tree was constructed using the protocol tree T_{P_0,V_0}, for every node u in the emulation tree with transcript $\gamma(u) = \alpha_1\beta_1, \dots, \alpha_j\beta_j$ it holds that $P_0(x, \alpha_1, \dots, \alpha_i) = \beta_i$ for every $i \leq j$. Applying Claim 29, the probability that V accepts at the end of the interaction with P is equal to the probability that V_0 accepts at the end of the interaction with P_0. Thus, the completeness of the original interactive proof system is maintained.

Proof. Let ρ be the value of the coins of V. By the assumption, all the transcripts in the emulation tree $E_{\widetilde{P}}$ are consistent with the strategy of $\widetilde{P_0}$. Recall that, in each iteration of the emulation of V and $\widetilde{P}$, a new node in $E_{\widetilde{P}}$ is chosen, and the continuation of the transcript is according to the transcript of the new node. Hence, in each iteration the continuation of the transcript must be consistent with the transcript of $\widetilde{P_0}$. The proof follows by noting that the continuations of the transcript are also consistent with the strategy of V_0 with coins ρ. That is, if the verifier V chooses a continuation of the transcript suggested by P, then this transcript must be consistent with the strategy of V_0 with coins ρ. Otherwise, the verifier sends its next message based on the strategy of V_0 with coin tosses ρ. It follows that in every iteration the current transcript is consistent with the strategy of $\widetilde{P_0}$ and V_0 with coins ρ.

From the assumption that V does not abort until the final checks, we know that after the last iteration the complete transcript had been interacted. Hence, this complete transcript is equal to the transcript of the interaction between $\widetilde{P_0}$ and V_0 with coins ρ.

The final check of the emulation of $\widetilde{P}$ and V with random coin ρ pass if and only if V_0 with coins ρ accepts the complete transcript that has been interacted. Recall that the private coins in the original and new emulation are sampled using the same probability distribution. Thus, the probability that V accepts at the end of the interaction with $\widetilde{P}$ is equal to the probability that V_0 accepts at the end of the interaction with $\widetilde{P_0}$. ■

C.4.2 Soundness

Let x be a no-instance. We shall show that V rejects x with probability at least $s(n)$, the original soundness parameter.

The main part of the soundness proof here is Lemma 30, which is a slightly simplified version of Lemma 16 in Section 6.2.1. The other two components of the soundness analysis of the public-coin system (which are defining the notion of real weight of a node, and the actual proof of soundness) are not required for the private-coin soundness analysis.

The crux of the proof is extracting a deterministic strategy $\widetilde{P_0}$ for the original prover using the strategy of $\widetilde{P}$. We can assume, without loss of generality, that $\widetilde{P}$ is deterministic since for every probabilistic prover there is a deterministic prover for which the verifier's rejection probability is at least as high. Because the prover is deterministic we know that there is an emulation tree underlying the prover's strategy $\widetilde{P}$ (this emulation tree is simply the protocol tree of the new interactive proof system). We denote this emulation tree by $E_{\widetilde{P}}$. We define a strategy for $\widetilde{P_0}$ by using the transcripts in $E_{\widetilde{P}}$. That is, for each $u \in E_{\widetilde{P}}$ with transcript $\gamma(u) = \alpha_1\beta_1 \ldots, \alpha_j\beta_j$, we define $\widetilde{P_0}(x, \alpha_1, \ldots, \alpha_i) := \beta_i$ for all $i \le j$. We extend $\widetilde{P_0}$'s strategy to transcripts that do not appear in $E_{\widetilde{P}}$ in an arbitrary way. The main part of the analysis consists of showing that the transcripts of every two nodes in $E_{\widetilde{P}}$ are prover consistent, i.e. that their maximal common prefix ends with a prover message, and thus the strategy of $\widetilde{P_0}$ is well defined.

We can assume, without loss of generality, that the strategy of $\widetilde{P}$ is such that the verifier does not abort until the final checks. This is because every prover strategy in which the verifier aborts in one of the intermediate checks can be modified to a prover strategy in which the verifier does not abort until the final check and the verifier's acceptance probability is at least as large. In Lemma 30 we show that this implies that all the transcripts in $E_{\widetilde{P}}$ are prover consistent, and thus the strategy of $\widetilde{P_0}$ is well defined. The proof of soundness then follows by applying Claim 29 (provided in subsection C.4.1), that implies that the probability that V accepts when interacting with $\widetilde{P}$ is equal to the probability that V_0 accepts when interacting with $\widetilde{P_0}$, which is at most the soundness parameter $s(n)$.

It is left to show that the strategy of $\widetilde{P_0}$ is well defined, i.e. that no two nodes $u, v \in E_{\widetilde{P}}$ share the prefix of prover-verifier interaction but differ on the prover's response. That is, we show that every two nodes $u, v \in E_{\widetilde{P}}$ are **prover consistent** (see Definition 26).

For a node $u \in E_{\widetilde{P}}$ provided during the emulation, denote by $S(u)$ the list of seen nodes from $E_{\widetilde{P}}$ at the beginning of the iteration in which u was the input node (i.e. was the node handled on that iteration). That is, the nodes in $S(u)$ are the ancestors of u in $E_{\widetilde{P}}$ and their children. Validation 2b implies that each node u in the emulation tree is prover consistent with the transcripts of the nodes in $S(u)$. We show that using validations 2a,2c and 2d it follows that *every* two transcripts in the emulation tree are prover consistent.

Lemma 30 *(Prover consistency of the emulation tree). If $\widetilde{P}$ is a prover strategy for the new emulation such that the verifier V does not abort until the final checks, then every two transcripts of nodes in the emulation tree $E_{\widetilde{P}}$ are prover consistent.*

Note that the following proof is a slightly simplified version of the proof of Lemma 16 in Section 6.

Proof. Let u and v two nodes in the emulation tree $E_{\widetilde{P}}$, we wish to show that their transcripts $\gamma(u)$ and $\gamma(v)$ are prover-consistent. If one of the nodes is in the seen node list S of the other node (that is if $u \in S(v)$ or $v \in S(u)$) then the transcripts of u and v must be prover consistent by validation 2b. Otherwise, the intersection between the list of seen nodes of u and v is non-empty, and particular it contains the least common ancestor of u and v, denoted by z, as well as the children of z that are ancestors of u and v, denoted by a and b respectively, see Figure 17 for illustration. (Note that z is not equal to u or to v, otherwise $u \in S(v)$ or $v \in S(u)$. Similarly $a \neq u$ and $b \neq v$.) Hence, by using the prover consistency between the transcripts of a, b, z and the transcript of u, as well as between the transcript of v, and by using structural validations between the nodes in the emulation tree (validations 2a, 2c and 2d) we are able to show that the transcripts of u and v are also prover consistent. Details follow.

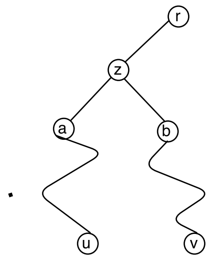

Figure 17: The subtree of E_{P_0,V_0} that contains u and v

Since a is an ancestor of u and b is an ancestor of v, then $\gamma(a)$ is a proper prefix of $\gamma(u)$, and $\gamma(b)$ is a proper prefix of $\gamma(v)$. We consider two cases according to the relation between $\gamma(a)$ and $\gamma(b)$.

First, consider the case that $\gamma(a)$ is not a prefix of $\gamma(b)$ and $\gamma(b)$ is not a prefix of $\gamma(a)$. By validation 2a we know that each node in this path from u to a is a prefix of its parent. Thus, $\gamma(a)$ is a prefix of $\gamma(u)$. Similarly, $\gamma(b)$ is a prefix of $\gamma(v)$. Recall that we are in the case that $\gamma(a)$ is not a prefix of $\gamma(b)$ and $\gamma(b)$ is not a prefix of $\gamma(a)$, and so the maximal prefix on which $\gamma(a)$ and $\gamma(b)$ agree upon is a proper prefix of both. This common prefix equals the maximal prefix on which $\gamma(u)$ and $\gamma(v)$ agree. We know that $\gamma(a)$ and $\gamma(b)$ are prover-consistent because the prover provides a along with b as children of z and we assume that validation 2b is satisfied. Since the maximal prefix that $\gamma(u)$ and $\gamma(v)$ agree on is equal to the maximal prefix that $\gamma(a)$ and $\gamma(b)$ agree on, it follows that $\gamma(v)$ and $\gamma(u)$ are also prover-consistent. See Figure 18 for illustration.

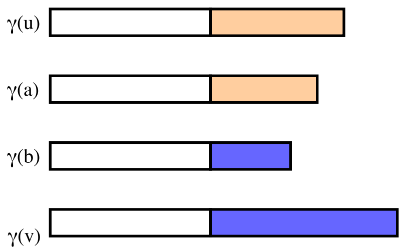

Figure 18: $\gamma(a)$ and $\gamma(b)$ agree on the prefix in white, while $\gamma(a)$ is a prefix of $\gamma(u)$ and $\gamma(b)$ is a prefix of $\gamma(v)$.

Note that $\gamma(a)$ cannot be equal to $\gamma(b)$ since that would be a violation of validation 2d. We are left with the case that one of the transcripts $\gamma(a)$ and $\gamma(b)$ is a proper prefix of the other, and assume, without loss of generality, that $\gamma(a)$ is a proper prefix of $\gamma(b)$. Denote the transcript of b by $\gamma(b) = \alpha_1\beta_1, \ldots, \alpha_k\beta_k$.

We claim that $\gamma(b)$ is not a prefix of $\gamma(u)$. Assume in contradiction that $\gamma(b)$ is a prefix of $\gamma(u)$. Note that $b \in S(u)$, so by validation 2c, $\gamma(b) \neq \gamma(u)$, so $\gamma(b)$ must be a proper prefix of $\gamma(u)$. Denote the nodes in the path from a to u in $E_{\widetilde{P}}$ by $a = a_0, a_1, \ldots, a_k = u$ and by a_j the first node in the path such that $\gamma(a_j)$ is a prefix of $\gamma(b)$ and $\gamma(a_{j+1})$ is not a prefix of $\gamma(b)$. There must exist such node a_j since $\gamma(a)$ is a prefix of $\gamma(b)$, and $\gamma(u)$ is not a prefix of $\gamma(b)$. Note that since a_{j+1} is an ancestor of u in $E_{\widetilde{P}}$, then by validation 2a it follows that $\gamma(a_{j+1})$ is a prefix of $\gamma(u)$. Hence, $\gamma(b)$ and $\gamma(a_{j+1})$ are both prefixes of $\gamma(u)$. Thus, one of them must be a prefix of the other, and so in this case $\gamma(b)$ is a prefix of $\gamma(a_{j+1})$. It follows that we have a violation to validation 2c, since $b \in S(a_j)$ where $\gamma(a_j)$ is a prefix of $\gamma(b)$ and $\gamma(b)$ is a prefix of $\gamma(a_{j+1})$ (see Figure 19). Hence, we reached a contradiction to the hypothesis that V does not abort in the intermediate validations, and so $\gamma(b)$ cannot be a prefix of $\gamma(u)$.

Because $\gamma(b)$ is not a prefix of $\gamma(u)$, the maximal common prefix of $\gamma(u)$ and $\gamma(b)$ is a proper prefix of $\gamma(b)$. We know that $b \in S(u)$ because b is a child of z, which is an ancestor of u. Hence, by validation 2b, the transcript of u and the transcript of $b \in S(u)$ are prover consistent. It follows that the maximal common prefix of $\gamma(u)$ and $\gamma(b)$, which is a proper prefix of $\gamma(b)$, ends with a prover message.

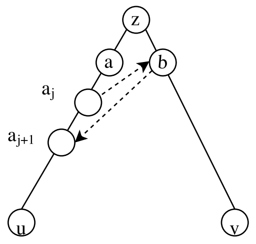

Figure 19: The dashed arrow pointing from b to a_{j+1} represent the fact that $\gamma(b)$ is a prefix of $\gamma(a_{j+1})$, and similarly that $\gamma(a_j)$ is a prefix of $\gamma(b)$.

Lastly, note that from validation 2a the transcript of each node in the path from b to v is a prefix of the transcript of its parent. Thus, $\gamma(b)$ is a prefix of $\gamma(v)$. It follows that the maximal common prefix of $\gamma(v)$ and $\gamma(u)$ is contained in the maximal common prefix of $\gamma(u)$ and $\gamma(b)$, so it ends with a prover message (See Figure 20). ■

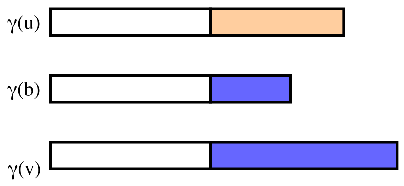

Figure 20: The maximal common prefix between $\gamma(u)$ and $\gamma(b)$ appears in white. Since $\gamma(b)$ is a prefix of $\gamma(v)$ the maximal common prefix of $\gamma(u)$ and $\gamma(v)$ is the same as the former.

Acknowledgments

I would like to thank my advisor, Prof. Oded Goldreich, for his encouragement to approach the problem studied in this paper, as well as for his guidance and support in the research and the writing process. I also thank Dr. Guy Rothblum for useful discussions.

References

[1] Laszlo Babai. Trading Group Theory for Randomness. In *17th ACM Symposium on the Theory of Computing*, pages 421–429, 1985.

[2] Laszlo Babai and Shlomo Moran. Arthur-Merlin Games: A Randomized Proof System and a Hierarchy of Complexity Classes. *Journal of Computer and System Science*, Vol. 36, pages 254–276, 1988.

[3] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Randomness in Interactive Proofs. *Computational Complexity*, Vol. 4, No. 4 (1993), pp. 319–354.

[4] Oded Goldreich. *Computational Complexity: A Conceptual Perspective.* Cambridge University Press, 2008.

[5] Oded Goldreich and Maya Leshkowitz. On Emulating Interactive Proofs with Public Coins. ECCC TR16-066, 2016.

[6] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. *Journal of the ACM*, Vol. 38, No. 3, pages 691–729, 1991. Preliminary version in *27th FOCS*, 1986.

[7] Oded Goldreich, Salil Vadhan, and Avi Wigderson. On interactive proofs with a laconic provers. *Computational Complexity*, Vol. 11, pages 1–53, 2002.

[8] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing*, Vol. 18, pages 186–208, 1989. Preliminary version in *17th STOC*, 1985. Earlier versions date to 1982.

[9] Shafi Goldwasser and Michael Sipser. Private Coins versus Public Coins in Interactive Proof Systems. *Advances in Computing Research: a research annual*, Vol. 5 (Randomness and Computation, S. Micali, ed.), pages 73–90, 1989. Extended abstract in *18th STOC*, 1986.

[10] Adam R. Klivans, Dieter van Melkebeek. Graph Nonisomorphism Has Subexponential Size Proofs Unless the Polynomial-Time Hierarchy Collapses. *SIAM Journal on Computing* Vol. 31, No. 5, pages 1501–1526,2002.

[11] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, Vol. 39, No. 4, pages 859–868, 1992. Extended abstract in *31st FOCS*, 1990.

[12] Ronen Shaltiel, Christopher Umans. Low-End Uniform Hardness versus Randomness Tradeoffs for AM. *SIAM Journal on Computing* Vol. 39, No. 3, pages 1006–1037, 2009.

[13] Adi Shamir. IP = PSPACE. *Journal of the ACM*, Vol. 39, No. 4, pages 869–877, 1992. Preliminary version in *31st FOCS*, 1990.