THEORY OF COMPUTING, Volume 18 (13), 2022, pp. 1–65
www.theoryofcomputing.org

SPECIAL ISSUE: RANDOM 2018

Round Complexity Versus Randomness Complexity in Interactive Proofs

Maya Leshkowitz*

Received September 19, 2018; Revised January 15, 2022; Published June 7, 2022

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 $O(r(n)/\log r(n))$. Furthermore, the randomness complexity is preserved up to a constant factor, and the resulting interactive proof system has perfect completeness.

An extended abstract of this paper appeared in the Proceedings of Proceedings of the 22nd International Conference on Randomization and Computation (RANDOM'18) [11].

*Work done while the author was at Weizmann Institute of Science

ACM Classification: F.1.3

AMS Classification: 68Q10, 68Q15, 68Q87

Key words and phrases: complexity theory, interactive proofs

DOI: 10.4086/toc.2022.v018a013

Contents

1 Introduction

The notion of interactive proof systems, put forward by Goldwasser, Micali, Rackoff [8] and Babai [1], and the demonstration of their power by Lund, Fortnow, Karloff, Nisan [12] and Shamir [14] 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 MA (defined by Babai [1]), are essentially the randomized version of 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 round complexity in interactive proof systems has some well-known results: Babai introduced public coin interactive proofs (a. k. a. Arthur–Merlin proofs) in [1], and showed that the round complexity of any bounded-round Arthur–Merlin proof can be reduced to AM (a two-message Arthur–Merlin proof). Babai and Moran extended this result in [2] and showed that the round complexity of Arthur–Merlin proofs for instances of length n can be reduced by any constant factor [2], assuming we keep at least one AM alternation. The public coin transformation of Goldwasser and Sipser [9] (which essentially preserves the number of rounds) extends this result to private-coin interactive proof systems. It is also known that a stronger round reduction is quite unlikely. 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.)

Regarding the completeness in interactive proofs, the fact that every interactive proof system can be transformed into one with perfect completeness was shown in [4].

[1]We refer to unconditional results and not to the long line of research on randomness versus hardness trade-offs that relies on uniform or non-uniform assumptions, see, e. g., [10],[13].

1.1 Round complexity versus randomness complexity

A natural question, which to the best of our knowledge has not been considered before, is what is the relation between the two foregoing complexity measures. We do suspect that there are cases when the randomness complexity cannot be made smaller than the round complexity without trivially increasing the number of rounds. This is because 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 (since this reduces to P). But *can the round complexity be greater than the randomness 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 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 the 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's messages may have very little information content (from the prover's point of view), and so in principle we could 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 content. These are the issues we deal with when showing that even in general interactive proof systems, randomness complexity $r(n)$ yields round complexity $O\left(r(n)/\log r(n)\right)$. Hence, in interactive proof systems, *the round complexity is smaller than the randomness complexity*.

Theorem 1.1 (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 $O(r(n)/\log r(n))$ 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.

1.1.1 Conditional tightness

The round complexity obtained by Theorem 1.1 is the best one may hope for at this time, since a result asserting round complexity $o(r(n)/\log r(n))$ for any set that has an interactive proof system of randomness complexity $r(n)$ would yield an unexpected result that seems currently out of reach. Specifically, it would place SAT in coAM-TIME$(2^{o(n)})$, which contradicts common beliefs. The full reasoning is as follows:

[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.

1. A variant of the celebrated interactive proof system for $\overline{\mathtt{SAT}}$ [14] 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 order $\mathrm{poly}(n)$. See [7, Sec 8].)[3]

2. On the other hand, any set having an m-round interactive proof system is in $\mathsf{AM\text{-}TIME}(n^{O(m)})$, 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 $\mathsf{AM\text{-}TIME}(2^{o(n)})$, whereas $\mathsf{AM\text{-}TIME}(T)$ may equal $\mathsf{NTIME}(\mathrm{poly}(T))$.

1.2 On the proof of Theorem 1.1

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 $O(r(n)/\log r(n))$ 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

We describe the strategies of the new prover P and verifier V in iterations, where each iteration may correspond to several rounds of the original protocol of prover P_0 and verifier V_0. The idea of the emulation protocol is that, in every iteration, we would like P to send possible continuations of the current transcript (each possibly describing execution segments of several rounds) that reveal much information about the verifier's random coins. Hence, P sends maximal transcripts such that each account for a noticeable fraction of the residual probability mass.[4] The verifier V then checks if one of these transcripts is consistent with the strategy of V_0 determined by the values of its random coins, which were tossed upfront. (That is, does the transcript contain the messages that V_0 would send using the private coin tosses of V and the previous messages.) If so, V picks a maximum transcript consistent with the strategy of V_0, and they continue their interaction from that point. Otherwise, V sends its next message (based on the aforementioned coins) without using the continuations suggested by P. We stress that the

[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 polynomial in n/ℓ variables, 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 $p = \mathrm{poly}(n)$, where p is selected by the verifier at random among such primes.

[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.

only source of V's randomness is its private coins tossed upfront, which are used to determine the continuation of the transcript in each subsequent iteration.

We elaborate on how the prover P 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 maximal continuation of γ that is a "heavy continuation." By a heavy continuation γ', we mean that γ' has probability mass greater than $p(\gamma)/r(n)$, when subtracting from it the probability mass of the continuations of γ that were either sent by P in previous iterations, or previously in this iteration. A heavy continuation is maximal in the sense that all its continuations are not heavy (probability mass not greater than $p(\gamma)/r(n)$). Note that there are at most $r(n)$ heavy continuations of γ, so the prover can send them to the verifier (who is computationally bounded).

This conditioning allows P to send several continuations of the transcript, some of which may also be continuations of the others. Consider for example the case that the prover sends $\gamma\alpha_1\beta_1\alpha_2\beta_2$ 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 $p(\gamma)/r(n)$. The reasoning is as follows. If the verifier V chooses one of the transcripts suggested by the prover P, then on the *next* iteration the residual probability mass of each of its continuations is lower than $p(\gamma)/r(n)$. Otherwise, this continuation should have been suggested by the prover on the previous iteration. If V 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 $p(\gamma)/r(n)$, otherwise the continuation $\gamma\widetilde{\alpha_1}$ should have been suggested by P.

It follows that after $O(r(n)/\log r(n))$ iterations the probability of the transcript generated is at most $(1/r(n))^{r(n)/\log r(n)} = 1/2^{r(n)}$, which means that there is a unique value of the coin tosses consistent with the transcript. Hence, a complete transcript is generated and V can reject or accept at this point. It is easy to show that if P follows the prescribed emulation, then the verifier V 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 for proving soundness. It is essential to include validation checks that guarantee that the transcripts provided by P are consistent with some legal prover P_0 strategy for the original protocol.[5] This means that if P provides two transcripts that share a prefix, this common prefix must end with a prover's message and diverge only depending on the verifier's response. This implies that the prover P_0 answers in the same way to the same verifier V_0's messages, which means that P's strategy is consistent with some prover P_0 of the original proof system and so the soundness of the original proof system is maintained.

[5]Otherwise P can provide different transcripts tailored to the verifier's private coin tosses, whereas the original prover could not have employed such a strategy since it is not exposed to the private coin tosses.

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 (and have the verifier reject right away if the sum of these probabilities does not match the claimed probability of the transcript as determined before the current iteration). In the last iteration of the emulation, 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 to the soundness when the prover provides wrong values for the residual probabilities. As for the second issue, note that maliciously raising the claimed 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 difficulties in the next iteration. Indeed, a careful analysis shows that actually, the prover gains nothing by such behaviour, 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 have the prover pack many transcripts into a single auxiliary message. 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 iteration). 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 will 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 (in order to guarantee soundness).

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 Relation to previous public coin emulation protocols

The basic idea used in emulating a private coin interactive proof by a public coin one is having the prover assist the verifier in generating its next message using public coins, instead of relying on private coins to generate the messages. Goldwasser and Sipser, who first suggested this emulation strategy [9], cluster the potential messages that the original verifier could have sent on the next round into clusters according to the (approximate) number of coin sequences that support each message. A constant-round, public-coin sampling protocol is utilized in order to sample from the cluster of messages that have the largest number of coin sequences. This emulation succeeds assuming a sufficiently large initial gap between completeness and soundness, which is obtained at a cost of blowup of $\operatorname{poly}(n)$ factor in randomness complexity. Thus, the original public coin emulation preserves the round complexity of the private coin interactive proof but not the randomness complexity.

An alternative public coin emulation, suggested by Goldreich and Leshkowitz [5], is similar to the original emulation but differs in the way the chosen cluster of messages (from which the sampling is performed) is determined. Whereas in the original emulation the chosen cluster is deterministically determined as the one with the largest number of coins, in the alternative emulation the chosen cluster is selected probabilistically according to its weight (i. e., the number of coin tosses in the cluster). Therefore, the alternative emulation gets closer to sampling from the actual distribution of prover verifier transcripts. Consequently, it requires a smaller initial gap between completeness and soundness. Similarly to the original emulation, the alternative emulation preserves the round complexity but not the randomness complexity, although the blowup in randomness complexity is slightly reduced due to the decrease in the completeness/soundness gap.

Our emulation protocol was designed with the goal of giving an upper bound on the round complexity in terms of the randomness complexity, while preserving the randomness complexity. To this end, the way the prover assists the verifier in generating its next message is different from the method used in the aforementioned emulations. The prover supplies the verifier with heavy continuations, which may correspond to multiple interaction rounds, that account for a noticeable fraction of the residual weight. In addition to the heavy continuations, the prover clusters the messages that each account for a small fraction of the residual weight using a succinct representation that supports the verification that messages in each cluster are disjoint. The method from the alternative emulation is then used to probabilistically select from the joint set of clusters and heavy continuations. When a cluster is selected, instead of using a sampling protocol to sample a message from the cluster, the emulation protocol is executed iteratively until reaching a heavy continuation (which corresponds to a message of the original interactive proof system). For this reason, the emulation protocol does not preserve the round complexity, which may increase by a $\operatorname{poly}(n)$ factor. However, addressing our initial concern, the combination of message generation and selection methods ensure that the randomness complexity $r(n)$ is preserved up to a constant factor and the new round complexity is at most $O(r(n)/\log r(n))$.

The original and alternative public coin emulations did not obtain perfect completeness since they used a sampling protocol for sampling a message from a cluster, which may fail even when the prover is honest. Since our emulation does not utilize such a protocol, we obtain perfect completeness for free.

An additional known public coin emulation is obtained using the LFKN–Shamir protocol [12, 14] to derive a public coin interactive proof system with perfect completeness to any set in PSPACE, and applying the protocol on the trivial PSPACE emulation of the private coin interactive proof system. This public coin emulation with perfect completeness is derived at a cost of polynomial round complexity and randomness complexity.

1.4 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 $O\left(r(n)/\log r(n)\right)$ 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 A, we show how to emulate the existing interactive proof system with a private coin emulation protocol that has $O\left(r(n)/\log r(n)\right)$ 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 A 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.

In Appendix B, we show an additional way to emulate the existing proof system with a public coin one that has perfect completeness. We stress that this is emulation is weaker than the one in the main theorem since it does not reduce the round complexity.

The two emulations in Appendix A and Appendix B together can be used to derive an alternative proof of the main theorem. This alternative proof of the main theorem is more modular, splitting the transformation into two simpler independent steps. We show how this can be done in Appendix B.

2 Preliminaries

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

Definition 2.1 (Interactive Proof Systems)**.** Let $c, s : \mathbb{N} \to [0,1]$ such that $c(|x|) \geq s(|x|) + 1/\text{poly}(|x|)$. An interactive proof system for a set S is a two-party game, between a verifier, denoted V, executing a probabilistic polynomial-time strategy, and a prover, denoted P, executing a *(computationally unbounded)* strategy, satisfying the following two conditions:

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

When c and s are not specified, we mean $c \equiv 2/3$ and $s \equiv 1/3$. We denote by 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 A.1 is similar to the one described here, except for the definition of weights.

Fixing an interactive proof between a prover P and a verifier V, and an instance x of length n, we describe the possible prover-verifier interactions on common input x using a tree $T_{P,V}$ whose height corresponds to the number of rounds of interaction. For some $\ell = \ell(n)$, we assume without loss of generality that in round i the verifier sends a message $\alpha_i \in \{0,1\}^\ell$, and the prover responds with a message $\beta_i \in \{0,1\}^\ell$. We can also assume, without loss of generality, that the prover's strategy is deterministic and fixed. Each node v on level j represents a possible prover-verifier transcript for the first j rounds of the interaction. The branching of $T_{P,V}$ represents the possible ways to extend the transcript to the next round. The number of ways to extend the transcript is the number of possible messages of the verifier in the next round since the prover is deterministic. 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 in $T_{P,V}$ 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 leads the verifier either 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 repetitions of an interactive proof system with standard completeness and soundness parameters, we can assume that the interactive proof system has completeness parameter 9/10 and soundness parameter 1/10.[6] Note that this blows up the randomness complexity only by a constant factor (as compared to our interactive proof

[6]The new interactive proof system will work by performing multiple invocations of the original interactive proof system. The verifier will accept if the majority of the original runs are accepting.

for the standard $1/3, 2/3$ parameters). Therefore, if x is a yes-instance then at least $9/10 \cdot 2^{r(n)}$ of its leaves represent accepting runs, and if x is a no-instance then at most $1/10 \cdot 2^{r(n)}$ 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 3.1 (Weight of a leaf)**.** Let u be a leaf in $T_{P,V}$ with transcript $\gamma(u)$ which corresponds to the full transcript of the interaction of P and V on input x, when V uses coin sequence ρ; that is,

$$\gamma(u) = (\alpha_1, \beta_1, \ldots, \alpha_m, \beta_m, (\rho, \sigma)) \tag{3.1}$$

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 3.2 (Weight of a node)**.** The weight $w(u)$ of a node u in $T_{P,V}$ is the sum of the weights of the leaves that are descendants of u.

Note that $w(u)$ is proportional to 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 have explained how to represent the possible executions of an m-round interactive proof system between prover P_0 and verifier V_0 on some instance x, where the protocol utilizes $r(n)$ coins. This results in a protocol tree T_{P_0,V_0} of height m with $2^{r(n)}$ leaves. Our goal is to transform this protocol tree to an emulation tree, denoted by E_{P_0,V_0}, that defines a prover strategy for a $O\left(r(n)/\log r(n)\right)$-round public coin emulation protocol. This transformation is done using the `Build_Tree` procedure. First, we describe how the emulation tree is used in the very restricted case where the protocol tree is already suitable for our proposed public coin protocol 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 A.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.

4.1.1 The protocol tree is of height $O\left(r(n)/\log r(n)\right)$ 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 T_{P_0,V_0} is at least $c \cdot 2^{r(n)}$ (where c is the completeness bound). The emulation is initiated at the root of T_{P_0,V_0} and on each round of the emulation the prover assists the verifier at progressing one step down the T_{P_0,V_0}. (This assistance is required because the verifier does not have access to T_{P_0,V_0}.) Each round consists of the prover providing the verifier with the descriptions of the children of the current node u that was 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 claimed 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 $1+1/r(n)$ using $O\left(\log r(n)\right)$ 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 according to the coin tosses revealed, 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 actual weight. Namely, the actual 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 bounds of the original interactive proof system. Thus, there is a multiplicative gap of s/c between the actual weight and the one claimed. We can show that, in expectation, this gap is maintained throughout the emulation, up to a factor of $(1 + 1/r(n))^{r(n)} < \mathrm{e}$ 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 $\mathrm{e}s/c$ which is smaller than $1/3$ for a suitable choice of s and c.

4.1.2 The degree of the protocol tree is bounded by poly(n)

In this case the height of the protocol tree T_{P_0,V_0} may be asymptotically larger than $r(n)/\log r(n)$. We create a new tree of height $O\left(r(n)/\log r(n)\right)$ 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**, which we denote by E_{P_0,V_0}. The nodes in the E_{P_0,V_0} are nodes in T_{P_0,V_0}. However, the children of a node u in E_{P_0,V_0} may be non-immediate descendants of u in T_{P_0,V_0}.

We start with a protocol tree T_{P_0,V_0} rooted at r whose weight is $w(r)$, and on each step we modify this tree towards creating an emulation tree E_{P_0,V_0}. We define a **heavy descendant** of node r to be a node whose weight is at least $w(r)/r(n)$ and the weight of each of its children is smaller than $w(r)/r(n)$. Note that there are at most $r(n)$ heavy descendants for each node in the tree.

We modify T_{P_0,V_0} 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.

The `Build_Tree(r)` begins by identifying the heavy descendants of r (they can also be children of r in $T^r_{P_0,V_0}$), which will become **heavy children** of r. Denote by $T^u_{P_0,V_0}$ the intermediate tree after we identify and set u as a heavy descendant of r. This way we have a notation for every transformation of the tree in which we set an additional child in the tree. That is, if the next tree after $T^u_{P_0,V_0}$ is $T^w_{P_0,V_0}$ then in $T^w_{P_0,V_0}$ both u and w are heavy children of r. After we identified the children of r in the new emulation tree, we call the `Build_Tree` procedure recursively on each child of r, which creates an emulation tree rooted at that node.

Observe that in the final emulation tree E_{P_0,V_0}, for each node u, the weight of each grandchild of u is at most $w(u)/r(n)$. This is because if v is a heavy child of u in E_{P_0,V_0}, then the weight of each descendant of v in $T^u_{P_0,V_0}$ is at most $w(u)/r(n)$. Since the children of v in the emulation tree are descendants of v in $T^u_{P_0,V_0}$ the claim holds. Otherwise, v is a non-heavy child of u, so its weight is smaller than $w(u)/r(n)$ and hence the weight of each child of v is also smaller than $w(u)/r(n)$.

It follows that the height of E_{P_0,V_0} is $O\left(r(n)/\log r(n)\right)$. The number of children of each node is at most poly(n), because we add at most $r(n)$ heavy children to the original children of each node, and $r(n)$ is at most polynomial in n. Hence, the emulation tree has properties similar to the protocol tree in the previous paragraph, so it is suitable for our public coin emulation described in the previous subsection.

4.1.3 The general case

In general, the degree of the protocol tree T_{P_0,V_0} 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 E_{P_0,V_0} is $O\left(r(n)/\log r(n)\right)$, but its degree may be super-polynomial in n (due to the original children). Hence, we will not be able to perform the emulation by having the prover provide the verifier with the descriptions of the children of the current node. Thus, we also need to make sure we create E_{P_0,V_0} so that its degree is 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_{P_0,V_0} may become non-immediate descendants in the next intermediate tree, and non-immediate descendants of r may become immediate children of r in the next intermediate tree (due to lifting).

Determining the children of r is done in two steps, as part of the `Build_Tree(r)` procedure. The first step is identifying the heavy descendants $v_1, \dots, v_k$ of r and lifting them to be children of r, creating heavy children. This is done by creating intermediate trees $T^{v_1}_{P_0,V_0}, \dots, T^{v_k}_{P_0,V_0}$, through a sequence of editing steps. On each step, we create a new intermediate tree and edit it on the next step, by identifying and moving the next heavy descendant. 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 $w(r)/r(n)$ and at most $2w(r)/r(n)$ (except for, possibly, the last group which is only required to have weight smaller than $2w(r)/r(n)$). We create a new **interval child** v for each such group, where the children of v are the nodes in the group.

After creating all the interval children of r, the children of r in the final emulation tree E_{P_0,V_0} are exactly the children of r in $T^z_{P_0,V_0}$, where z is the last interval child of r created. The number of children r has is at most $r(n)$, since the weight of each child of r (except for possibly the last interval child) is at least $w(r)/r(n)$. Next, the procedure is called recursively on the children of r in $T^z_{P_0,V_0}$ 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 T_{P_0,V_0}.

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 intermediate 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 descendants that are grouped under this node.

We show in the analysis that the height of E_{P_0,V_0} is $O\left(r(n)/\log r(n)\right)$. The degree of nodes in E_{P_0,V_0} is at most $r(n)$, hence it is suitable for public coin emulation like in the previous subsection.

(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 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 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 and place them as children of u. 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 4.1 (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. $w(v) \geq w(u)/r(n)$
2. Either v is a leaf, or for each child z of v it holds that $w(z) < w(u)/r(n)$.

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) so it is placed directly under u.

See Figure 1.

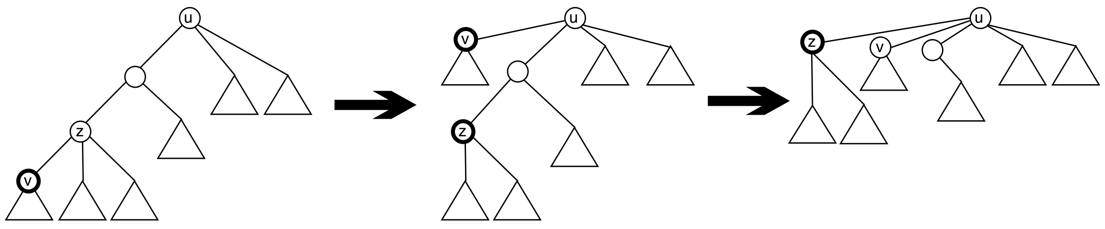

Figure 1: `Build_Heavy`: In the first step, v is identified as a heavy descendant of u and moved to be a heavy child of u. In the second step, z is identified and moved to be a heavy child of u. The triangles represent subtrees of the original tree.

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 $w(u)/r(n)$ 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 the 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 $W(s', e) \geq w(u)/r(n)$ 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 $2w(u)/r(n)$ and at least $w(u)/r(n)$, except possibly for the last interval child, whose weight is at least 1.

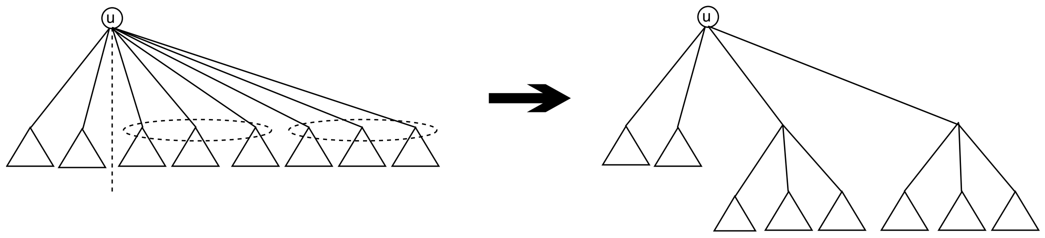

Figure 2: `Build_Interval`: The left diagram represents the tree before the `Build_Interval` procedure. The nodes to the left of the dashed line are heavy children of u. The group of nodes inside each dashed circle are united under an interval node. The tree on the right is the result of applying the `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 a heavy child of u and hence $\gamma(v) = \gamma(u)\alpha_1\beta_1, \dots, \alpha_k\beta_k$ for some $\alpha_1, \beta_1, \dots, \alpha_k, \beta_k \in \{0,1\}^\ell$, 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 4.2 (Node degree). *Each node u in the final emulation tree E_{P_0,V_0} has at most $r(n)$ children.*

Proof. Note that we call `Build_Tree` on every node in E_{P_0,V_0}. By definition, the weight of each heavy child of u is at least $w(u)/r(n)$. The weight of each interval child is also at least $w(u)/r(n)$ except for possibly the last interval child whose weight is non zero. Therefore, the number of children is at most $r(n)$. (If there were $r(n)+1$ children or more, then the $r(n)$ first children would have weight of at least $w(u)/r(n)$, 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 4.3 (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 $2w(u)/r(n)$.*

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 intermediate 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 each child of v in T_u is at most $w(u)/r(n)$. Thus, the weights of the children of v in T_u is also at most $w(u)/r(n)$. 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 $w(u)/r(n)$. Otherwise, z is an interval child of v so its weight is at most $2w(v)/r(n)$, which is at most $2w(u)/r(n)$.

In case v is an interval child of u, its weight is at most $2w(u)/r(n)$. Hence, the weight of each grandchild of u which is a child of v, is also at most $2w(u)/r(n)$. □

Corollary 4.4 (Corollary to Claim 4.3)). *The height of the final emulation tree E_{P_0,V_0} is $O\left(r(n)/\log r(n)\right)$.*

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 4.3, the weight of a node in level $2i$ of the emulation tree is at most $2^{r(n)}/\left(\frac{r(n)}{2}\right)^i$. Taking

$i = \lceil r(n)/\log(r(n)/2) \rceil$ 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 4.3 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 $2 \cdot \lceil r(n)/(\log(r(n)) - 1) \rceil + 1$. □

Claim 4.5 (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 weight 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

5.1 Emulation preliminaries

Next, we describe the strategy of the designated prover P and verifier V in the new protocol ("emulation"). 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 performs 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 part of the emulation tree seen so far. 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 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 are possibly children of u in the emulation tree. For nodes in the original protocol tree, v is possibly 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 possibly 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 the transcript of v equals the transcript of u and the range of v is a partial range of the range of

u, then v is possibly a child of u which the verifier regards as an **interval child**. If the transcript of u is a proper prefix of the transcript of v and if $\gamma(u) = \alpha_1^u\beta_1^u, \dots, \alpha_i^u\beta_i^u$, and α_{i+1}^v (the $i+1$ verifier's message in the transcript of v) is in the range of u then v is possibly a child of u, which the verifier regards as a **heavy child**.

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 5.1 (Transcript Descendant)**.** Denote by u and v nodes in the emulation tree with transcripts $\gamma(u) = \alpha_1^u\beta_1^u, \dots, \alpha_i^u\beta_i^u$ and $\gamma(v) = \alpha_1^v\beta_1^v \dots \alpha_j^v\beta_j^v$ 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 $\alpha_{i+1}^v \in R(u)$

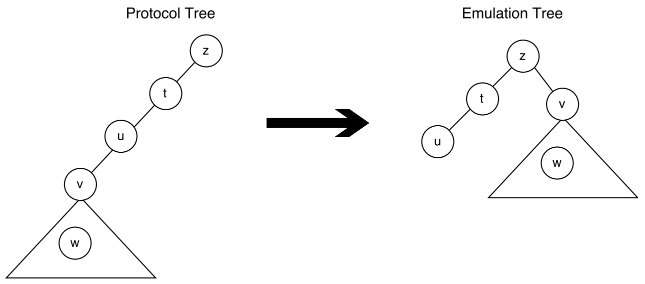

Figure 3: Truncation during `Build_Tree`. The node v is identified as a heavy descendant of z, so it is moved (along with the subtree that is rooted at it) to be a heavy child of z.

It is easy to show that for every node u in E_{P_0,V_0}, every descendant of u is a transcript descendant of u. The proof is similar to the proof of validation (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 5.2 (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, z is a transcript descendant of u.*

Proof. 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 $\alpha_{i+1}^v \in R(u)$, 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).

□

The condition of v being a transcript descendant of u is not sufficient to guarantee that v is possibly 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 is possibly 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 **seen-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 $S(u)$, which the verifier sees up to the iteration that the input node is u, are a subtree of the emulation tree that is composed of a path from the root of the tree to u, augmented with the children of the nodes in the path. See Figure 4.

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

Definition 5.3 (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 or equal to the length of the shorter transcript, if $(\alpha^u_1\beta^u_1, \ldots, \alpha^u_j) = (\alpha^v_1\beta^v_1, \ldots, \alpha^v_j)$ then $\beta^u_j = \beta^v_j$).

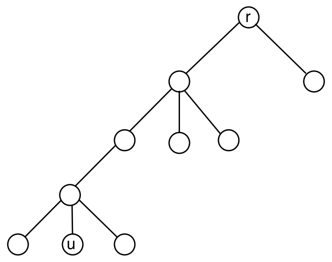

Figure 4: The nodes in the seen-list, S, in the iteration that the input node is u.

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.

We stress that we give an honest-prover centric description of the emulation protocol. This means that Step 1 describes the intended behaviour from an honest prover to provide the descriptions of the heavy children. A dishonest prover may deviate from this step, and the purpose of the verifier's checks to see that even if there was a deviation the prover can not succeed in claiming that a no-instance is a yes-instance with high probability.

5.2 Emulation construction

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

Construction 5.4. (the ith iteration) On input a non-leaf node u and seen-list S.

1. The prover provides the descriptions of the children $v_1, \dots, v_d$ of u:

$$(\gamma(v_1), R(v_1), w(v_1)), \dots, (\gamma(v_d), R(v_d), w(v_d))$$

2. The verifier performs 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,

$$w(u) = \sum_{j=1}^{d} w(v_j) \tag{5.1}$$

(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 $\gamma\left(v_j\right) = \gamma(u)$; that is, if $R\left(v_j\right) \neq [0^\ell, 1^\ell]$, then $\gamma\left(v_j\right) = \gamma(u)$ 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 5.1.) Specifically, for $v \in S$ then if v is a transcript descendant of u, then v_j 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. (In particular, v_j should not be equal to w shown in the figure.)
(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 $R\left(v_j\right) \cap R\left(v_k\right) = \emptyset$.

(g) For each $j \in [d]$ the verifier checks that $\gamma(v_j)$ is **prover consistent** (see Definition 5.3) 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 $O(\log r(n))$ coin tosses. That is, J is such that

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

We can only afford to use $O(\log r(n))$ public coins per iteration since the height of the emulation tree, and hence the number of iterations of the emulation, is $O\left(r(n)/\log r(n)\right)$ and we want the total number of public coins to be $O(r(n))$. Hence, we compromise on sampling each child with probability proportional to $w(v_j)$, and instead sample with approximate probability. See the explanation for approximate sampling in Subsection 5.4.

4. The verifier adds all the children of u to the seen-list S; that is $S \leftarrow S \cup \{v_1, \ldots, v_d\}$. Denote by j the index the verifier choose in 3. 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 V_0 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 $\gamma\left(v\right) = \alpha_1\beta_1, \ldots, \alpha_m\beta_m$. 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.

(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).

5.3 The number of rounds and randomness complexity

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

5.4 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, $p_j = w(v_j)/\sum_{i=1}^{d} w(v_i)$. 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 $p'_j \le p_j \cdot (1 + 1/r(n))$. Moreover, the probability distribution J should be one that the verifier can sample from using $k = O\left(\log r(n)\right)$ coin tosses. Note that our method of approximation also satisfies $p'_j > p_j - 1/(r(n))^3$ for every $j \in [d]$, although the lower bound is not used in our work.

Let $k \in \mathbb{N}$ such that $2^{k-1} < (r(n))^3 \le 2^k$. Assume, without loss of generality, that p_d is the largest probability and thus $p_d \ge 1/d \ge 1/r(n)$. For $j < d$ define p'_j by rounding down p_j to the closest integer multiple 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 $p'_j \le p_j$. For $j = d$,

$$\begin{aligned} p'_d &= 1 - \sum_{i=1}^{d-1} \frac{\lfloor p_i \cdot 2^k \rfloor}{2^k} \\ &\le 1 - \sum_{i=1}^{d-1} \frac{p_i \cdot 2^k - 1}{2^k} \\ &= 1 - \sum_{i=1}^{d-1} p_i + (d-1) \cdot 2^{-k} \,. \end{aligned} \tag{5.2}$$

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

$$(d-1) \cdot 2^{-k} \leq r(n) \cdot (r(n))^{-3} = (r(n))^{-2} . \tag{5.3}$$

Because D is a probability distribution we get that $p_d = 1 - \sum_{i=1}^{d-1} p_i$ and so plugging Eq. (5.3) in Eq. (5.2) we get that $p'_d \leq p_d + 1/(r(n))^2$.

Using the fact that $p_d \geq 1/r(n)$ it follows that $1/(r(n))^2 \leq p_d/r(n)$ and hence

$$p'_d \leq p_d \cdot (1 + 1/r(n))$$

6 Analysis of the emulation

We show that the interactive proof system is transformed by the emulation protocol of Construction 5.4, 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 1/3.

6.1 Completeness

We claim that the emulation protocol of Construction 5.4 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 $9/10 \cdot 2^{r(n)}$ 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 $9/10 \cdot 2^{r(n)}$. 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 $9/10 \cdot 2^{r(n)}$ as well. Hence, P can make a valid initial claim of weight at least $9/10 \cdot 2^{r(n)}$

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 tree as part of the `Build_Tree` procedure.

We denote by $T^v_{P_0,V_0}$ the tree during construction, after the step that v is identified either as a heavy child, or created as an interval child in the tree. We can list the intermediate trees according to the order the nodes are placed starting from the root r and until the protocol tree is transformed to an emulation tree: $L = T_{P_0,V_0} = T^r_{P_0,V_0}, T^{u_1}_{P_0,V_0}, \ldots, T^{u_m}_{P_0,V_0} = E_{P_0,V_0}$. For every tree in this list, we show that for every node u in the tree the validations when u is the input node of the iteration pass. Therefore, it will follow that the validations also pass for every node in the final emulation tree E_{P_0,V_0}.

When we say that a validation passes relative to a (possibly intermediate) tree $T^z_{P_0,V_0}$ and node u in $T^z_{P_0,V_0}$, we mean that if the tree $T^z_{P_0,V_0}$ had been used as an emulation tree then in the iteration which u is the input node, the validation would have passed. Note that possibly the degree of u in $T^z_{P_0,V_0}$ is not polynomial in n, and hence we can not use $T^z_{P_0,V_0}$ to define the emulation protocol, since the verifier should run in polynomial time. However, we can still prove that each validation passes for the intermediate trees assuming that the verifier was not polynomially bounded. Recall that the nodes in $T^z_{P_0,V_0}$, 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 $[0^\ell, 1^\ell]$.

Let $T^z_{P_0,V_0}$ be an intermediate tree from the list above, created when assigning the node z (either as an interval child or heavy child) as part of the `Build_Tree` procedure. We assume that the validation we are currently checking holds for every node in $T^z_{P_0,V_0}$, and show that it also holds in the next step of the construction, that is the next tree in the list L. 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 intermediate tree after this modification by $T^v_{P_0,V_0}$.

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 $T^v_{P_0,V_0}$, as well as nodes whose seen-list changes. Exactly which nodes are affected depends on the validation.

Remark 6.1. Let v be a node in some tree $T^z_{P_0,V_0}$ that the `Build_Tree` procedure has not been invoked on yet. Recall that the children of v in $T^z_{P_0,V_0}$ 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 $T^z_{P_0,V_0}$ 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 $T^z_{P_0,V_0}$. Thus, the subtree of $T^z_{P_0,V_0}$ rooted at v, denoted by $T^z_{P_0,V_0}(v)$, is a subtree of T_{P_0,V_0}. (This is true up to the point that v might be an interval node and thus not in T_{P_0,V_0}. In this case changing just the node v to the node v' defined above we get 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 $v_1, \dots, v_d$ provided by the prover as part of the emulation. The validations are numbered as in Construction 5.4. Assuming that the validations hold for every node in $T^z_{P_0,V_0}$, we shall prove that these validations hold for every node in $T^v_{P_0,V_0}$ (the next intermediate tree in the list).

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 Subsection A.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 $\gamma(v_i) = \gamma\left(v_j\right)$ then $R(v_i) \neq R\left(v_j\right)$.
First, note that in T_{P_0,V_0} all the nodes have distinct transcripts, and in particular for every node all of its children have distinct descriptions. Let $T^v_{P_0,V_0}$ be an intermediate tree in the list. We assume that this validation passes for all the trees prior to $T^v_{P_0,V_0}$ in the list and we show it also holds in $T^v_{P_0,V_0}$. Denote the parent of v in $T^v_{P_0,V_0}$ by z. The only node in $T^v_{P_0,V_0}$ that may have new children relative to the ones in the proceeding tree in the list L is z,

and hence from the assumption, it sufficed to show that the validation holds for z. If v is an interval child of z then its description is distinct from the other children of z since the ranges of the interval children are distinct. If v is determined as a heavy child of z, then it is a descendant of z in $T^z_{P_0,V_0}$. By Remark 6.1, $T^z_{P_0,V_0}(z)$ is a subtree of T_{P_0,V_0}. Hence, each node in $T^z_{P_0,V_0}(z)$ has a different description. All the heavy children of z in $T^v_{P_0,V_0}$ are in $T^z_{P_0,V_0}(z)$, and hence they have distinct descriptions. To conclude note that the heavy children have full range, and interval children have partial range, and hence in particular the heavy and interval children of z are distinct from each other.

(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 is a node that we determine as a child for z.) Assuming that Eq. (6.1) holds for every node in $T^z_{P_0,V_0}$, we shall show that it holds for every node in $T^v_{P_0,V_0}$ as well. We consider two cases, according to if v is a heavy child or interval child 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 $T^z_{P_0,V_0}(z)$ 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 because we subtract $w(v)$ off the weight of z_1, and we add an additional child v with weight $w(v)$. For the other nodes in $T^v_{P_0,V_0}$ 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, this is also true of all the other nodes in $T^v_{P_0,V_0}$.

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 5.1) of its parent. As before, we shall assume that every node in $T^z_{P_0,V_0}$ maintains the property that it is a transcript descendant of its parent, and show that this property is maintained in the tree $T^v_{P_0,V_0}$, after the assignment of v as a child of z. We only need to show this validation holds for z, which possibly has a new child v, and for v itself, which might not have been a node in $T^z_{P_0,V_0}$. The property holds for all the other nodes in $T^v_{P_0,V_0}$ from our assumption on $T^z_{P_0,V_0}$. We consider two cases:

- If v is a heavy child of z, then v was a descendant of z in $T^z_{P_0,V_0}$ that we move to be a child of z (along with its descendants) and set its range to be full range. By our assumption, each node in the path from z to v in $T^z_{P_0,V_0}$ is a transcript descendant of its parent, so by transitivity (see Claim 5.2), 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 in $T^v_{P_0,V_0}$ are transcript descendants of v. This is because the children of v are all children of z in $T^z_{P_0,V_0}$, such that their next verifier message is in the range of v. Hence, the children of v are transcript descendants of type ii of 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 $R\left(v_j\right) \neq [0^\ell, 1^\ell]$ then $\gamma\left(v_j\right) = \gamma\left(u\right)$ must hold. This holds because the only case where $R\left(v_j\right) \neq [0^\ell, 1^\ell]$ is if v_j is an interval child of u, and in this case $\gamma\left(v_j\right) = \gamma\left(u\right)$ because of how we create interval children. Note that once we create an interval child in some intermediate tree it stays a child of its parent throughout the subsequent tree transformations, and hence also in the final emulation tree.

(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. For every $y \in S$, the verifier checks that if y is a transcript descendant of u, then v_j should not be a transcript descendant of y. Note that if y is a transcript descendant of u then by validation (c) we know that y is not an ancestor of u.
In order to show that the validation is satisfied, we shall first define the notation of seen-list of a node, and then prove that the following claim holds for every node in the emulation tree E_{P_0,V_0}.

Definition 6.2 (Seen-list of a node.)**.** When we consider a tree T that is not the final emulation tree, and a node $v \in T$, then we regard the seen-list of v, denoted by $S(v)$, as the list containing the nodes that are ancestors of v, augmented with their children.

Claim 6.3. *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.

First, we shall show that the claim holds in the initial protocol tree T_{P_0,V_0}. Each node in T_{P_0,V_0} is a transcript descendant only 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. Thus, the claim holds vacuously. Next, we shall assume that the claim holds in the tree $T^z_{P_0,V_0}$ before creating a child v of z, and we show that it holds in the tree $T^v_{P_0,V_0}$ after the identification v as a heavy child of z, 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 $T^v_{P_0,V_0}$, it is enough to consider the nodes $c \in T^v_{P_0,V_0}$ who have new descendants or new nodes in $S(c)$ relative to the ones they had in $T^z_{P_0,V_0}$. The descendants of the nodes in $T^v_{P_0,V_0}$ are descendants of it in $T^z_{P_0,V_0}$. The only nodes in $T^v_{P_0,V_0}$ 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. Determining the heavy descendants of z is done by searching the tree from bottom to top (this is implied by the definition of heavy descendants, 4.1). It follows that the ancestors of v in $T^z_{P_0,V_0}$ which are transcript descendants of z have not been identified as heavy descendants of z (or any other node) at this point yet, and so in $T^z_{P_0,V_0}$ the node v is only a transcript descendant of its ancestors. Thus, if c is a node in $T^v_{P_0,V_0}$ such that $v \in S(c)$ and v is a transcript descendant of c, then c is an ancestor of v in $T^z_{P_0,V_0}$. It is left to check that the descendants of c in $T^v_{P_0,V_0}$ are not transcript descendants of v. From Note 6.1, it follows that the subtree of $T^v_{P_0,V_0}$ rooted at c, denoted by $T^v_{P_0,V_0}(c)$, is a subtree of T_{P_0,V_0}. Thus, because $v \notin T^v_{P_0,V_0}(c)$ (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.)

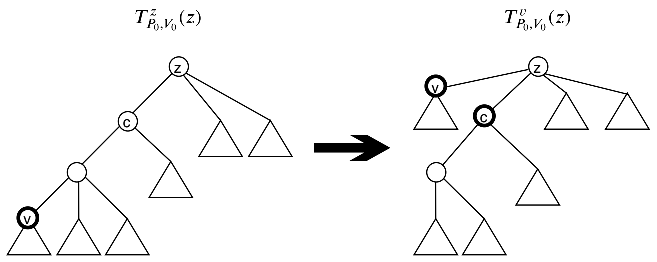

Figure 5: v is identified as a heavy child of z, and c is a descendant of z.

- Let v be an interval child created for z. We shall assume that the claim holds in the tree $T^z_{P_0,V_0}$ before the assignment of v, and show that it holds in $T^v_{P_0,V_0}$. It is enough to show that the claim holds when the node u is one of the following: either a new interval child v or a node in $T^v_{P_0,V_0}$ that has new descendants or a node in $T^v_{P_0,V_0}$ that has new nodes in their seen-list (relative to the ones in $T^z_{P_0,V_0}$). (For the other nodes in

$T^v_{P_0,V_0}$ the claim follows from our assumption regarding $T^z_{P_0,V_0}$.)
The only nodes in $T^v_{P_0,V_0}$ that have new descendants that they did not have in $T^z_{P_0,V_0}$ are the ancestors of v in $T^v_{P_0,V_0}$, which now have v as their descendant. Let d be some ancestor of v in $T^v_{P_0,V_0}$, and let $c \in S(d)$ be a transcript descendant of d. We need to show that v is not a transcript descendant of c. We know that v has children in $T^v_{P_0,V_0}$ otherwise it would not been created as an interval node, so let t be a child of v in $T^v_{P_0,V_0}$ (see Figure 6). It follows that t is a transcript descendant of v. Assume by 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 $T^z_{P_0,V_0}$. This is in contradiction to our hypothesis that the claim holds in $T^z_{P_0,V_0}$.

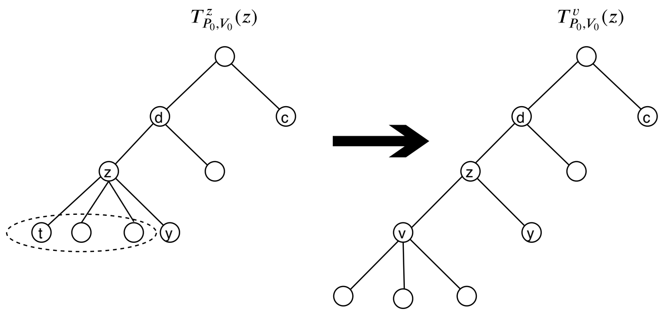

Figure 6: v is an interval child created by uniting 3 children of z.

The nodes whose seen-list S increases in $T^v_{P_0,V_0}$ relative to $T^z_{P_0,V_0}$ 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 (this follows from validation 2c and transitivity). 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 $T^v_{P_0,V_0}$ are not transcript descendants of b. There are two different options.

- Either $b \in S(z)$ in $T^z_{P_0,V_0}$, 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 $T^v_{P_0,V_0}$ is a descendant of z in T_{P_0,V_0}z. Hence, from the fact that the claim is true for z in $T^z_{P_0,V_0}$ we know that the descendants of v in $T^v_{P_0,V_0}$ 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 6.1, it follows that $T^v_{P_0,V_0}(v)$ is a subtree of T_{P_0,V_0}. Hence, from the fact that b is not in

$T^v_{P_0,V_0}(v)$ it follows that the transcript descendants of b are not in $T^v_{P_0,V_0}(v)$ either. In other words, the descendants of v in $T^v_{P_0,V_0}$ 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 $R\left(v_j\right) \cap R\left(v_k\right) = \emptyset$.
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 5.3) 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 it 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 4.5 the leaves of the emulation tree E_{P_0,V_0} are the leaves of the protocol tree whose weights are 1. Hence, $V_0\left(x, \rho, \alpha_1, \ldots, \alpha_m\right) = 1$, and for every $i = 1, \ldots, m$ it holds that $\alpha_i = V_0\left(x, \rho, \beta_1, \ldots, \beta_{i-1}\right)$. In addition $w(v) = 1$.

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 $1/3$. 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 $9/10 \cdot 2^{r(n)}$ (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 actual weight of the node, is at most $1/10 \cdot 2^{r(n)}$. We want to show that this gap is approximately maintained with high probability at least until the last iteration, and hence the verifier rejects.

In order to proceed with this analysis, we need to define the notion of "actual 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 proof system being emulated. However, choosing the prover of the original system is less straightforward. 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 A.4.2. The other two components of the soundness analysis of the public coin system (which define the notion of actual weight of a node, and bound the gap between the claimed and actual weight) 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 acceptance probability is at least as high. (Recall that we want to show that the verifier rejects with high probability. Hence, it suffices to take a deterministic prover which the verifier rejects with lower probability when interacting with, and showing that this probability is still high enough).

We can also 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. (The prover can compute the transcripts of the children such that they are different continuations of their parent and prover consistent with the list S instead of every node where the verifier would have aborted in the intermediate checks.) Since $\widetilde{P}$ is deterministic and the verifier doesn't abort until the final checks, the prover's strategy can be represented using an emulation tree which we denote by $E_{\widetilde{P}}$.

We show that we can extract a deterministic strategy $\widetilde{P_0}$ for the original prover using the strategy of $\widetilde{P}$. 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 $\widetilde{P_0}(x, \alpha_1, \ldots, \alpha_i) \coloneqq \beta_i$ for all $i \leq j$. 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 5.3).

For a node $u \in E_{\widetilde{P}}$ provided during the emulation, recall that we denote by $S(u)$ the seen-list (see Definition 6.2) the list of 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.

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 6.4 (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 the verifier does not abort until the final checks. Let ℓ be a node with a range and a transcript field 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.)

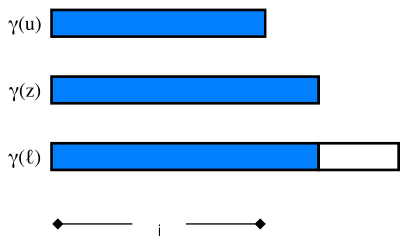

Figure 7: The transcripts of u,z and ℓ when $|\gamma(z)| \geq |\gamma(u)|$

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 $j > i$. For an index i and a node u denote by α_i^u the ith verifier's message in the transcript of u. Because $\gamma(z)$ is a prefix of $\gamma(\ell)$ it follows that $\alpha_{i+1}^z = \alpha_{i+1}^\ell$. From the fact that ℓ is a transcript descendant of type (ii) of u we know that $\alpha_{i+1}^\ell \in R(u)$. Thus, $\alpha_{i+1}^z \in R(u)$ 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 $R(u) = \left[0^\ell, 1^\ell\right]$ then $R(z) \subseteq R(u)$, so z is a transcript descendant of u of type (i). Similarly, if $R(z) = \left[0^\ell, 1^\ell\right]$ 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 validation 2c passes for every node in $E_{\widetilde{P}}$, 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.

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 $\gamma(z)$. To see why this is a contradiction consider two

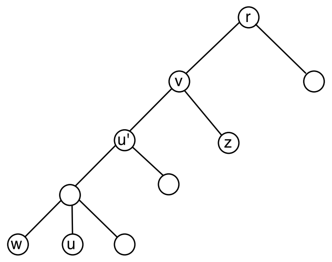

Figure 8: z and u are two nodes in $S(w)$ whose least common ancestor is v.

cases. 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.

Note that the following proof is a more involved version of the proof of Lemma A.9 in Section A.4.

Lemma 6.5 (Prover consistency of the emulation tree)**.** *If the strategy of the prover $\widetilde{P}$ for the new emulation is 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 be 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. We denote by $S'(v)$ the seen-list of the parent of v. That is, if u is the parent of v in $E_{\widetilde{P}}$ then $S'(v) = S(u)$. In the general case that v is a descendant of u then $u \in S'(v)$ 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 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. (If $v = b$ then $a \notin S'(v)$ but then $\gamma(v)$ and $\gamma(a)$ are prover consistent also from validation 2g because they are both children of z.) 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 $\gamma(b)$.

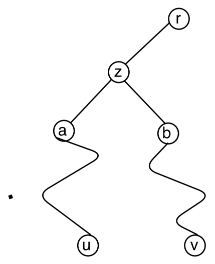

Figure 9: The subtree of $E_{\widetilde{P}}$ that contains u and v

First, consider the case that $\gamma(a)$ is not a prefix of $\gamma(b)$ and $\gamma(b)$ is not a prefix of $\gamma(a)$. It follows that 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.

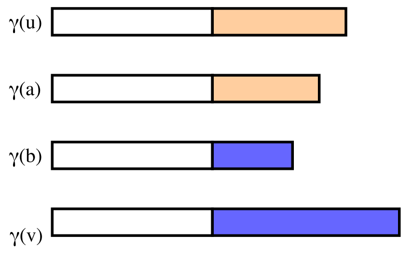

Figure 10: $\gamma(a)$ and $\gamma(b)$ agree on the prefix in white, while $\gamma(a)$ is a prefix of $\gamma(u)$ and $\gamma(b)$ is a prefix of $\gamma(v)$.

Next, consider the case when $\gamma(a) = \gamma(b)$, and assume that the transcript of a and b contain messages from i rounds. In this case both a and b are interval children of z. This is because at least one of them needs to have a partial transcript, otherwise validation 2a would be violated. Assume without loss of generality that a has a partial transcript. Because a has a partial transcript, from validation 2d the transcript of z is equal to the transcript of a and b, and the range of z is full. Hence, if the range of b is full then the description of b and z are the same, and in particular b is not a transcript descendant of z, which violates validation 2c. We have established that a and b are interval children of z and from validation 2f it follows that $R(a) \cap R(b) = \emptyset$. Note that u is a transcript descendant of a, and v is a transcript descendant of b of type ii, since the transcripts $\gamma(a)$ and $\gamma(b)$ are proper prefixes of $\gamma(u)$ and $\gamma(v)$ respectively. As a result, $\alpha^v_{i+1} \in R(b)$ 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 by validation 2d and hence $R(b) = [0^\ell, 1^\ell]$.

We claim that u is not a transcript descendant of b. Assume, by 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 6.4 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.

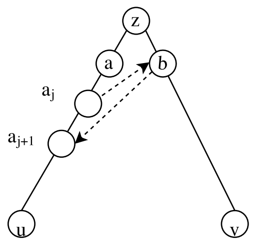

Figure 11: The dashed arrow pointing from b to a_{j+1} represent the fact that a_{j+1} is a transcript descendant of b, and similarly that b is a transcript descendant of a_j.

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 $E_{\widetilde{P}}$. 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). □

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

We want to introduce the notion of the "actual weight" of a node in $E_{\widetilde{P}}$. This "actual weight" should be proportional to the probability that the node represents a partial interaction of the

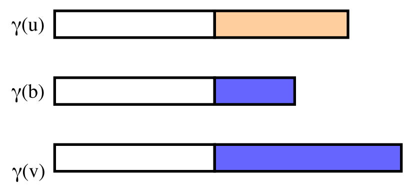

Figure 12: The maximal common prefix between $\gamma(u)$ and $\gamma(b)$ appears in white. Since $\gamma(b)$ is a prefix of $\gamma(v)$ the maximal common prefix of $\gamma(u)$ and $\gamma(v)$ is the same as the former.

verifier V_0 with the prover $\widetilde{P}_0$, which was extracted in the previous subsection, and that this partial interaction 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, and so they are not specific to the emulation tree and the way which it was constructed. 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 heavy nodes. Hence, instead of using the weight of a node in $T_{\widetilde{P_0},V_0}$ we consider the number of leaves whose weights are 1 and are transcript descendants of it.

Definition 6.6 (Actual 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. (In particular, $u \notin Trunc(u)$.)
We define $L^{\widetilde{P}}(u)$ as

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

We define the actual 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 actual weight of a node in the emulation tree $E_{\widetilde{P}}$, which asserts that the actual weight of a node is at least as large as the sum of the weights of its children.

Claim 6.7. *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 6.8. *For each distinct $i, j \in [d]$, it holds that $L^{\widetilde{P}}(v_j) \cap L^{\widetilde{P}}(v_i) = \emptyset$.*

Fact 6.9. *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 6.8. 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 by 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 6.4 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 $v_j \in Trunc\,(v_i)$, 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 $\ell \in L^1\left(v_j\right)$. 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 6.8.

Turning to the proof of Fact 6.9, 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 by contradiction that there exists $z \in Trunc\,(u)$ such that $\ell \in L^1\,(z)$ and hence $\ell \notin L^{\widetilde{P}}(u)$. Recall that $z \in Trunc(u)$ 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 6.4 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 $z \in S\left(v_j\right)$ and so $z \in Trunc\,\left(v_j\right)$. But because $\ell \in L^1\,(z)$ we get that $\ell \notin L^{\widetilde{P}}(v_j)$, in contradiction to our hypothesis.

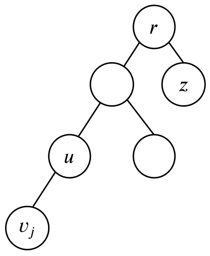

Figure 13: The emulation tree after z has been truncated and moved to be a chid of r.

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

6.2.3 Bounding the gap

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

Definition 6.10 (Gaps)**.** The gap for node $u \in E_{\widetilde{P}}$ denoted by $g(u)$, is the ratio between $W^{\widetilde{P}}(u)$, the actual weight of node u according to the strategy of $\widetilde{P}$, and the claimed weight $w'(u)$.

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

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.

Let v be a node chosen on the ith iteration of a random execution of the emulation protocol by $\widetilde{P}$ and V. We denote the value of the gap on the ith iteration by $g_i = g(v)$, and by g_0 the value of the gap of the root r.

We consider the m' iteration emulation protocol defined in Construction 5.4, 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 Claim 6.11 below, we analyze the change in the gap on the ith iteration, and show that it does not increase too much in expectation.

Claim 6.11. *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*

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

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

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

By the definition of the gap for node v_j,

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

According to Step 3 of the emulation protocol,

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

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

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

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 6.7 in Eq. (6.5) we get

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

The claim follows. □

6.2.4 Concluding the soundness 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 $W^{\widetilde{P}}(r) \leq 1/10 \cdot 2^{r(n)}$. Hence, if the prover claims that some no-instance is a yes-instance, then at the beginning of the emulation $w'(r) \geq 9/10 \cdot 2^{r(n)}$ whereas $W^{\widetilde{P}}(r) \leq 1/10 \cdot 2^{r(n)}$, thus $g_0 \leq 1/9$. 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 so $W^{\widetilde{P}}(v) = w'(v) = 1$ and so $g_{m'} = g\left(v\right) = W^{\widetilde{P}}(v)/w'(v) = 1$.

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

$$\mathbf{Pr}\left[G_{m'} \geq 1\right] \leq \mathbb{E}\left[G_{m'}\right] \ .$$

From Claim 6.11 we know that for every sequence of coin tosses $r_1, \ldots, r_{i-1}$ that determine g_{i-1}

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

Hence,

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

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

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

From property 4.4, the height of the emulation tree and hence the number of iterations of the new emulation is at most $m' = 2 \cdot \lceil r(n)/\log r(n) \rceil + 1$. For $n \geq 8$ the value of m' is at most $r(n)$ and thus,

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

where the last equality follows from the fact that $g_0 \leq 1/9$. Hence, the verifier accepts with probability of at most 1/3.

Appendix A 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. Moreover, we can use this weaker version as a component of an alternative proof of the main theorem, which we show in Appendix B.

Theorem A.1. *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 $O(r(n)/\log r(n))$ 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 A.1 we describe the protocol tree of the original proof system. The protocol tree is used to construct an emulation tree in Section A.2. In Section A.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 (A.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.

A.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 between prover P and verifier V and an instance x of length n, we describe the possible prover-verifier interactions of the system on common input x using a tree $T_{P,V}$ whose height corresponds to the number of rounds of interaction. For some $\ell = \ell(n)$, we assume without loss of generality that in each iteration 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 $T_{P,V}$ 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 $T_{P,V}$ 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 $T_{P,V}$ 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 A.2 (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 A.3 (Weight of a node)**.** The weight of a node u in the protocol tree $T_{P,V}$ is the sum of the weights of the leaves that are descendants of u.

Note that the weight of node in $T_{P,V}$, 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.

A.2 The emulation tree

Using the protocol tree T_{P_0,V_0}, we create a new tree of height $O\left(r(n)/\log r(n)\right)$ to guide the prover's strategy. We call this tree the **emulation tree** and we denote it by E_{P_0,V_0}. The nodes in E_{P_0,V_0} are nodes from T_{P_0,V_0}, however the children of a node u in E_{P_0,V_0} may be non-immediate descendants of u in T_{P_0,V_0}.

The procedure for constructing the private coin emulation tree E_{P_0,V_0} is similar to the one described for the the main public coin emulation, provided that the degree of T_{P_0,V_0} is $poly(n)$. In the foregoing private coin emulation we only care about the height of E_{P_0,V_0}, 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.

The `Build_Tree` procedure is a recursive procedure that reads and updates a tree T, which is initially set to equal the protocol tree T_{P_0,V_0}, until obtaining the final emulation tree E_{P_0,V_0}. We start with a protocol tree T_{P_0,V_0} rooted at u whose weight is $w(u) = 2^{r(n)}$, and on each step we modify this tree towards creating E_{P_0,V_0}. We define a **heavy descendant** of u in T to be a node in T whose weight is at least $\frac{w(u)}{r(n)}$, such that this descendant is either a leaf, or the weight of each of its children is smaller than $\frac{w(u)}{r(n)}$. Note that there are at most $r(n)$ 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 intermediate tree after the stage that we identify the heavy descendants of u and raise them to be children of u. The `Build_Tree(u)` procedure begins 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.

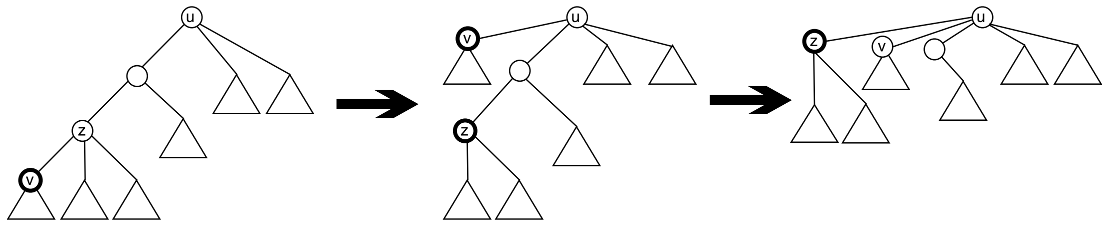

Figure 14: In the first step, v is identified as a heavy descendant of u and moved to be a heavy child of u. In the second step, z is identified and moved to be a heavy child of u. The triangles represent subtrees of the original tree.

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

It follows that the weight of a node in level $2r(n)/\log r(n)$ is at most $\frac{2^{r(n)}}{(r(n))^{r(n)/\log r(n)}} = 1$. 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 $O\left(r(n)/\log r(n)\right)$. We note that the number of heavy children of each node is at most $r(n)$, whereas the number of non-heavy children may be exponential in n.

A.3 Emulation protocol

A.3.1 Protocol preliminaries

Next, we describe the strategy of the designated prover P and verifier V in the new protocol ("emulation"). 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 V_0 strategy. 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 a possible 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 should provide 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 performs 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 A.4 (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 **seen-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.

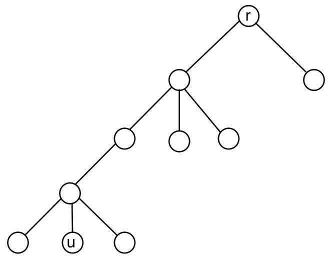

Figure 15: The nodes in the seen-list, S, in the iteration that the input node is u.

We want to claim that all the transcripts in the emulation tree of an 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 (as done with the heavy children in the designated construction).

A.3.2 Protocol construction

We stress that we give an honest prover centric description of the emulation protocol. This means that Step 1 describes the intended behaviour from an honest prover to provide the transcripts of the heavy children. A dishonest prover may deviate from this description, and the purpose of the verifier's checks to see that even if there was a deviation the prover can not succeed in claiming that a no-instance is a yes-instance with high probability.

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 seen-list S 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 A.5. (the ith iteration) On input a node u and seen-list 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 performs 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 $\gamma(v_i) \neq \gamma(v_j)$.

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 $V_0(x, \rho, \beta_1, \ldots, \beta_m) = 1$.

A.3.3 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. The number of iterations is equal to the height of the emulation tree, which is equal to $O\left(r(n)/\log r(n)\right)$.

A.4 Proof of correctness

A.4.1 Completeness

We claim that if x is a yes-instance, then the new verifier V accepts with probability at least $c(n)$, the completeness parameter of the original interactive proof system. The proof of completeness is partitioned into three parts. First, we shall show that the strategy of honest prover P, as specified in Subsection A.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 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 prover, 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. We know that 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, otherwise V would have chosen this continuation and not have reached u. Hence, $\gamma(u)\alpha\beta$ 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 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 tree as part of the `Build_Tree` procedure. We denote by $T^{v}_{P_0,V_0}$ the tree during construction, after v is identified as a heavy child and placed as a child of z in the tree. We can list the intermediate trees according to the order the nodes are identified starting from the root r and until the protocol tree is transformed to an emulation tree: $L = T_{P_0,V_0} = T^{r}_{P_0,V_0}, T^{u_1}_{P_0,V_0}, \ldots, T^{u_m}_{P_0,V_0} = E_{P_0,V_0}$. For every tree in this list, we show that for every node u in the tree the validations when u is the input node of the iteration pass. Therefore, it will follow that the validations also pass for every node in the final emulation tree E_{P_0,V_0}.

Let T be an intermediate tree, and u a node in T. When we say that a validation passes relative to T and u, 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 seen-list S consists of the ancestors of u and their children in T. Recall that the emulation protocol does not use T as an emulation tree, since it is an intermediate tree so its height has not been reduced enough to satisfy the bound required by the theorem. However, if T had been used the validations would have passed.

Let $T^{z}_{P_0,V_0}$ be an intermediate tree from the list above, created when identifying the node z as a heavy child of some other node. We assume that the validation we are currently checking holds for every node in $T^{z}_{P_0,V_0}$, and show that it also holds in the next step of the construction, that is the next tree in the list L. Recall that the next step is identifying a heavy child for z (or if z doesn't have heavy children then identifying the next heavy child for the next node in the traversal of the intermediate tree $T^{z}_{P_0,V_0}$). Denote the next heavy child being identified by v, and the intermediate tree after this modification by $T^{v}_{P_0,V_0}$.

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 $T^{v}_{P_0,V_0}$, as well as nodes whose seen-list changes. Exactly which nodes are affected depends on the validation.

Remark A.6. Let v be a node in $T^{z}_{P_0,V_0}$ that the `Build_Tree` procedure has not been invoked on yet. Recall that the children of v in $T^{z}_{P_0,V_0}$ 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 $T^{z}_{P_0,V_0}$ 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 $T^{z}_{P_0,V_0}$. Thus, the subtree of $T^{z}_{P_0,V_0}$ rooted at v, denoted by $T^{z}_{P_0,V_0}(v)$, is a subtree of T_{P_0,V_0}.

Now, we go over the validations, 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 A.5. Assuming that the validations hold for every node in $T^{z}_{P_0,V_0}$, we shall prove that these validations hold for every node in $T^{v}_{P_0,V_0}$ (the next intermediate tree in the list).

(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 $T^{z}_{P_0,V_0}$ 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 $T^v_{P_0,V_0}$ that has a child it did not have in $T^z_{P_0,V_0}$ 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 $T^z_{P_0,V_0}$ 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 $T^v_{P_0,V_0}$, 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 transcripts 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 together with validation (a) 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.

Definition A.7 (Seen-list of a node)**.** When we consider an intermediate tree $T^z_{P_0,V_0}$, which may not be the final emulation tree, and some node $v \in T$, then we denote by $S(v)$ the seen-list S in the beginning of the iteration where v is the input node handled. That is, $S(v)$ is the list containing the nodes that are ancestors of v in $T^z_{P_0,V_0}$, 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 intermediate tree $T^z_{P_0,V_0}$ before identifying and moving the child v of z, and we show that it holds in the tree $T^v_{P_0,V_0}$ after the identification of v as a heavy child of z.

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 $T^v_{P_0,V_0}$, it is enough to consider the nodes $u \in T^v_{P_0,V_0}$ that have new descendants or new nodes in $S(u)$ relative to the ones they had in $T^z_{P_0,V_0}$. The descendants of every node in $T^v_{P_0,V_0}$ are all descendants of it in $T^z_{P_0,V_0}$.

The only nodes in $T^v_{P_0,V_0}$ that have new nodes in their seen-list are the descendants of z, since now $\gamma(v)$ is in their seen-list (recall definition in Remark A.6), whereas $\gamma(v)$ may not have been in their seen-list before the move. By Remark A.6, 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 $T^v_{P_0,V_0}$. Since determining the heavy descendants of z is done bottom up (this is implied by the definition of heavy descendants in Subsection A.2), if $\gamma(u)$ is a prefix of $\gamma(v)$ then u is an ancestor of v in $T^z_{P_0,V_0}$. It is left to check that $\gamma(v)$ is not a prefix of the transcripts of the descendants of u in $T^v_{P_0,V_0}$. By Remark A.6, it follows that the subtree of $T^v_{P_0,V_0}$ rooted at u, denoted by $T^v_{P_0,V_0}(u)$, is a subtree of T_{P_0,V_0}. Thus, because $v \notin T^v_{P_0,V_0}(u)$ (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 $T^v_{P_0,V_0}(u)$, which are the descendants of u in $T^v_{P_0,V_0}$. (See Figure 16.)

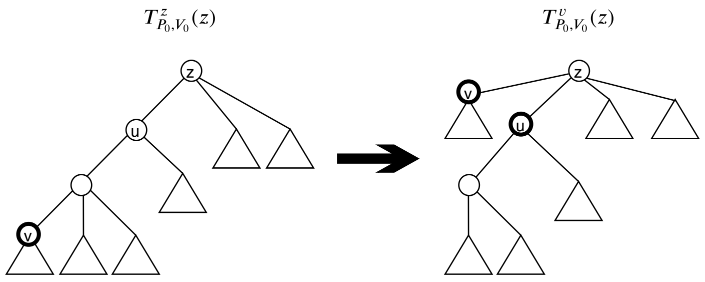

Figure 16: v is identified as a heavy child of z, and u is a descendant of z, which was an ancestor of v in $T^z_{P_0,V_0}$.

(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 A.8 (Relating leaves in the two trees). *Let $\widetilde{P}$ be a prover for the emulation of input x with emulation tree $E_{\widetilde{P}}$, using a strategy 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 $\gamma(u) = \alpha_1\beta_1, \ldots, \alpha_j\beta_j$, the strategy of $\widetilde{P_0}$ satisfies $\widetilde{P_0}(x, \alpha_1, \ldots, \alpha_i) = \beta_i$ for every $i \le j$. 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 A.8, 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 $\gamma(u) = \alpha_1\beta_1, \ldots, \alpha_j\beta_j$ it holds that $P_0(x, \alpha_1, \ldots, \alpha_i) = \beta_i$ for every $i \le j$. Applying Claim A.8, 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 $\widetilde{P}_0$. □

A.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 A.9, which is a slightly simplified version of Lemma 6.5 in Section 6.2.1. The other two components of the soundness analysis of the public coin system (which are defining the notion of actual weight of a node, and bounding the gap between the claimed and actual weight) are not required for the private coin soundness analysis.

The crux of the proof is extracting a deterministic strategy for the original prover $\widetilde{P}_0$ 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 acceptance probability is at least as high. Hence, if we show soundness holds with deterministic provers it implies soundness is maintained also with non-deterministic provers. Because the prover is deterministic we know that there is an emulation tree underlying the strategy of prover $\widetilde{P}$. In an emulation tree every node contains a transcript that extends the transcript of its parent. Hence, the protocol tree of $\widetilde{P}_0$ and V can be used to define an emulation tree, which we denote 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 $i \leq j$. 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. (The prover can compute the transcripts of the children such that they are different continuations of their parent and prover consistent with the list S instead of every node where the verifier would have aborted in the intermediate checks.) In Lemma A.9 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 A.8 (provided in Subsection A.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 A.4).

For a node $u \in E_{\widetilde{P}}$ provided during the emulation, denote by $S(u)$ the seen-list 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 A.9 (Prover consistency of the emulation tree)**.** *If $\widetilde{P}$ is a prover for the new emulation protocol using a strategy such that the verifier V does not abort until the final checks (for every verifier randomness), 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 6.5 in Section 6.

Proof. Let u and v be 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-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 seen-list of 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.

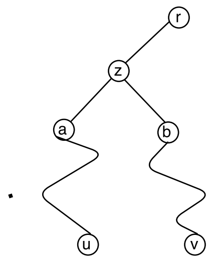

Figure 17: The subtree of $E_{\widetilde{P}}$ that contains u and v

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 children. 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.

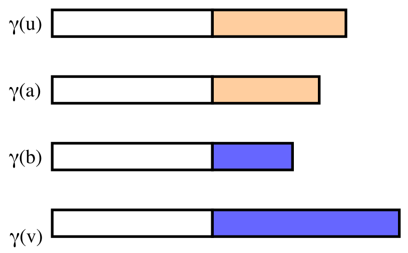

Figure 18: $\gamma(a)$ and $\gamma(b)$ agree on the prefix in white, while $\gamma(a)$ is a prefix of $\gamma(u)$ and $\gamma(b)$ is a prefix of $\gamma(v)$.

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 by contradiction that $\gamma(b)$ is a prefix of $\gamma(u)$. Note that $b \in S(u)$, so by validation 2c (which passed for the parent of u) $\gamma(b) \neq \gamma(u)$, 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 $a = a_0, a_1, \ldots, a_n = u$ 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 proper 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.

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 $\gamma(b)$,

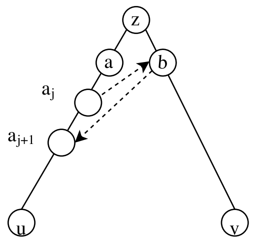

Figure 19: The dashed arrow pointing from b to a_{j+1} represent the fact that $\gamma(b)$ is a prefix of $\gamma(a_{j+1})$, and similarly that $\gamma(a_j)$ is a prefix of $\gamma(b)$.

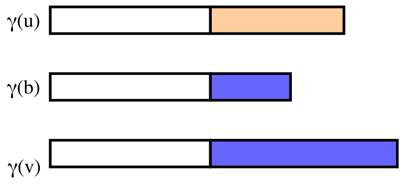

Figure 20: The maximal common prefix between $\gamma(u)$ and $\gamma(b)$ appears in white. Since $\gamma(b)$ is a prefix of $\gamma(v)$ the maximal common prefix of $\gamma(u)$ and $\gamma(v)$ is the same as the former.

so it ends with a prover message (See Figure 20).

□

Appendix B Alternative public coin emulation

In this Appendix we sketch a proof for an additional weaker version of the main theorem, one which transforms a private coin interactive proof system to a public coin interactive proof system. This transformation does not reduce the round complexity, and it does not preserve the randomness complexity, but it also does not blow up the randomness complexity by much. Recall that this is indeed weaker than the main theorem, because this transformation does not yield a public coin interactive proof where the round complexity is bounded by the randomness complexity. Moreover, the randomness complexity is not preserved.

The contribution of this theorem is also due to the fact that we can derive the main Theorem 1.1 from Theorem B.1 and Theorem A.1. In this way, we obtain a modular proof of the main theorem in which the creation of the protocol tree and its emulation are separated into two stages. We would like to thank an anonymous reviewer for suggesting the idea of proving theorem Theorem B.1.

Theorem B.1 (Randomness Efficient Public Coin Emulation)**.** *Suppose that S has an interactive proof system of randomness complexity $r(n)$ and round complexity $m(n)$ for instances of length n. Then, S has a* public coin *interactive proof system of round complexity $O(m'(n))$, where $m'(n)$ =max$\{r(n)/\log r(n), m(n)\}$, and randomness complexity $O(\log r(n) \cdot m')$. Furthermore, the resulting interactive proof system has perfect completeness.*

The section is organized as follows, first we show how to conclude the main theorem from the two weaker theorems. Next, we sketch a proof for Theorem B.1.

B.1 Concluding the main Theorem 1.1 using the alternative emulation

For clarity, we begin by restating the main Theorem.

Theorem 1.1 (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 $O(r(n)/\log r(n))$ and randomness complexity $O(r(n))$. Furthermore, the resulting interactive proof system has perfect completeness.

Next, we give an alternative proof of the main theorem. See the main sections of the paper for the standard proof.

Proof. Fix an interactive proof for the set S, that has randomness complexity $r(n)$ for instances of length n. From Theorem A.1, it follows that S has a private coin interactive proof system of round complexity $O(r(n)/\log r(n))$ and randomness complexity $r(n)$. Applying Theorem B.1 on the new interactive proof system for S, if follows that S has a public coin interactive proof system of round complexity $O(r(n)/\log r(n))$, and randomness complexity $O\big(\log r(n) \cdot \big(r(n)/\log r(n)\big)\big) = O(r(n))$. Furthermore, the resulting interactive proof system has perfect completeness. □

Reflection. In this alternative proof, we first use Theorem A.1 to reduce the number of rounds of the private coin interactive proof system to $O\left(r(n)/\log r(n)\right)$. We then apply Theorem B.1 on the new interactive proof system and convert it to a public coin interactive proof system that does not increase the round complexity to above $O\left(r(n)/\log r(n)\right)$, and has randomness complexity of $O(\log r(n))$ times the number of rounds, so randomness complexity is $O(r(n))$. In particular, using other known transformations of private coin to public coin interactive proofs, instead of Theorem B.1, would not have worked in deriving the main theorem since these transformations blow up the randomness complexity by too much.

The main difference between the standard proof in the main section of this paper and the alternative proof, is that in the standard proof the transformation is done in one shoot. That is, there is one transformation of the protocol tree of the original interactive proof system to an emulation tree that *both* has degree bounded polynomially and height bounded by $O\left(r(n)/\log r(n)\right)$. Then this emulation tree is used to perform a public coin emulation and the verifer has to perform on every round both validations concerning the degree reduction and validations concerning the height reduction. In contrast, in the alternative proof the transformation of the protocol tree is separated to two parts: first, reduce the height of the tree (which will reduce the number of rounds to $O\left(r(n)/\log r(n)\right)$), and then reduce the degree of the tree (which will enable emulating with public coins). The emulation itself is also separated into two parts. Since we are using the private coin transformation we derived with Theorem A.1, and then emulating it using the emulation of Theorem B.1, it follows that all the validations defined in the private coin emulation protocol are postponed to after the last iteration of public coin emulation Construction B.6 (see the protocol in the following sections).

B.2 Proof Sketch of Theorem B.1

We sketch the proof of Theorem B.1, beginning by giving an overview in Section B.2.1. Next, we describe the construction of the emulation tree and its properties in Section B.2.2 and Section B.2.3, and finally describe the emulation protocol in Section B.2.4 . We omit the analysis of the emulation protocol. We refer the interested reader to the analysis in Section 6, which includes the ideas required for the analysis of this theorem.

B.2.1 Overview

Fixing an interactive proof between a prover P and a verifier V, and an instance x of length n, we describe the possible prover-verifier interactions on common input x using the protocol tree $T_{P,V}$ defined in Section 3. The height of $T_{P,V}$ corresponds to the number of rounds $m(n)$ of interaction, and the number of leaves is $2^{r(n)}$, where $r(n)$ is the randomness complexity. As in the proof of the main theorem, our goal is to transform the protocol tree to an emulation tree, denoted by E_{P_0,V_0}, that defines a prover strategy for a public coin emulation. Recall that the way we use the emulation tree to define a prover strategy for the public coin emulation is by initiating the emulation at the tree root, and on each round having the prover assist the verifier at progressing one step down the emulation tree. (This assistance is required because the verifier does not have access to E_{P_0,V_0}.) Each round consists of the prover providing the verifier with the descriptions of the children of the current node u that was sampled on the previous round, where these descriptions contain the weights of the various children. Hence, in order to apply this emulation strategy, the prover should use an emulation tree with a degree that is bounded by $\mathrm{poly}(n)$, whereas the degree of $T_{P,V}$ can be exponential in n. Thus, our goal is to transform the protocol tree $T_{P,V}$ to an emulation tree E_{P_0,V_0}, with a degree bounded by $\mathrm{poly}(n)$, and the height of the tree (which corresponds to the number of rounds) should be $O(m')$, for $m'(n) = \max\{r(n)/\log r(n), m(n)\}$. This is done using the `Build_Tree(r)` procedure.

In order to reduce the degree when it is too large, we group the children of r under new children, which we call **interval children**. (Note that here we don't need to first lift the heavy descendants of r as we did in the proofs of the other theorems, because we are not trying to reduce the number of rounds, unlike in the other theorems.) In general, children of r in T_{P_0,V_0} may become non-immediate descendants during the procedure.

The `Build_Tree(r)` procedure unites the children of r into groups. We unite the children of r by lexicographic order of their transcript field, such that the weight of each group is larger than $w(r)/r(n)$ and at most $2w(r)/r(n)$ (except for, possibly, the last group which is only required to have weight smaller than $2w(r)/r(n)$). We create a new **interval child** v for each such group, where the children of v are the nodes in the group. If r has only one child u, then instead of creating an interval child for r, we leave u as an original child of r.

Denote by $T^u_{P_0,V_0}$ the intermediate tree after we identify and set u as an interval or original child of r. This way we have a notation for every transformation of the tree in which we create an additional child.

After creating all the interval children of r (or identifying the original child), the children of r in the final emulation tree E_{P_0,V_0} are exactly the children of r in $T^z_{P_0,V_0}$, where z is the last interval child of r created. The number of children r has is at most $r(n)$, since the weight of each interval child of r (except for possibly the last interval child) is at least $w(r)/r(n)$. Next, the procedure is called recursively on the children of r in $T^{z}_{P_0,V_0}$ in order to create the final emulation tree.

The description of a node u in the emulation tree is composed of the **transcript** field $\gamma(u) = \alpha_1\beta_1 \ldots \alpha_i\beta_i$ 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. 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 intermediate 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. Original 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 descendants that are grouped under this node.

We show in the analysis that the height of E_{P_0,V_0} is $O(m'(n))$, where $m'(n) = \max\{r(n)/\log r(n), m(n)\}$. The degree of nodes in E_{P_0,V_0} is at most $r(n)$, hence it is suitable for public coin emulation.

(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 prover is computationally unbounded. Therefore the running time is not an issue.)

B.2.2 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 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. If u has only one child v with non zero weight, then the procedure deletes the children of u whose weights are equal to zero, identifies v as an **original child** of u, and sets the range of v to be full range $R(r) = [0^{\ell}, 1^{\ell}]$. Otherwise, the `Build_Tree` procedure invokes the

`Build_Interval(u)` procedure, in order to create 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_Interval(u).` This procedure groups the 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 $Interval(s', e)$ the set of children of u in $T(u)$ whose 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 $Interval(s', e)$ can be a proper subset of the non-heavy children of u. We define the weight of the set $Interval(s', e)$, which we denote by $W(s', e)$, as the sum of the weights of nodes in $Interval(s', e)$.
   We set e', to be the minimal $e \in \{0,1\}^\ell$ that satisfies $W(s', e) \geq w(u)/r(n)$. 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 $Interval(s', e')$. Set u as a parent of v and let v be the parent of all nodes that are in $Interval(s', e')$.

Note that the weight of an interval child of u is at most $2w(u)/r(n)$ and at least $w(u)/r(n)$, except possibly for the last interval child, whose weight is at least 1.

B.2.3 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 original child of u and hence $\gamma(v) = \gamma(u)\alpha\beta$ for some $\alpha, \beta \in \{0,1\}^\ell$, 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. We omit the proofs for most of these properties and we refer the reader to the proofs in Section 4.3, which are a slightly more complex version of the corresponding claims here. We only include the proof of Corollary B.4 since it is different from the one in Section 4.3.

Claim B.2 (Node degree)**.** *Each node u in the final emulation tree E_{P_0,V_0} has at most $r(n)$ children.*

Claim B.3 (Weight reduction)**.** *For every node u in the final emulation tree E_{P_0,V_0}, and each grandchild z of u, either z is a child of an interval child of u, and the weight of z is at most $2w(u)/r(n)$, or z is an original child of u.*

Corollary B.4 (Corollary to Claim B.3)**.** *The height of the final emulation tree E_{P_0,V_0} is $O(m'(n))$, where $m'(n)$ =max$\{r(n)/\log r(n), m(n)\}$.*

Proof. Fixing an iteration that begins with input node u, the prover helps the verifier go one step down E_{P_0,V_0}. Hence, two iterations later the chosen node z is a grandchild of u, so from Claim B.3 either z is a child of an interval child of u and the weight of z is at most $2w(u)/r(n)$, or z is a child of an original child of u. Note that after visiting m original children while going down the tree we have reached a leaf of E_{P_0,V_0}. In addition, after visiting $3r(n)/\log r(n)$ interval children the weight of the node we reached is at most $2^{r(n)}/(r(n)/2)^{3(r(n)/\log r(n))} < 1$, so we have reached a leaf. It follows that after $2(m + 3r(n)/\log r(n))$ iterations we have reached a leaf (either by visiting m original children or having the weight reduced with interval children). The multiplication by two comes from the fact that we have a guarantee of visiting an original child or reducing the weight only after two iterations. □

Claim B.5 (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.

B.2.4 Public Coin Emulation

Next, we describe the strategy of the designated prover P and verifier V in the new protocol ("emulation"). 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 performs 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 part of the emulation tree seen so far. 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 exposition and definitions for this section are identical to the ones in Section 5, with the minor change of having original children in this emulation, instead of the heavy children in the other Section. Hence, we will not repeat the exposition here, and the reader is referred to read the exposition in Section 5, which includes transcript descendant Definition 5.1, transitivity Definition 5.2 and prover consistency Definition 5.3. Note that what is stated about heavy children in Section 5 holds for original children in this emulation (although the creation of the two types of children is different, the claims we make about them in this section are identical).

Construction B.6. (the ith iteration) On input a non-leaf node u.

1. The prover provides the descriptions of the children $v_1, \ldots, v_d$ of u:
$$(\gamma(v_1), R(v_1), w(v_1)), \ldots, (\gamma(v_d), R(v_d), w(v_d))$$

2. If $d = 1$, then u has a single child v_1, and the verifier performs the following validations and rejects if any of them fails:
   (a) The verifier checks that the weight of v_1 is equal to the weight of u.
   (b) The verifier checks that v_1 is a transcript descendant of u.

3. Otherwise, if $d > 1$ the verifier performs 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,
$$w(u) = \sum_{j=1}^{d} w(v_j) \tag{B.1}$$
   (c) For each $j \in [d]$, the verifier checks that v_j is a transcript descendant of u.
   (d) For each child v_j, the verifier checks that $\gamma(v_j) = \gamma(u)$.
   (e) The verifier checks that the ranges of all the children are disjoint; that is, for every two children v_j and v_k, the verifier checks that $R(v_j) \cap R(v_k) = \emptyset$.

4. 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 $O(\log r(n))$ coin tosses. That is, J is such that
$$\mathbf{Pr}[J = j] \leq \frac{w(v_j)}{\sum_{i=1}^{d} w(v_i)} \cdot (1 + 1/r(n)) \tag{B.1}$$
   We only utilize $O(\log r(n))$ public coins per iteration since we want this emulation to be as efficient as possible in terms of randomness complexity, without sacrificing soundness by more than a constant factor in total. Hence, we compromise on sampling each child with probability proportional to $w(v_j)$, and instead sample with approximate probability. See the explanation for approximate sampling in Appendix 5.4.

By our conventions, the last message the verifier V_0 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).

Acknowledgments.

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

References

[1] LÁSZLÓ BABAI: Trading group theory for randomness. In *Proc. 17th STOC*, pp. 421–429. ACM Press, 1985. [doi:10.1145/22145.22192] 4

[2] LÁSZLÓ BABAI AND SHLOMO MORAN: Arthur–Merlin games: A randomized proof system and a hierarchy of complexity classes. *J. Comput. System Sci.*, 36(2):254–276, 1988. [doi:10.1016/0022-0000(88)90028-1] 4

[3] MIHIR BELLARE, ODED GOLDREICH, AND SHAFI GOLDWASSER: Randomness in interactive proofs. *Comput. Complexity*, 3(4):319–354, 1993. [doi:10.1007/BF01275487] 4

[4] MARTIN FÜRER, ODED GOLDREICH, YISHAY MANSOUR, MICHAEL SIPSER, AND STATHIS ZACHOS: On completeness and soundness in interactive proof systems. In SILVIO MICALI, editor, *Randomness and Computation*, volume 5 of *Adv. Comput. Res.*, pp. 429–442. JAI Press, 1989. Author's website. 4

[5] ODED GOLDREICH AND MAYA LESHKOWITZ: On emulating interactive proofs with public coins. In ODED GOLDREICH, editor, *Computational Complexity and Property Testing: On the Interplay Between Randomness and Computation*, pp. 178–198. Springer, 2020. [doi:10.1007/978-3-030-43662-9_12, ECCC:TR16-066] 4, 9

[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. *J. ACM*, 38(3):690–728, 1991. Preliminary version in FOCS'86. [doi:10.1145/116825.116852] 5

[7] ODED GOLDREICH, SALIL P. VADHAN, AND AVI WIGDERSON: On interactive proofs with a laconic prover. *Comput. Complexity*, 11(1–2):1–53, 2002. [doi:10.1007/s00037-002-0169-0] 6

[8] SHAFI GOLDWASSER, SILVIO MICALI, AND CHARLES RACKOFF: The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. Preliminary version in STOC'85. [doi:10.1137/0218012] 4

[9] SHAFI GOLDWASSER AND MICHAEL SIPSER: Private coins versus public coins in interactive proof systems. In SILVIO MICALI, editor, *Randomness and Computation*, volume 5 of *Adv. Comput. Res.*, pp. 73–90. JAI Press, 1989. Preliminary version in STOC'86. 4, 5, 9

[10] ADAM R. KLIVANS AND DIETER VAN MELKEBEEK: Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM J. Comput.*, 31(5):1501–1526, 2002. [doi:10.1137/S0097539700389652] 4

[11] MAYA LESHKOWITZ: Round complexity versus randomness complexity in interactive proofs. In *Proc. 22nd Internat. Conf. on Randomization and Computation (RANDOM'18)*, pp. 49:1–16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. [doi:10.4230/LIPIcs.APPROX-RANDOM.2018.49] 1

[12] CARSTEN LUND, LANCE FORTNOW, HOWARD J. KARLOFF, AND NOAM NISAN: Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992. Preliminary version in FOCS'90. [doi:10.1145/146585.146605] 4, 10

[13] RONEN SHALTIEL AND CHRISTOPHER UMANS: Low-end uniform hardness versus randomness tradeoffs for AM. *SIAM J. Comput.*, 39(3):1006–1037, 2009. [doi:10.1137/070698348] 4

[14] ADI SHAMIR: IP = PSPACE. *J. ACM*, 39(4):869–877, 1992. Preliminary version in FOCS'90. [doi:10.1145/146585.146609] 4, 6, 10

AUTHOR

Maya Leshkowitz
Ph. D. student
The Hebrew University of Jerusalem
Jerusalem, Israel
mleshkowitz@gmail.com

ABOUT THE AUTHOR

MAYA LESHKOWITZ received her M. Sc. from Weizmann Institute of Science, under the supervision of Oded Goldreich.

Maya discovered her love for solving mathematical questions as a child when she played and solved riddles together with her grandfather, Shimon Even. In graduate school, although she was interested in Cognitive Science and human behavior, the Math and Computer Science courses at the Hebrew University sparked her old passion and led her to pursue a M. Sc. in Computer Science. Currently, she is a Ph. D student in the Cognitive Science department at the Hebrew University, under the supervision of Ran R. Hassin. She is interested in how people group information into chunks (such as by event or category) and how these chunks later influence attitudes and evaluations.

When she is not doing research, she enjoys drawing, dancing, and traveling.