cryptography

MDPI

Article

Models for Generation of Proof Forest in zk-SNARK Based Sidechains

Yuri Bespalov [1,*], Lyudmila Kovalchuk [2,3], Hanna Nelasa [4], Roman Oliynykov [2,5,*] and Rob Viglione [6]

1 Department of Mathematical Methods in Theoretical Physics, Bogolyubov Institute for Theoretical Physics, National Academy of Sciences of Ukraine, 03143 Kiev, Ukraine
2 Input Output, 61022 Kharkiv, Ukraine
3 G. E. Pukhov Institute for Modelling in Energy Engineering, National Academy of Sciences of Ukraine, 03164 Kyiv, Ukraine
4 Department of Information Security, National University "Zaporizhzhia Polytechnic", 69063 Zaporizhzhia, Ukraine; annanelasa@gmail.com
5 Department of Information Systems and Technologies Security, V. N. Karazin Kharkiv National University, 61022 Kharkiv, Ukraine
6 Horizen Labs, Austin, TX 78701, USA
* Correspondence: yu.n.bespalov@gmail.com (Y.B.); roman.oliynykov@iohk.io (R.O.)

Abstract: Sidechains are among the most promising scalability and extended functionality solutions for blockchains. Application of zero knowledge techniques (Latus, Mina) allows for reaching high level security and general throughput, though it brings new challenges on keeping decentralization where significant effort is required for robust computation of zk-proofs. We consider a simultaneous decentralized creation of various zk-proof trees that form proof-trees sequences in sidechains in the model that combines behavior of provers, both deterministic (mutually consistent) or stochastic (independent) and types of proof trees. We define the concept of efficiency of such process, introduce its quantity measure and recommend parameters for tree creation. In deterministic cases, the sequences of published trees are ultimately periodic and ensure the highest possible efficiency (no collisions in proof creation). In stochastic cases, we obtain a universal measure of prover efficiencies given by the explicit formula in one case or calculated by a simulation model in another case. The optimal number of allowed provers' positions for a step can be set for various sidechain parameters, such as number of provers, number of time steps within one block, etc. Benefits and restrictions for utilization of non-perfect binary proof trees are also explicitly presented.

Keywords: blockchain; sidechain; zk-SNARK; succinct blockchain; binary tree; perfect tree; free magma; operad; PRO; occupancy distribution

Citation: Bespalov, Y.; Kovalchuk, L.; Nelasa, H.; Oliynykov, R.; Viglione. R. Models for Generation of Proof Forest in zk-SNARK Based Sidechains. *Cryptography* **2023**, *7*, 14. https://doi.org/10.3390/cryptography7010014

Academic Editor: Kentaroh Toyoda

Received: 10 December 2022
Revised: 27 February 2023
Accepted: 1 March 2023
Published: 7 March 2023

1. Introduction

Sidechains (SCs) were first proposed in [1] and quickly became a universal and very comfortable tool in blockchain technology, while [1] proposed sidechains with a "classical" proof-of-work (PoW) protocol for usage in Bitcoin blockchain, the next article [2] builds secure proof-of-stake (PoS) sidechains, and articles [3,4] create a special type of sidechain consensus, the Latus protocol, which is secure for both PoW and PoS sidechains. They may be considered as an additional construction bound to blockchain (called mainchain (MC)) with the aim to provide some additional functionalities. SCs should be bound to an MC, as described in mentioned articles, to provide stronger security guarantees using blockchain properties, such as liveness and persistence [5]. Binding means that SCs also should send some information to the MC to guarantee the fairness of transformations in the SC [3].

In Latus, this information contains a series of recursive zk-SNARK-proofs [6,7] to establish decentralized and verifiable cross-chain transfers. Such an approach is similar to the ones proposed in [8] (Mina) and [9] (zkSync Era), but with additional features that allow its secure usage in sidechains. Latus introduces a special dispatching scheme that assigns

the generation of proofs randomly to interested parties, who then implement these tasks in parallel and submit generated proofs to the blockchain. An incentive scheme provides a reward for each accepted proof.

There are a variety of purposes to use SCs in different blockchains: for making smart-grid systems scalable and adaptable [10]; for secure data isolation in scalable consortium blockchain architecture based on world-state collaborative storage methods [11]; for secure parallel computation [12]; and for creating an exchange platform that allows the ability to trade tokens issued from different sidechains [13].

SCs may be based on various consensus protocols (such as PoW [14], PoS [15], etc.).

In this paper, we assume that SC uses the Latus protocol [4], which is a hybrid PoS based on Ouroboros Praos [16]. However, as SCs may be based on various consensuses, our main results may be applied for other consensus protocols (such as proof of work, proof of stake, etc.) with similar properties, allowing distributed proof generation and recursive SNARKs.

Following the modern trends in blockchain investigations, we use SNARKs to prove the correct functioning of SCs (detailed survey about different types of SNARKs used in blockchain can be found in [17]). These cause some modifications of the Merkle tree, which we call the "proof tree".

Distributed proof generation for block creation is the basic feature of Latus consensus. In brief, the procedure of block creation is as follows. A block forger—an entity that creates the block—shares a list of transactions to be included into the block. Then, other entities, called provers, construct SNARK-proofs for these transactions and also for each node of the corresponding proof trees built on these transactions.

Each prover, who creates a SNARK-proof, sets the prices for their proofs, according to the pricing policy of the current epoch. In the case of collisions, when there are several proofs for some node of the binary tree, the block forger chooses the cheapest one. Proofs that were included into the block are paid by the block forger from the corresponding transaction fees.

Our previous papers [18–20] considered only issues related to a single block creation. We used probabilistic and game-theoretic approaches to answer the questions on the optimal choice of SC parameters (e.g., a recommended number of transactions per block and a recommended value of incentives chosen). The model in which we obtain our results is very close to realistic with only one simplifying assumption: splitting of the block construction process into a fixed number of similar steps. Such assumption can be justified by the fact that the network synchronization time is essentially smaller than the proof generation time.

In this paper, unlike the previous ones, we consider algorithms related to simultaneous construction of trees by different block forgers. The sequence of such trees forms a linearly ordered forest. When the following block is published, the most left tree is included into it, and the remaining ones are regarded as a buffer for further block generation. The base assumptions used in this paper is almost the same as in the our previous papers, with the following modification: In practice, blockchain's sequences of transitions are proved using base SNARK for certifying single state transitions (in leaves) and merge SNARK for merging two proofs (in other vertices) ([21], section 4.1.1). We consider the scheme that replaces each base proof for a transaction by merged proofs for pair of leaves from the new additional level. So, we obtain a mathematically equivalent model where trees are one level higher and a pair of fresh leaves correspond to each transaction; therefore, we obtain more succinct algorithms.

The paper is organized as follows. The necessary basic preliminary results on binary trees are presented in Section 2. For us it is convenient to consider the sets of strict binary trees and of non-strict binary trees as specific realizations of the free magma $\mathcal{M} = \coprod_{n \geqslant 0} \mathcal{M}_0$ with a single generator. In Section 2.2, we introduce agreed monoid structures on infinite sequences of strict binary trees and positive numbers. The background of these construc-

tions is explained in Appendix A. In Section 2.3, we introduce the concept of perfect forest necessary for description of the main algorithm.

Section 3 presents the general scheme for 2×2 algorithms corresponding to the prover's behavior: deterministic (mutually consistent) or stochastic (independent), and shapes of generated trees: strict binary or perfect binary. It works with a potentially infinite sequence of trees t called a buffer. In the external for-loop, the first element is removed from the buffer and published as the following block. Block generation is divided into a fixed number of steps, each of which calls the OneStep() procedure. In all 2×2 cases each of suitable subsequent pairs of trees is merged into a single tree. Selection of such suitable pairs, deterministic or stochastic, is the unique difference among 2×2 cases (see Table 1). Its result can be encoded as a sequence Λ of trees with the height $\leqslant 1$. After this, the result of OneStep() procedure can be described as a function $f : t \mapsto \Lambda \circ t$, acting on the buffer t via the product defined in Section 2.2. Iteration of f over all steps and then a removal of the first tree from the buffer determines a function $F(t)$ describing transformation of the buffer during block generation. In Section 3.1, we consider a reduction of the above scheme, where only the number of leaves of each tree is taken into account. This is sufficient to study the throughput parameters of the blockchain(s). This reduction is possible due to the homomorphism of monoids described in Section 2.2.

In Section 4, we consider the case when behavior is deterministic. We prove that a sequence $(t, Ft, F^2t, \ldots)$ of buffer states takes a finite number of values and, hence, this sequence and the sequence of published trees are ultimately periodic. In Section 4.1, deterministic generation of strict binary trees is considered. The sequences of buffer states and published blocks are ultimately periodic with a period of length 1. This follows from the fact that transformation of the buffer after each block publishing is a weak contraction with respect to the Hamming distance. Under some natural assumption the height of the tree published over the period is the sum of (the integer part of) the binary logarithm of the number of provers and of the number of steps. An explicit formula for the fixed point is obtained in terms of the product on infinite sequences of binary trees. In Section 4.2, deterministic generation of perfect binary trees is considered. It is observed that the period of the sequence of published trees consists of perfect trees of two heights $h_{\max -1}$ and $h_{\max}$. The total efficiency is independent of shapes of generated trees and can be expressed by an explicit formula and admits natural asymptotics. Explicit results are obtained in the cases of a singe prover of a single step.

In Section 5, we consider the case when behavior is stochastic. In Section 5.1, it is shown that the placement of provers into suitable positions at each stage can be described by the classical occupancy distribution. In the deterministic case, the total number of generated proofs is the product of numbers of published blocks, of steps in a block and of provers. In the stochastic case, we define the total efficiency (respectively the block publishing efficiency) as the total number of generated proofs (respectively the number of proofs embodied in published blocks) divided by the above product. Thus, both efficiencies are numbers in the interval $[0, 1]$. They are convenient to compare the cases of various shapes and various values of integer parameters. The total efficiency is independent of shapes of generated trees and is expressed by an explicit formula and admits natural asymptotics. In Section 5.2, average values of the block publishing efficiency are calculated using a simulation model. We compare two cases when the shape of trees is binary strict or perfect. The maxima of these values as functions of numbers of positions are investigated. In Section 5.3 for generation of strict binary trees in a stochastic case we investigate an analog of the deterministic Formula (22) for heights of trees. Using the least squares method, we find the best approximation of the expected height of generated trees linear on the number of steps and logarithmic on the number of provers.

In Appendix A, we recall the concept of a non-symmetric operad and construction of the corresponding PRO. As the following step, one can obtain a monoid structure of infinite sequences of operations. The non-symmetric magma operad and the non-symmetric semigroup operad as its factor come from corresponding free objects with a single generator.

The corresponding monoids of infinite sequences of strict binary trees and its factor-monoid given by taking the number of leaves of each tree are considered in Section 2.2.

In Appendix B, we describe the software implementation of our algorithms and explain how they are used in the main text.

2. Preliminaries: Binary Trees and Forests

2.1. Binary Trees and Free Magmas

Trees, as a part of Graph Theory, are used in mathematics ([22], Chapter 5) and computer science, (see the classical volume [23]). In this subsection, we give some necessary definitions related to various types of binary rooted trees and forests, as there is no precise description convenient for our purposes and there might exist a confusion in terminology.

Rooted trees can be defined either (1) recursively, or as (2) partially ordered sets, or as (3) special types of graphs. All of these points of view are useful in their respective contexts.

A *tree* is defined as a connected acyclic undirected graph. Choosing a vertex in a tree as a root determines the natural direction for all edges (towards the root). However, usually it is not shown explicitly.

A *rooted tree* is a directed graph with a distinguished vertex called the *root* such that there exists a unique path from each vertex to the root. Thus, the set of vertices is equipped with a natural partial order: $v \leqslant w$ if there exists a path from v to w. In addition, the rooted tree itself becomes the Hasse diagram for this partial order (being drawn root up). If $v \to w$ is an edge in a directed rooted tree, we say that v is a *child* of w, and w is a *parent* of v. A vertex that has no child is called a *leaf*. Let us denote as $\operatorname{ht} t$ the *height* of the tree t, i.e., the maximal length of the path from a leaf to the root. Thus, leaves are minimal elements, and the root is the greatest element in the corresponding partially ordered set.

Each tree can be embedded in the plane. For a rooted tree all such embeddings are classified by linear orderings for the children of each vertex. Thus, if all such orderings are fixed, we say they are an *ordered rooted tree* (or a *plane rooted tree*).

Here, we mainly consider *strict binary plane rooted trees* (Instead of the term "strict binary tree", the terms "complete binary trees" and "full binary trees" are also used.), where "strict binary" means that every vertex is either a leaf or has exactly two children (left and right). In this case, for each non-leaf vertex, its left child vertex and its right child vertex are roots of maximal binary subtrees called, respectively, a left child tree and a right child tree.

Notation 1. *Let us denote as $n_v(t)$ the number of vertices, as $n_i(t)$ the number of internal vertices and as $n_\ell(t)$ the number of leaves in a rooted tree t.*

In a strict binary tree t these numbers satisfy the identities

$$n_v(t) = n_i(t) + n_\ell(t), \qquad n_\ell(t) = n_i(t) + 1. \tag{1}$$

A *magma* is a set equipped with a binary operation $\star$ without any additional properties Chapter 1 [23], ([24], 0.2). The free magma $\mathcal{M}$ with a single generator $*$ (up to isomorphism) is described recursively as

$$\mathcal{M} := \coprod_{n=1}^{\infty} \mathcal{M}_n$$

of recursively described subsets:

$$\mathcal{M}_1 := \{*\}, \qquad \mathcal{M}_n := \coprod_{m=1}^{n-1} \{(xy) \mid x \in \mathcal{M}_m \wedge y \in \mathcal{M}_{n-m}\} \quad \text{for } n > 1.$$

Thus, an element of $\mathcal{M}_{n+1}$ is a bracketing of a string of $n+1$ symbols $*$, i.e., the insertion of n left parentheses and n right parentheses defining the order of an n-fold application of the binary operation, e.g., five elements of $\mathcal{M}_4$ are presented in Figure 1a

(with the outer brackets removed). Cardinality of $\mathcal{M}_{n+1}$ is counted by Catalan number $C_n := \frac{1}{n+1}\binom{2n}{n}$. Various kinds of objects that are counted using Catalan numbers are called Catalan families. The Stanley's book [25] presents 214 Catalan families including various types of trees.

Strict binary trees are mentioned in Exercise 2.5 [25]. The generator of the free magma of strict binary trees is the zero height tree $\bullet$ whose single vertex is the root. For two binary trees t_1, t_2, the binary tree $t_1 \star t_2$ is a binary tree with a new root, whose left child tree is t_1 and the right child tree is t_2. Strict binary trees with four leaves corresponding to elements of $\mathcal{M}_4$ are presented in Figure 1c.

Non-strict binary trees (often in computer science literature just "binary trees") obtain yet another example of Catalan family Exercise 2.4 [25]. Thus, they can be defined recursively in the same way as strict binary trees, with the only difference that their free magma generator is the empty tree. A non-empty non-strict binary tree can be represented by a plane rooted tree, where each vertex has 0, 1 or 2 children, but with additional structure, every single child must be labeled "left" or "right". Such trees with three vertices corresponding to elements of $\mathcal{M}_4$ are presented in Figure 1b. Note that a non-strict binary tree is obtained from the corresponding strict binary by removing all its leaves.

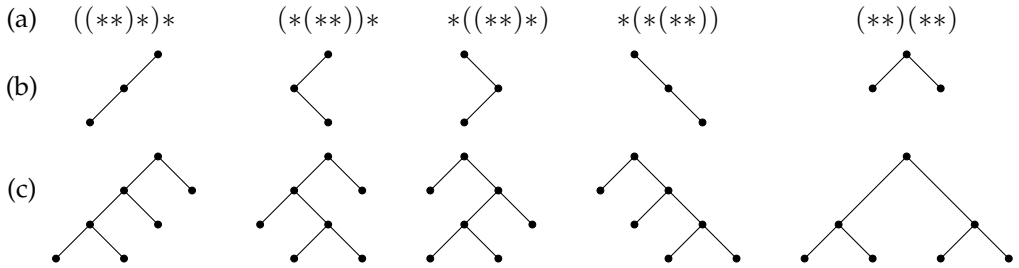

Figure 1. Images of the free magma: (**a**) bracketing, (**b**) non-strict binary tree, (**c**) strict binary tree.

2.2. Monoid of Sequences of Strict Binary Trees

Here, we consider composition $t \circ t'$ of infinite sequences of strict binary trees given by the gluing of subsequent leaves in t with subsequent roots in t', and composition $\ell \circ \ell'$ of infinite sequences of positive integers corresponding to the numbers of leaves in a tree.

First, for a strict binary tree t with n leaves and strict binary trees $t_1, \ldots, t_n$, we define the composition $t \circ (t_1, \ldots, t_n)$ obtained by gluing together the i-th leaf in t with the root of t_i for $i = 1, 2, \ldots n$.

This obtains the a non-symmetric operad structure on strict binary trees. Then one can apply the construction from Appendix A to obtain the monoidal structures described below. The details of the Appendix A are not necessary to understand the remainder of the material.

Let $\mathbb{T} = \mathbb{T}_{\text{strict}}$ be a set of sequences $(t_i)_{i \geqslant 0}$ of strict binary trees with only a finite number of non-zero height trees, equipped with a binary operation $(t, t') \mapsto t \circ t'$, where

$$(t \circ t')_i = t_i \circ \left(t'_{n_\ell(t_0)+\cdots+n_\ell(t_{i-1})}, t'_{n_\ell(t_0)+\cdots+n_\ell(t_{i-1})+1}, \ldots, t'_{n_\ell(t_0)+\cdots+n_\ell(t_i)-1}\right), \tag{2}$$

i.e., each t_i is successively applied in the sense of (A1) to the first not yet used $n_\ell(t_i)$ elements of t'.

Let $\mathbb{L} = \mathbb{L}_{\text{strict}}$ be a set of sequences $(\ell_i)_{i \geqslant 0}$ of positive integers with only a finite number of elements > 1, equipped with a binary operation $(\ell, \ell') \mapsto \ell \circ \ell'$, where

$$(\ell \circ \ell')_i = \ell'_{\ell_0+\cdots+\ell_{i-1}} + \ell'_{\ell_0+\cdots+\ell_{i-1}+1} + \cdots + \ell'_{\ell_0+\cdots+\ell_i-1}, \tag{3}$$

i.e., successively each $(t \circ t')$ is the sum of the first not yet used ℓ_i elements of ℓ'.

Corollary 1. *The above binary operations turn both* $\mathbb{T}$ *and* $\mathbb{L}$ *into monoids. The units are* $t^{(0)} = (\bullet)_{i\geqslant 0} \in \mathbb{T}$ *and* $\ell^{(0)} = (1)_{i\geqslant 0} \in \mathbb{L}$*. The map* $\mathbb{T} \to \mathbb{L}$, $\left(t_i\right)_{i\geqslant 0} \mapsto \left(n_\ell(t_i)\right)_{i\geqslant 0}$ *is a morphism of monoids.*

2.3. Perfect Binary Trees and Forests

A rooted tree is called *perfect* ("complete" by some authors) if all leaves are of the same distance to the root (that equals to the height h of the tree).

Notation 2. *For each non-negative integer* h*, the unique perfect binary tree of height* h *is denoted as* $t^\star_h$*.*

The perfect binary tree $t^\star_h$ has 2^h leaves; moreover, 2^d vertices are on the distance d from the root (they can be subsequently indexed with $0 \leqslant j < 2^d$, whose binary string representation has length d), and $2^{h+1} - 1$ vertices in total. See Figure 2.

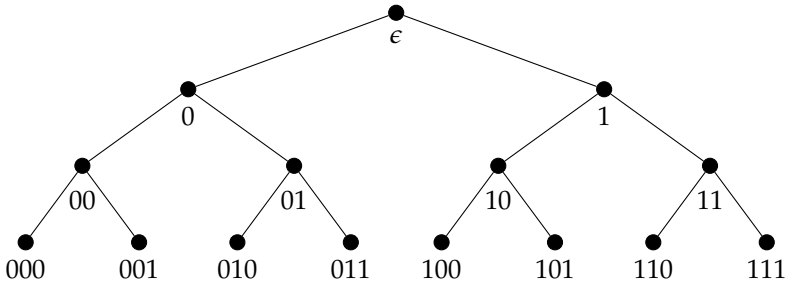

Figure 2. The perfect binary tree $t^\star_3$ with nodes labeled by binary strings

Note that a non-strict binary tree can be obtained from the corresponding strict binary tree by removing all its leaves.

All perfect binary trees are built recursively using magma square:

$$t^\star_0 = \bullet, \qquad t^\star_{h+1} = t^\star_h \star t^\star_h.$$

Perfect binary trees are most convenient to store a maximal number of leaves (that correspond to transactions).

A *forest* (respectively *plane forest*) f is a co-product (disconnected disjoint union) of a family of trees (respectively plane trees) $(t_i)_{i\in I}$. In both cases, the set of components I is unordered.

Let us recall that a subset $I \subseteq P$ in a poset P is called a *down-set* (respectively *up-set*) if, for each $x \in I$ and $y \in P$ with $y \leqslant x$ (respectively $y \geqslant x$), we have $y \in I$. Note that a subset $I \subseteq P$ is a down-set if its complement $P\backslash I$ is an up-set.

Lemma 1. *For a finite sequence* $f = (t_i)_{0\leqslant i<p}$ *of strict binary trees, the following conditions are equivalent:*

1. *There exists a bracketing, i.e., an element of the free magma* $\mathcal{M}_p$*, such that, after the corresponding application of* $p-1$ *magma operations* $\star$*, a perfect binary tree* $t^\star(f)$ *is obtained;*
2. $(t_i)_{0\leqslant i<p}$ *is a down-set of some perfect binary tree containing all its leaves with components ordered from left to right;*
3. *(a) Each tree* t_i *in the family is perfect,*
   (b) For each i, $n_\ell(t_i)$ *divides* $\sum_{j=0}^{j=i-1} n_\ell(j)$*;*
   (c) The total sum $\sum_{j=0}^{j=p-1} n_\ell(j)$ *is an power of* 2*.*

Proof. 1 $\Leftrightarrow$ 2: For a down-set in the perfect binary tree containing all its leaves, its complement is the non-strict binary tree corresponding to the appropriate bracketing.

$1 \Rightarrow 3$: (a) and (c) are obvious; to prove (b), let t' be a subtree of height h' in a perfect tree t of height h such that each leaf in t' is a leaf in t. Then, there exists a subforest in t that consists of $2^{h-h'}$ perfect trees of height h including t'.

$3 \Rightarrow 2$: Split the leaves of f into equal left and right halves. If there is a tree t_i in our forest whose leaves are in both halves, then, from 3(c), we conclude that the forest consists of this unique tree. Otherwise, we can represent $f = f_\ell \sqcup f_r$ as a co-product of two forests satisfying condition 3, so we suppose $t^\star(f) = t^\star(f_\ell) \star t^\star(f_r)$ and repeat the above choice for f_ℓ and f_r. □

Definition 1. *A finite sequence $(t_i)_{0 \leqslant i < p}$ of binary trees satisfying conditions of the above lemma is called a* perfect binary forest.

Figure 3 shows an example of a perfect binary forest and the corresponding perfect binary tree $t_4^\star = (t_2^\star \star (t_1^\star \star (t_0^\star \star t_0^\star))) \star (t_2^\star \star ((t_0^\star \star t_0^\star) \star t_1^\star))$.

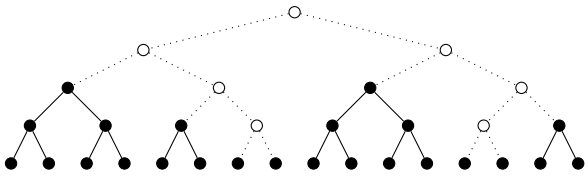

Figure 3. Example of a perfect binary forest.

In what follows, we use (potentially) infinite sequences of proof trees $(t_i)_{i \geqslant 0}$ with only a finite number of non-zero height trees as a buffer of proofs.

Definition 2. *An infinite sequences of binary trees $(t_i)_{i \geqslant 0}$ with only a finite number of non-zero height trees is called* perfect *if some its initial fragment $(t_i)_{0 \leqslant i < p}$, containing all non-zero height trees, is perfect.*

Proposition 1. *A sequence $t \in \mathbb{T}$ is perfect if there exists $t' \in \mathbb{T}$ such that $t' \circ t$ is a sequence that consists of a single perfect tree followed by zero-height trees.*

Definition 3. *A pair (t_i, t_{i+1}) of subsequent trees in a perfect sequence is called a* perfect, *if the new sequence with the pair (t_i, t_{i+1}) replaced with the product $t_i \star t_{i+1}$ remains perfect.*

Note that the binary operation between two strict binary trees can be written as $t \star t' = \wedge \circ (t, t')$, where $\wedge$ is the strict binary tree of the height 1. Thus, to replace (t_i, t_{i+1}) with the product $t_i \star t_{i+1}$ is the same as to take the composition $\wedge_i \circ t$ in the sense of (2), where $\wedge_i := (\underbrace{\bullet, \ldots, \bullet}_{i}, \wedge, \underbrace{\bullet, \ldots}_{\infty})$.

According to Lemma 1 a pair (t_i, t_{i+1}) is perfect if $n_\ell(t_i) = n_\ell(t_{i+1})$ and $2n_\ell(t_i)$ divides $\sum_{j=0}^{j=i-1} n_\ell(j)$.

Equivalently, suppose that the vertices of the whole perfect tree are labeled as in Figure 2. A pair (t_i, t_{i+1}) of neighboring trees in its down-set is perfect if their roots are labeled by strings '$w0$' and '$w1$' respectively for some $w \in \{0, 1\}^*$.

Obviously, two different perfect pairs cannot intersect. Thus, perfect pairs in a perfect sequence form an infinite sequence.

3. The General Scheme

A special feature of Algorithm 1 considered in this paper is a simultaneous construction of several trees forming a linear ordered forest, rather than a single tree. Periodically, the leftmost tree is included into the published block, and the remainder are considered as a buffer for further block generation.

Algorithm 1: The general scheme of blocks generation

Input: $b_{\text{behavior}}, b_{\text{shape}}, n_{\text{bl}}, n_{\text{st}}, n_{\text{pr}}, n_{\text{pos}}$;
Output: $t^{\copyright} = \left(t_i^{\copyright}\right)_{0 \leqslant i < n_{\text{bl}}}$;
Initialization: $t \leftarrow (\bullet)_{i>0}$; $t^{\copyright} \leftarrow ()$;
for $i \leftarrow 1$ **to** n_{bl} **do**
  for $j \leftarrow 1$ **to** n_{st} **do**
    OneStep$(b_{\text{behavior}}, b_{\text{shape}}, n_{\text{pr}}, n_{\text{pos}})$;
  Block publishing: remove t_0 from the buffer t and append t_0 to the list $t^{\copyright}$;

Input parameters are:

- Dichotomous b_{behavior} regulates behavior of provers, **deterministic** (mutually consistent) or **stochastic** (independent);
- Dichotomous b_{shape} describes the shape of generated trees, only **perfect** binary trees or arbitrary **strict** binary trees;
- n_{bl} is the number of blocks published during the epoch;
- n_{st} is the number of steps for one block generation;
- n_{pr} is the number of provers;
- n_{pos} is the number of positions allocated for proof.

During the work of the algorithm, a potentially infinite sequence of binary trees $t = (t_i)_{i \geqslant 0}$ arises that we call a *buffer*. It describes the state of the system and is used in the further block generation.

The output is the sequence $t^{\copyright} = (t_i^{\copyright})_{0 \leqslant i < n_{\text{bl}}}$ of binary trees included into published blocks (one tree per block).

The published list is initialized with the empty list $t^{\copyright} \leftarrow ()$. Theoretically, we assume that at the beginning $t = (\bullet)_{i \geqslant 0}$ is a potentially infinite sequence of zero height trees. In practice, we initialize t with an empty list and put zero-height trees there as needed.

Remark 1. *In a specific implementation, instead of two lists of binary trees, one can work with a single list, and an integer pointer p; trees with indexes $i < p$ are considered as published, and trees with indexes $i \geqslant p$ form a buffer. Block publishing in these terms is just an increment of the pointer $p \leftarrow p + 1$.*

The external for-loop of Algorithm 1 runs over all n_{bl} blocks in the epoch. The body of this loop consists of the internal for-loop that runs over n_{st} steps in the block and the block publishing directives. The body of the internal loop consists of Procedure OneStep$(b_{\text{behavior}}, b_{\text{shape}}, n_{\text{pr}}, n_{\text{pos}})$.

Procedure OneStep($b_{\text{behavior}}, b_{\text{shape}}, n_{\text{pr}}, n_{\text{pos}}$)

switch $b_{behavior}$ **do**
  case *deterministic* **do**
    switch b_{shape} **do**
      case *strict* **do**
        for $k \leftarrow n_{pr} - 1$ **to** 0 **by** -1 **do**
          in buffer t replace t_{2k}, t_{2k+1} by $t_{2k} \star t_{2k+1}$;
      case *perfect* **do**
        for $k \leftarrow n_{pr} - 1$ **to** 0 **by** -1 **do**
          in t replace the k-th perfect pair with their magma product;
  case *stochastic* **do**
    RandomPositions $\leftarrow \varnothing$;
    for $k \leftarrow 0$ **to** n_{pr} **do**
      put to RandomPositions a random integer from 0 to $n_{\text{pos}} - 1$;
    sort RandomPositions descending;
    switch b_{shape} **do**
      case *strict* **do**
        foreach k *in* RandomPositions **do**
          in buffer t replace t_{2k}, t_{2k+1} by $t_{2k} \star t_{2k+1}$;
      case *perfect* **do**
        foreach k *in* RandomPositions **do**
          in t replace the k-th perfect pair with their magma product;

This procedure presents the general scheme for 2×2 different cases corresponding to combinations of two dichotomous parameters b_{behavior} and b_{shape}. It acts on the buffer: some subsequent pairs of trees (t_k, t_{k+1}) are replaced with their magma product $t_k \star t_{k+1}$, and these pairs essentially depend on dichotomous parameters b_{behavior} and b_{shape}.

In the case "b_{behavior} is deterministic", the number of such pairs coincides with the number of provers n_{pr}; then, for the case that "b_{shape} is strict", these pairs are first n_{pr} subsequent pairs of the trees in the buffer: $\{(t_0, t_1), (t_2, t_3), \dots (t_{2n_{\text{pr}}-2}, t_{2n_{\text{pr}}-1})\}$; for the case that b_{shape} is perfect, we use first n_{pr} perfect pairs in the buffer. In the case "b_{behavior} is stochastic", first n_{pos} subsequent/perfect positions are reserved for the random choice. Each prover independently with equal probability selects one of these positions. For positions selected by at least one prover, the corresponding pair of trees is replaced with its magma product.

For fixed input parameters b_{behavior}, b_{shape}, n_{st}, n_{pr}, n_{pos}, the following functions (deterministic or random) together with their domains and codomains are defined. Let us denote

- $T = T_{b_{\text{shape}}}$ the set of b_{shape} (i.e., strict/perfect) binary trees;
- $\mathbb{T} = \mathbb{T}_{b_{\text{shape}}}$ the set of sequences $t = (t_i)_{i \geqslant 0}$ of b_{shape} binary trees with finitely many non-zero height trees;
- $t^{[]}$ be the sequence obtained by the shift, i.e., $t_i^{[]} = t_{i+1}$ for $i \geqslant 0$.

$$f = f_{b_{\text{behavior}}, b_{\text{shape}}, n_{\text{pr}}, n_{\text{pos}}} : \mathbb{T} \to \mathbb{T}, \qquad t \mapsto \Lambda \circ t, \tag{4}$$

$$F = F_{b_{\text{behavior}}, b_{\text{shape}}, n_{\text{st}}, n_{\text{pr}}, n_{\text{pos}}} : \mathbb{T} \to \mathbb{T}, \qquad t \mapsto \left(f^{n_{\text{st}}}(t)\right)^{[]}, \tag{5}$$

$$\pi = \pi_{b_{\text{behavior}}, b_{\text{shape}}, n_{\text{st}}, n_{\text{pr}}, n_{\text{pos}}} : \mathbb{T} \to T, \qquad t \mapsto \left(f^{n_{\text{st}}}(t)\right)_0. \tag{6}$$

Here, the function (4) is determined by Procedure OneStep. It can be described as a product (2) with the sequence $\Lambda = \Lambda_{b_{\text{behavior}}, b_{\text{shape}}, n_{\text{pr}}, n_{\text{pos}}} \in \mathbb{T}$ of trees of height $\leqslant 1$. The

internal for-loop of Algorithm 1 means the n_{st}-th iteration of this function. Then, block publishing determines the map (6), and the state of the buffer after each block publishing is changed according to the map (5).

Table 1 shows the whole scheme. This four cases are also implemented in software described in Appendix B.

Table 1. The scheme of 2×2 algorithms.

	Strict binary trees: merged positions are first pairs of subsequent trees	**Perfect binary trees:** merged positions are first so-called perfect pairs
Deterministic prover behavior: provers act mutually consistent, taking the first possible n_{pr} merge positions	b_{behavior} is deterministic b_{shape} is strict	b_{behavior} is deterministic b_{shape} is perfect
Stochastic prover behavior: provers act independently, with each selecting one of n_{pos} merge positions	b_{behavior} is stochastic b_{shape} is strict	b_{behavior} is stochastic b_{shape} is perfect

3.1. The Algorithm in Terms of Numbers of Leaves

Let us recall that a perfect tree is completely determined by its height, but there are C_n strict binary trees with $n+1$ leaves. However, the exact shape of trees is not important for the blockchain('s) throughput parameters' study. Thus, we can collect information about the number of leaves (and, therefore, (1) about the number of vertices) and optionally about heights of trees. Thus, instead of the sequence of trees $t = (t_i)_{i \geqslant 0}$, one can consider the sequence of integers $\ell = (\ell_i)_{i \geqslant 0}$, where $\ell_i = n_\ell(t_i)$ is the number of leaves of the corresponding tree. The initial state is described as $(\ell_i = 1)_{i \geqslant 0}$. The magma operation $\star$ on strict trees corresponds to addition of the numbers of leaves: $n_\ell(t_i \star t_{i+1}) = n_\ell(t_i) + n_\ell(t_{i+1})$.

A pair of numbers (ℓ_j, ℓ_{j+1}) in a perfect sequence $(\ell_i)_{i \geqslant 0}$ is perfect (i.e., corresponds to the perfect pair of trees) if

$$\ell_j = \ell_{j+1} \quad \text{and} \quad 2\ell_j \ \text{ divides } \ \sum_{i=0}^{j-1} \ell_i. \tag{7}$$

If, moreover, the sequence $(\ell_i)_{i \geqslant 0}$ is non-increasing, the condition (7) takes the equivalent form

$$\ell_j = \ell_{j+1} \quad \text{and} \quad \#\{i \,|\, 0 \leqslant i < j \wedge \ell_i = \ell_j\} \quad \text{is even.} \tag{8}$$

Let us denote by $\mathbb{L} = \mathbb{L}_{b_{\mathrm{shape}}}$ a set of sequences $(\ell_i)_{i \geqslant 0}$ of positive integers with only a finite number of elements > 1, in the case that b_{shape} is perfect. For a version of Algorithm 1 in terms of numbers of leaves, one considers the functions corresponding to (4)–(6). We keep the same names for the functions. Different (co)domains avoid ambiguity:

$$f : \mathbb{L} \to \mathbb{L}, \qquad \ell \mapsto \lambda \circ \ell, \tag{9}$$

$$F : \mathbb{L} \to \mathbb{L}, \qquad \ell \mapsto \left(f^{n_{\mathrm{st}}}(\ell)\right)^{[]}, \tag{10}$$

$$\pi : \mathbb{L} \to \mathbb{Z}_{>0}, \qquad t \mapsto \left(f^{n_{\mathrm{st}}}(\ell)\right)_0, \tag{11}$$

where in (9), $\lambda = \lambda_{b_{\mathrm{behavior}}, b_{\mathrm{shape}}, n_{\mathrm{pr}}, n_{\mathrm{pos}}} \in \mathbb{L}$ is the sequence of 1 or 2.

Functionality of the transition from trees to their numbers of leaves is expressed by the commutative squares

$$\begin{array}{ccc} \mathbb{T} & \xrightarrow{f} & \mathbb{T} \\ \downarrow{\scriptstyle n_\ell} & & \downarrow{\scriptstyle n_\ell} \\ \mathbb{L} & \xrightarrow{f} & \mathbb{L} \end{array} \qquad \begin{array}{ccc} \mathbb{T} & \xrightarrow{F} & \mathbb{T} \\ \downarrow{\scriptstyle n_\ell} & & \downarrow{\scriptstyle n_\ell} \\ \mathbb{L} & \xrightarrow{F} & \mathbb{L} \end{array} \qquad \begin{array}{ccc} \mathbb{T} & \xrightarrow{\pi} & T \\ \downarrow{\scriptstyle n_\ell} & & \downarrow{\scriptstyle n_\ell} \\ \mathbb{L} & \xrightarrow{\pi} & \mathbb{Z}_{>0} \end{array}$$

Here, $n_\ell : \mathbb{T} \to \mathbb{L}$ means $(t_i)_{i\geqslant 0} \mapsto \big(n_\ell(t_i)\big)_{i\geqslant 0}$.

Let us define the sequences of sequences (of elements of $\mathbb{L}$) as

$$\ell^{(n,k)} := f^k\big(F^n\big((1)_{i\geqslant 0}\big)\big), \qquad n \geqslant 0, \qquad 0 \leqslant k \leqslant n_{\mathrm{pr}}, \tag{12}$$

$$\ell^{(n)} := \ell^{(n,0)} = F^n\big((1)_{i\geqslant 0}\big), \qquad n \geqslant 0. \tag{13}$$

Elements of the double sequence $\big(\ell^{(n,k)}\big)$ are naturally linearly ordered, lexicographically on the pairs (n,k). In this sequence,

$$\ell^{(n,k+1)} = f\big(\ell^{(n,k)}\big), \qquad \ell^{(n+1,0)} = \ell^{(n,n_{\mathrm{st}})\,[\,]}.$$

The *length* of the sequence $\ell = (\ell_i)_{i\geqslant 0} \in L$ is defined as follows:

$$\operatorname{Len}\ell := \min_j\{j \geqslant 0 \mid \forall i \geqslant j : \ell_i = 1\}. \tag{14}$$

4. Deterministic Case

In this section, we consider only the case of Algorithm 1 when b_{behavior} is deterministic. We consider sequences $\ell^{(n,k)}$ for an arbitrary n that corresponds to the limiting case $n_{\mathrm{bl}} \to \infty$ of an infinite outer for-loop. One can also consider the infinite sequence $\hat{\ell}$ of the number of leaves of published trees with elements

$$\ell^{\copyright}_n = \pi\big(\ell^{(n)}\big), \qquad n \geqslant 0.$$

Lemma 2. *In the case, $b_{behavior}$ is deterministic, and the function $f : \mathbb{L} \to \mathbb{L}$ preserves a property of a sequence to be non-increasing.*

Proof. In the case b_{shape} is perfect, take into account (8). □

Corollary 2. *In the case that $b_{behavior}$ is deterministic, the sequences $\ell^{(n,k)}$ from (12) are non-increasing for all $n \geqslant 0$ and $0 \leqslant k \leqslant n_{pr}$.*

Let us denote as $\mathbb{L}^{\searrow}$ a set of non-increasing sequences from $\mathbb{L}$.

For a non-increasing sequence $\ell \in \mathbb{L}$ (e.g., for all $\ell^{(n,k)}$ when b_{behavior} is deterministic), the Formula (14) can be simplified:

$$\operatorname{Len}\ell = \min_j\{j \geqslant 0 \mid \ell_j = 1\}.$$

For $\ell \in \mathbb{L}$, the sum

$$N_i(\ell) := \sum_j (\ell_j - 1) \tag{15}$$

has only finite non-zero summands and, according to (1), can be interpreted as the total numbers of internal vertices. Independently on b_{shape}, for $\ell \in \mathbb{L}$, we have the identities

$$N_i(f(\ell)) = N_i(\ell) + n_{\mathrm{pr}}.$$

$$\pi(\ell) - 1 + N_i(F(\ell)) = N_i(\ell) + n_{\mathrm{st}} n_{\mathrm{pr}}. \tag{16}$$

Lemma 3. *In the case "$b_{behavior}$ is deterministic", for fixed n_{st}, n_{pr} the total number of internal vertices (and, hence, the length and all elements) of all $\ell^{(n,k)}$ are bounded together, i.e.,:*

$$\sup_{n,k} N_i(\ell^{(n,k)}) < \infty, \qquad \sup_{n,k} \operatorname{Len} \ell^{(n,k)} < \infty, \qquad \sup_{n,k,j} \ell_j^{(n,k)} < \infty.$$

Proof. In the case "b_{shape} is strict", $\operatorname{Len} \ell^{(n,k)} \leqslant n_{\mathrm{pr}}$ and $\operatorname{Len} \ell^{(n)} < n_{\mathrm{pr}}$. This follows from the observation that $\operatorname{Len} \ell \leqslant 2n_{\mathrm{pr}}$ implies $\operatorname{Len} f(\ell) \leqslant n_{\mathrm{pr}}$.

In the case that "b_{shape} is perfect", the number of the same non-zero height trees in the buffer is bounded by $n_{\mathrm{pr}} + 1$. See (27) for more accurate estimates.

Then, we can estimate from above the total number of internal vertices:

$$N_i(\ell^{(n)}) \leqslant C_{b_{\mathrm{shape}}}(\ell_0^{(n)}) = \begin{cases} (n_{\mathrm{pr}} - 1)(\ell_0^{(n)} - 1), & \text{if } b_{\mathrm{shape}} \text{ is strict,} \\ (n_{\mathrm{pr}} + 1)(2\ell_0^{(n)} - \log_2 \ell_0^{(n)} - 1), & \text{if } b_{\mathrm{shape}} \text{ is perfect.} \end{cases}$$

In the second case, we use the fact that all $\ell_j^{(n)}$ are powers of 2. For $\ell_0^{(n)} > 1$ and $n_{\mathrm{pr}} > 1$, the function $C_{b_{\mathrm{shape}}}$ is monotone increasing and, hence, invertible. Let $N_i(\ell^{(n)}) \geqslant C_{b_{\mathrm{shape}}}(n_{\mathrm{st}} n_{\mathrm{pr}} + 1)$. Then, $\pi(\ell^{(n)}) \geqslant \ell_0^{(n)} \geqslant n_{\mathrm{st}} n_{\mathrm{pr}} + 1$ and, by (16), we have $N_i(F(\ell)) \leqslant N_i(\ell)$. Thus, $\sup_{n,k} N_i(\ell^{(n,k)}) < C_{b_{\mathrm{shape}}}(n_{\mathrm{st}} n_{\mathrm{pr}} + 1) + n_{\mathrm{st}} n_{\mathrm{pr}}$. □

Definition 4. *The sequence $x = (x_i)_{i \in \mathbb{Z}_{\geqslant -1}}$ is called* ultimately periodic *if there exists $r \in \mathbb{Z}_{\geqslant 0}$ and $s \in \mathbb{Z}_{>0}$ such that, if $i > r$, then $x_i = x_{i+s}$. In this case, we write*

$$x = \underbrace{x_0, \ldots, x_r}_{\textit{preperiod}}, (\underbrace{x_{r+1}, \ldots, x_{r+s}}_{\textit{period}}).$$

The minimal preperiod *is the minimal r satisfying the above condition. The* minimal period *is the minimal s for minimal r satisfying the above condition.*

Given a function $g : X \to X$ and an element $x_0 \in X$, the sequence $x_0, g(x_0), g(g(x_0)), \ldots$ is ultimately periodic, whenever X is finite. Lemma 3 implies that the set $\{\ell^{(n)} \mid n \geqslant 0\}$ is finite.

Corollary 3. *In the case that "$b_{behavior}$ is deterministic", the sequences $\left(\ell^{(n)}\right)_{n \geqslant 0}$ of buffer states and $\ell^{\copyright}$ of published trees are ultimately periodic.*

Lemma 4. *If $(\ell^{(r+1)}, \ldots, \ell^{(r+s)})$ is a period of $\left(\ell^{(n)}\right)$, then*

$$\ell_{r+1}^{\copyright} + \cdots + \ell_{r+s}^{\copyright} - s = s n_{st} n_{pr}. \tag{17}$$

Proof. The identity (17) is the result of the summing of (16) over a period, i.e., for $\ell = \ell^{(r+1)}, \ldots, \ell^{(r+s)}$. □

4.1. Generation of Strict Binary Trees

In this subsection, we consider the case "b_{behavior} is deterministic" and "b_{shape} is strict". In this case, one-step functions (4) and (9) are

$$t \mapsto \Lambda \circ t, \qquad \Lambda = n_{\mathrm{pr}}\wedge = (\underbrace{\wedge, \ldots, \wedge}_{n_{\mathrm{pr}}}, \underbrace{\bullet, \ldots}_{\infty}),$$
$$\ell \mapsto \lambda \circ \ell, \qquad \lambda = n_{\mathrm{pr}} \cdot \mathbf{2} = (\underbrace{2, \ldots, 2}_{n_{\mathrm{pr}}}, \underbrace{1, \ldots}_{\infty}).$$

Lemma 5.

$$t^{(0,k)} = (n_{pr}\wedge)^{\circ k}, \qquad t^{(n,k)} = (n_{pr}\wedge)^{\circ k} \circ t^{(n)}. \tag{18}$$

$$\ell^{(0,k)} = (n_{pr} \cdot \mathbf{2})^{\circ k}, \qquad \ell^{(n,k)} = (n_{pr} \cdot \mathbf{2})^{\circ k} \circ \ell^{(n)}. \tag{19}$$

Let us consider the *Hamming distance* for $\ell, \ell' \in \mathbb{L}^{\searrow}$:

$$d_H(\ell, \ell') := \#\{i \mid \ell_i \neq \ell'_i\}.$$

Lemma 6. *In the case that "$b_{behavior}$ is deterministic and b_{shape} is strict", the transformation F is a weak contraction on $(\mathbb{L}^{\searrow}, d_H)$:*

$$d_H(F(\ell), F(\ell')) < d_H(\ell, \ell'), \qquad \ell \neq \ell' \in \mathbb{L}^{\searrow}.$$

Corollary 4. *The sequence of buffer states $\left(\ell^{(n)}\right)_{n \geqslant 0}$ is stabilized at the fixed point of F. Thus, this sequence and the sequence of published blocks are ultimately periodic with a period of length 1. The tree published in the period has $n_{pr}n_{st} + 1$ leaves.*

The following lemma shows that, if $n_{\mathrm{st}} \geqslant \log_2(n_{\mathrm{pr}} - 1)$, then the length of pre-period $\leqslant 1$.

Lemma 7. *If $n_{st} \geqslant \log_2(n_{pr} - 1)$, then*

$$t^{(n)} = \left((n_{pr}\wedge)^{\circ n_{st}}\right)^{[]}, \qquad t^{\copyright}_n = \left((n_{pr}\wedge)^{\circ n_{st}} \circ \left((n_{pr}\wedge)^{\circ n_{st}}\right)^{[]}\right)_0, \qquad n \geqslant 1. \tag{20}$$

If $n_{st} \geqslant \log_2(n_{pr})$, then

$$\left((n_{pr}\wedge)^{\circ n_{st}}\right)^{[]} = \left((n_{pr}\wedge)^{\circ \lceil \log_2 n_{pr} \rceil}\right)^{[]}. \tag{21}$$

Proof. In both cases, we use the observation: for $t, t' \in \mathbb{T}$, if $n_\ell(t_0) \geqslant \operatorname{Len} t'$, then $(t \circ t')^{[]} = t^{[]}$. □

Thus, if $n_{\mathrm{st}} \geqslant \log_2(n_{\mathrm{pr}})$, then the period of the sequence of published trees consists of a single tree $\left((n_{\mathrm{pr}}\wedge)^{\circ n_{\mathrm{st}}} \circ \left((n_{\mathrm{pr}}\wedge)^{\circ \lceil \log_2 n_{\mathrm{pr}} \rceil}\right)^{[]}\right)_0$. According to (17), this is a tree with $n_{\mathrm{st}}n_{\mathrm{pr}} + 1$ leaves.

Proposition 2. *For $n_{st} \geqslant \log_2 n_{pr}$, the height of the tree published in the period is the following*

$$\mathrm{ht}\left((n_{pr}\wedge)^{\circ n_{st}} \circ \left((n_{pr}\wedge)^{\circ \lceil \log_2 n_{pr} \rceil}\right)^{[]}\right)_0 = \lfloor \log_2 n_{pr} \rfloor + n_{st}. \tag{22}$$

Proof. Both parts of (22) equal to $\mathrm{ht}\left((n_{\mathrm{pr}}\wedge)^{\circ n_{\mathrm{st}}}\right)_0 + \mathrm{ht}\left((n_{\mathrm{pr}}\wedge)^{\circ \lceil \log_2 n_{\mathrm{pr}} \rceil}\right)_1$. □

Example 1. *Let $n_{pr} = 2^m$ and $n_{st} \geqslant m$; then, the period of the buffers consists of the single sequence (21) of perfect trees*

$$((2^m\wedge)^{\circ m})^{[]} = (t^\star_m, t^\star_{m-1}, t^\star_{m-1}, \ldots, \underbrace{t^\star_1, \ldots, t^\star_1}_{2^{m-1}}, t^\star_0 \ldots).$$

Let us consider a semi-infinite matrix, where rows are given by (21) for $n_{\mathrm{pr}} \geqslant 1$, and the corresponding matrix, where each tree is replaced by the number of its leaves. Elements $\left((n\wedge)^{\circ\lceil\log_2 n\rceil}\right)_0 = t^\star_{\lceil\log_2 n\rceil}$ are removed after the shift. The matrix is lower triangular in the sense that $\left((n\wedge)^{\circ\lceil\log_2 n\rceil}\right)_m = \bullet$ for $m \geqslant n \geqslant 1$.

Proposition 3. *Let us denote $f_{k,p} = \left(p\cdot\wedge, (2^k-p)\cdot\bullet\right)$ the sequence of trees of length 2^k for $0 \leqslant p \leqslant 2^k$.*

1. *For each $m \geqslant 1$, the following sequence can be presented as the concatenation*

$$\left(\left((n\wedge)^{\circ\lceil\log_2 n\rceil}\right)_m\right)_{n>m} = \bigoplus_{k=1}^{\infty}\left(2^{k-1}(m-1)t^\star_k, t^\star_k\circ f_{k,1}, t^\star_k\circ f_{k,2}, \ldots, t^\star_k\circ f_{k,2^k}\right). \quad (23)$$

2. *For $n \geqslant m \geqslant 1$,*

$$\left((n\wedge)^{\circ\lceil\log_2 n\rceil}\right)_m = \begin{cases} t^\star_k\circ f_{k,n-2^k m}, & \text{if } \lceil\log_2\frac{n}{m+1}\rceil = \lfloor\log_2\frac{n}{m}\rfloor = k,\\ t^\star_k, & \text{if } \lceil\log_2\frac{n}{m}\rceil = \lfloor\log_2\frac{n}{m+1}\rfloor + 1 = k.\end{cases}$$

Proof. For each $m \geqslant 1$, the sequence (23) is the concatenation over all positive integers k of sequences consisting of $2^{k-1}(m-1)$ copies of $t^\star_k$ and then compositions $t^\star_k\circ(p\wedge)$ for p from 1 to 2^k. (Note that $t^\star_k\circ(2^k\wedge) = t^\star_{k+1}$.) The tree $t^\star_k$ appears in the positions $2^{k-1}(m+1) \leqslant n \leqslant 2^k m$, or, equivalently, $\frac{n}{m} \leqslant 2^k \leqslant \frac{2n}{m+1}$. The tree $t^\star_k\circ(p\wedge)$ appears in the position $2^k m < n = p + 2^k m \leqslant 2^k(m+1)$, or, equivalently, $\frac{n}{m+1} \leqslant 2^k < \frac{n}{m}$. □

Corollary 5. *For each $m \geqslant 1$,*

$$\left(\left((n\cdot\mathbf{2})^{\circ\lceil\log_2 n\rceil}\right)_m\right)_{n\geqslant m} = (1, \underbrace{2,\ldots,2}_{m}, 3, \ldots, 2^{k+1}-1, \underbrace{2^{k+1},\ldots,2^{k+1}}_{2^k(m-1)+1}, 2^{k+1}+1, \ldots) \quad (24)$$

is a sequence of positive integers with inserted additional $(m-1)\cdot 2^k$ copies of 2^{k+1} for each $k \geqslant 0$.

4.2. Generation of Perfect Binary Trees

Let us recall that the perfect tree is completely determined by its height. Thus, in this subsection, we consider the sequences $h = (h_i)_{i\geqslant 0}$ of heights of perfect trees instead of perfect trees itself: $t = \left(t^*_{h_i}\right)_{i\geqslant 0}$. Buffer states and published trees are characterized by the sequences of heights $h^{(n,k)}$ and $h^{\copyright}$ with $t^{(n,k)}_i = t^\star_{h^{(n,k)}_i}$ and $t^{\copyright}_i = t^\star_{h^{\copyright}_i}$.

According to Lemma 2, at each moment, the sequence of heights of perfect trees in the buffer is non-increasing. By Lemma 3, the heights of trees are bounded by the constant $h_{\max}$, the same for all buffer states.

Let $\mu_h = \mu^{(n,k)}_h$ be the number of perfect trees in the buffer of positive height h, i.e., the buffer at each step has the form

$$h^{(n,k)} = \left(h_{\max}^{\times\mu_{h_{\max}}}, (h_{\max}-1)^{\times\mu_{h_{\max}-1}}, \ldots, 1^{\times\mu_1}, 0^{\times\infty}\right), \quad (25)$$

where $h_i^{\times\mu} := \underbrace{h_i, \ldots, h_i}_{\mu}$.

The following Lemma describes these parameters for a fixed number of provers n_{pr} and number of steps n_{st}.

Lemma 8. *At every step,*

$$\mu_h \leqslant \frac{n_{pr}-1}{2^{h-1}} + 2, \qquad 1 \leqslant h \leqslant h_{\max}; \tag{26}$$

$$\sum_{h=1}^{h_{\max}} \left\lfloor \frac{\mu_h}{2} \right\rfloor \leqslant n_{pr}. \tag{27}$$

The map f corresponding to one step is given by the formula

$$\mu_h^{(n,k+1)} = 2\{\mu_h^{(n,k)}/2\} + \begin{cases} \lfloor \mu_{h-1}^{(n,k)}/2 \rfloor, & \text{if } h > 1, \\ n_{pr} - \sum_{h'} \lfloor \mu_{h'}^{(n,k)}/2 \rfloor, & \text{if } h = 1. \end{cases} \tag{28}$$

Here $2\{p/2\} \in \{0,1\}$ is the parity of p.

Proof. The inequality (26) is proved by induction. Let $\mu_h^{\max} = \max_{n,k} \mu_h^{(n,k)}$. Then $\mu_1^{\max} \leqslant 1 + n_{\mathrm{pr}}$ and $\mu_{h+1}^{\max} \leqslant 1 + \left\lfloor \frac{\mu_h^{\max}}{2} \right\rfloor \leqslant \frac{n_{\mathrm{pr}}-1}{2^h} + 2$.

The inequality (27) and the formula (28) are proved together. If (27) is satisfied, then the quantity of provers is sufficient to build all possible non-zero height perfect trees and, hence, we obtain (28). Now, suppose that $\sum_h \left\lfloor \frac{\mu_h}{2} \right\rfloor \leqslant n_{\mathrm{pr}}$; then, at the next step, by (28),

$$\begin{aligned} \sum_h \left\lfloor \frac{\mu_h^{(n,k+1)}}{2} \right\rfloor &= \left\lfloor \left\{ \frac{\mu_1^{(n,k)}}{2} \right\} + \frac{n_{\mathrm{pr}}}{2} - \frac{1}{2} \sum_{h'} \left\lfloor \frac{\mu_{h'}^{(n,k)}}{2} \right\rfloor \right\rfloor + \sum_{h=2}^{h_{\max}} \left\lfloor \left\{ \frac{\mu_h^{(n,k)}}{2} \right\} + \frac{1}{2} \left\lfloor \frac{\mu_{h-1}^{(n,k)}}{2} \right\rfloor \right\rfloor \\ &\leqslant \left\lfloor \frac{1}{2} + \frac{n_{\mathrm{pr}}}{2} - \frac{1}{2} \sum_{h'} \left\lfloor \frac{\mu_{h'}^{(n,k)}}{2} \right\rfloor \right\rfloor + \sum_{h=2}^{h_{\max}} \left\lfloor \frac{1}{2} + \frac{1}{2} \left\lfloor \frac{\mu_{h-1}^{(n,k)}}{2} \right\rfloor \right\rfloor \\ &\leqslant \left\lfloor \frac{2n_{\mathrm{pr}}+1}{2} \right\rfloor = n_{\mathrm{pr}}, \end{aligned}$$

where the last inequality follows from the fact that the number of non-zero summands in (27) is not greater than n_{pr}. □

Remark 2. *The above lemma allows for constructing an optimized implementation of OneStep() procedure according to formula (28), and then to obtain a (pre-)reduced form of $h^{\copyright}$ according to Definition 5. The remainder of the results in this subsection essentially use this implementation.*

Let $h^{\copyright} = h^{\copyright}(n_{\mathrm{pr}}, n_{\mathrm{st}})$ be the sequence of heights of trees in generated blocks. According to Corollary 3, it is ultimately periodic.

Hypothesis 1.

$$h_{\max} = \lfloor \log_2 n_{st} n_{pr} \rfloor + 1 = \min\{k \in \mathbb{Z}_{>0} \mid 2^k > n_{st} n_{pr}\}.$$

Moreover, $\mu_{h_{\max}} = 0$ after block publishing.

Hypothesis 2. *A period of $h^{\copyright}$ consists of perfect trees of heights $h_{\max -1}$ and $h_{\max}$ in some order.*

Corollary 6. *Let a (respectively b) be the number of perfect trees of height $h_{\max -1}$ (respectively $h_{\max}$). Then, the length of the period is*

$$a + b = 2^{\lfloor \log_2 n_{st} n_{pr} \rfloor - \nu},$$

for some $\nu \in \{0, 1, \ldots, \nu_2(n_{st} n_{pr} + 1)\}$, where $\nu_2(n_{st} n_{pr} + 1)$ is the multiplicity of 2 in prime factorization of $n_{st} n_{pr} + 1$, and

$$a = 2^{\lfloor \log_2 n_{st} n_{pr} \rfloor + 1 - \nu} - \frac{n_{st} n_{pr} + 1}{2^\nu}, \qquad b = \frac{n_{st} n_{pr} + 1}{2^\nu} - 2^{\lfloor \log_2 n_{st} n_{pr} \rfloor - \nu}.$$

Remark 3. *The difference $h^{\copyright}_{i+1} - h^{\copyright}_i$ can be arbitrarily large:*

$$h^{\copyright}(n_{pr}, n_{st}) = n_{st}, 2n_{st}, \ldots, \qquad \text{whenever} \quad n_{pr} \geqslant 2^{2n_{st}-1} + 2^{n_{st}-1}.$$

Definition 5. *The presentation $h^{\copyright} = h^{\copyright}_1, \ldots, h^{\copyright}_r, (h^{\copyright}_{r+1}, \ldots, h^{\copyright}_{r+s})$ of the sequence (of heights) of published trees with the smallest pre-period and period is called the* reduced form *of $h^{\copyright}$.*

If $h^{(1)}, \ldots, h^{(r')}, (h^{(r'+1)}, \ldots, h^{(r'+s')})$ is the presentation of the sequence of buffer states with the smallest pre-period and period, then the corresponding presentation of the sequence (of heights) of published trees $h^{\copyright} = h^{\copyright}_1, \ldots, h^{\copyright}_{r'}, (h^{\copyright}_{r'+1}, \ldots, h^{\copyright}_{r'+s'})$ is called the pre-reduced form *of $h^{\copyright}$.*

Remark 4. *The pre-reduced and reduced forms of $h^{\copyright}$ can be different as in the following cases:*

$$\begin{aligned} h^{\copyright}(2^6 + 1, 1) &= 1, 2, 3, 4, 5, 6^{\times 31}, (6^{\times 30}, 7, 6) \\ &= 1, 2, 3, 4, 5, 6^{\times 30}, (6^{\times 31}, 7); \end{aligned}$$

$$\begin{aligned} h^{\copyright}(360, 1) &= 1, 2, 3, 4, 5, 6, 7, 8^{\times 4}, 9, 8, (\underbrace{8, 9, \ldots, 9, 8}_{2^8}) \\ &= 1, 2, 3, 4, 5, 6, 7, 8^{\times 4}, 9, (\underbrace{8^{\times 2}, 9, \ldots, 9}_{2^8}); \end{aligned} \tag{29}$$

$$\begin{aligned} h^{\copyright}(361, 1) &= 1, 2, 3, 4, 5, 6, 7, 8^{\times 4}, 9, 8^{\times 2}, (\underbrace{9, 8, \ldots, 9, 8^2}_{2^7}) \\ &= 1, 2, 3, 4, 5, 6, 7, 8^{\times 4}, 9, (\underbrace{8^{\times 2}, 9, 8, \ldots, 9}_{2^7}). \end{aligned} \tag{30}$$

Hypothesis 3. *The pre-period of the reduced form of $h^{\copyright}$ is a non-decreasing sequence.*

Remark 5. *The above hypothesis is not true for the pre-reduced form of $h^{\copyright}$ as we can see in (29) and (30).*

4.2.1. The Case of a Single Prover $n_{\mathrm{pr}} = 1$

In the case of a single prover Conjectures 1 and 2 are obvious, moreover explicit description is obtained.

Proposition 4. *Let $2^h \leqslant k + 1 \leqslant 2^{h+1}$. In the case of a single prover $n_{pr} = 1$, the sequence $h^{\copyright} = h^{\copyright}(1, k)$ can be determined recursively together with an auxiliary sequence g that calculates the number of generated proofs in the buffer*

$$\begin{aligned} g_0 = 0, \quad h^{\copyright}_n = \lfloor \log_2(g_n + k + 1) \rfloor &= \begin{cases} h, & \text{if } g_n + k < 2^{h+1} - 1, \\ h + 1, & \text{otherwise}, \end{cases} \\ g_{n+1} &= g_n + k - 2^{h^{\copyright}_n} + 1. \end{aligned} \tag{31}$$

Corollary 7. *In the case of a single prover $n_{pr} = 1$, the sequence $h^{\copyright}(1,k)$ is periodic.*

Let $k, k' \in [2^h - 1, 2^{h+1} - 1]$ and $k + k' = 2^{h+1} + 2^h - 2$. Then, the periods of reduced forms of $h^{\copyright}(1,k)$ and $h^{\copyright}(1,k')$ are obtained each from the other with order reversing and replacement $h \leftrightarrow h + 1$.

Proof. The sequence of the buffer state is periodic because at each step the increment of the number of generated proofs in the buffer is constant k by module $2^h - 1$.

If $(q_0, q_1, \ldots, q_{s-1})$ is the period of the sequence of the number of generated proofs in the buffer for k steps, then $(2^h - q_{s-1}, \ldots, 2^h - q_1, 2^h - q_0)$ is the period of the sequence of the number of generated proofs in the buffer for $k' = 2^{h+1} + 2^h - 2 - k$ steps. Equation (31) for the first data implies the same equations for the second data. □

Example 2. *The cases of $n_{pr} = 1$ and n_{st} are closed to $2^h - 1$. The following presentations are special cases (31):*

- $k_1 = 2^h - 1, k_2 = 2^{h+1} - 1$

$$h^{\copyright}(1, 2^h - 1) = (h), \qquad h^{\copyright}(1, 2^{h+1} - 1) = (h + 1).$$

- $k_1 = 2^h, k_2 = 2^{h+1} - 2$

$$h^{\copyright}(1, 2^h) = (h^{\times(2^h-1)}, h + 1), \qquad h^{\copyright}(1, 2^{h+1} - 2) = (h, (h + 1)^{\times(2^h-1)}).$$

- $k_1 = 2^h + 1, k_2 = 2^{h+1} - 3$

$$h^{\copyright}(1, 2^h) + 1 = (h^{\times(2^{h-1}-1)}, h + 1), \qquad h^{\copyright}(1, 2^{h+1} - 3) = (h, (h + 1)^{\times(2^{h-1}-1)}).$$

- $k_1 = 2^h + 2, k_2 = 2^{h+1} - 4$, *if* $h = 3, 5, 7, \ldots$

$$h^{\copyright}(1, 2^h + 2) = \left(h^{\times\frac{2^h-2}{3}}, h + 1, h^{\times\frac{2^h-2}{3}}, h + 1, h^{\times\frac{2^h-5}{3}}, h + 1\right),$$
$$h^{\copyright}(1, 2^{h+1} - 4) = \left(h, (h + 1)^{\times\frac{2^h-5}{3}}, h, (h + 1)^{\times\frac{2^h-2}{3}}, h, (h + 1)^{\times\frac{2^h-2}{3}}\right),$$

if $h = 4, 6, 8, \ldots$

$$h^{\copyright}(1, 2^h + 2) = \left(h^{\times\frac{2^h-1}{3}}, h + 1, h^{\times\frac{2^h-4}{3}}, h + 1, h^{\times\frac{2^h-4}{3}}, h + 1\right),$$
$$h^{\copyright}(1, 2^{h+1} - 4) = \left(h, (h + 1)^{\times\frac{2^h-4}{3}}, h, (h + 1)^{\times\frac{2^h-4}{3}}, h, (h + 1)^{\times\frac{2^h-1}{3}}\right).$$

4.2.2. The Case of a Single Step $n_{\mathrm{st}} = 1$

Hypothesis 4. *Let $k_1, k_2 \in [2^h - 1, 2^{h+1} - 1]$ and $k_1 + k_2 = 2^{h+1} + 2^h - 2$. Then the periods of reduced forms of $h^{\copyright}(k_1, 1)$ and $h^{\copyright}(k_2, 1)$ are obtained each from the other by order reversing and replacement $h \leftrightarrow h + 1$.*

Example 3. *The cases of $n_{st} = 1$ and n_{pr} closed to $2^h - 1$ agree with the above hypothesis:*

- $k_1 = 2^h - 1, k_2 = 2^{h+1} - 1$

$$h^{\copyright}(2^h - 1, 1) = 1, 2, \ldots, h - 1, (h), \qquad h^{\copyright}(2^{h+1} - 1, 1) = 1, 2, \ldots, h, (h + 1);$$

- $k_1 = 2^h, k_2 = 2^{h+1} - 2$

$$h^{\copyright}(2^h, 1) = 1, 2, \ldots, h - 1, h^{\times(2^h-h)}, (h^{\times(2^h-1)}, h + 1),$$

$$h^{\copyright}(2^{\times(h+1)}-2,1)=1,2,\dots,h,(h,(h+1)^{\times(2^h-1)});$$

- $k_1=2^h$, $k_2=2^{h+1}-2$, $h\geqslant 3$

$$h^{\copyright}(2^h+1,1)=1,2,\dots,h-1,h^{\times\left(2^{h-1}-\left\lfloor\frac{h-1}{2}\right\rfloor\right)},(h^{\times(2^{h-1}-1)},h+1),$$

$$h^{\copyright}(2^{h+1}-3,1)=1,2,\dots,h,(h,(h+1)^{\times(2^{h-1}-1)}).$$

5. Stochastic Case

In this section, we consider the result of Algorithm 1 only in the case of "b_{behavior} is stochastic".

5.1. Occupancy Distribution: Efficiency

Placement of provers into suitable positions at each stage can be described by the classical occupancy distribution. See [26], the more recent [27] and references therein. Another closely related model is a coupon collector problem.

Thus, we have $n_{\text{pr}}\geqslant 1$ provers placed independently and with equal probability into $n_{\text{pos}}\geqslant 1$ positions. In terms of urn models [26], provers and positions correspond to balls and bins, respectively. Let us denote $\xi^{n_{\text{pr}}n_{\text{pos}}}$ a random number of non-empty positions. It takes integer values i from 1 to $\min\{n_{\text{pr}},n_{\text{pos}}\}$ with probabilities

$$\mathbf{Pr}\{\xi^{n_{\text{pr}}n_{\text{pos}}}=i\}=n_{\text{pos}}^{-n_{\text{pr}}}\binom{n_{\text{pos}}}{i}i!\left\{{n_{\text{pr}}\atop i}\right\}=n_{\text{pos}}^{-n_{\text{pr}}}(n_{\text{pos}})_i\left\{{n_{\text{pr}}\atop i}\right\}.$$

Here, we use the following notations:

Notation 3. • $\left\{{n\atop m}\right\}$ *is the Stirling number of the second kind, i.e., the number of factorizations of an n-element set to an m-element factor-set;*
- $(n)_k=n(n-1)\cdots(n-k+1)$ *is the falling factorial.*

Mean and variance of the occupancy distribution (e.g., Theorem 2A [27]) are obtained as the special cases of expected values of falling factorials Theorem 1A [27]:

$$\mathbf{E}\big(n_{\text{pos}}-\xi^{n_{\text{pr}}n_{\text{pos}}}\big)_r=\big(n_{\text{pos}}\big)_rE_r,\qquad E_r=\big(1-r/n_{\text{pos}}\big)^{n_{\text{pr}}},\qquad 0\leqslant r<n_{\text{pos}};$$
$$\mathbf{E}\,\xi^{n_{\text{pr}}n_{\text{pos}}}=n_{\text{pos}}(1-E_1),\qquad \mathbf{Var}\,\xi^{n_{\text{pr}}n_{\text{pos}}}=n_{\text{pos}}\big((n_{\text{pos}}-1)E_2+E_1-n_{\text{pos}}E_1^2\big).$$

Let $N_t:=\operatorname{Len}\ell$ denote the total number of generated trees, N_ℓ and N_{int}, respectively, the numbers of leaves and internal vertices in these trees. Similarly let $N_t^{\copyright}=n_{\text{bl}}$ be the number of published trees, $N_\ell^{\copyright}$ and $N_{\text{int}}^{\copyright}$, respectively, the number of leaves and internal vertices in these trees.

Note that, according to (1),

$$N_\ell=N_{\text{int}}+N_t,\qquad N_\ell^{\copyright}=N_{\text{int}}^{\copyright}+n_{\text{bl}}. \tag{32}$$

Moreover, as a random variable,

$$N_{\text{int}}=n_{\text{bl}}n_{\text{st}}\xi^{n_{\text{pr}}n_{\text{pos}}}. \tag{33}$$

Throughput of our system can be naturally measured by the number of processed transactions, i.e., be the number of leaves N_ℓ and $N_\ell^{\copyright}$. On the other hand, the useful work of provers is given by the number of internal vertices N_{int} and $N_{\text{int}}^{\copyright}$. According to (32), if $n_{\text{st}}n_{\text{pr}}\gg 1$, then $N_{\text{int}}/N_\ell\approx 1$ and $N_{\text{int}}^{\copyright}/N_\ell^{\copyright}\approx 1$.

Note that in the cases that "b_{behavior} is deterministic", $N_{\text{int}}=n_{\text{bl}}n_{\text{st}}n_{\text{pr}}$. We use this to define the "normalized" values:

- proof generation efficiency

$$\mathrm{Ef}(n_{\mathrm{pr}}, n_{\mathrm{pos}}) := \mathbf{E}\,\frac{N_{\mathrm{int}}}{n_{\mathrm{bl}} n_{\mathrm{st}} n_{\mathrm{pr}}} = \mathbf{E}\,\frac{\xi^{n_{\mathrm{pr}} n_{\mathrm{pos}}}}{n_{\mathrm{pr}}} = \frac{n_{\mathrm{pos}}}{n_{\mathrm{pr}}}\Big(1 - \big(1 - 1/n_{\mathrm{pos}}\big)^{n_{\mathrm{pr}}}\Big), \tag{34}$$

- proof publishing efficiency

$$\mathrm{Ef}^{\copyright}(n_{\mathrm{bl}}, n_{\mathrm{st}}, n_{\mathrm{pr}}, n_{\mathrm{pos}}) := \mathbf{E}\,\frac{N^{\copyright}_{\mathrm{int}}}{n_{\mathrm{bl}} n_{\mathrm{st}} n_{\mathrm{pr}}}. \tag{35}$$

Note that

$$0 < \mathrm{Ef}^{\copyright} \leqslant \mathrm{Ef} \leqslant 1.$$

There are two limit cases:

$$\lim_{n_{\mathrm{pos}} \to \infty} \mathrm{Ef}(n_{\mathrm{pr}}, n_{\mathrm{pos}}) = 1, \qquad \lim_{\substack{n_{\mathrm{pr}} \to \infty \\ n_{\mathrm{pos}} \to \infty \\ n_{\mathrm{pr}}/n_{\mathrm{pos}} \to \gamma}} \mathrm{Ef}(n_{\mathrm{pr}}, n_{\mathrm{pos}}) = \frac{1 - e^{-\gamma}}{\gamma}.$$

5.2. Simulation Model for Block Publishing Efficiencies

In both cases that "b_{shape} is strict" and "b_{shape} is perfect", $\mathrm{Ef}^{\copyright}$ is an increasing function of n_{bl}. In the cases that "b_{shape} is strict", the lengths of buffers $h^{(n)}$ are not greater then $n_{\mathrm{pos}} - 1$. This implies convergence in the following proposition.

Proposition 5. *In the case that "b_{shape} is strict", $\mathrm{Ef}^{\copyright} \nearrow \mathrm{Ef}$ whenever $n_{bl} \to \infty$.*

Hypothesis 5. *In the case that "b_{shape} is perfect", $\mathrm{Ef}^{\copyright} \nearrow \mathrm{Ef}$ whenever $n_{bl} \to \infty$.*

We try to illustrate the above convergences in Figure 4. As we can see, in the case that "b_{shape} is strict", $\mathrm{Ef}^{\copyright}$ tends quickly to Ef. On the contrary, in the case that "b_{shape} is perfect", the convergence is very slow. Note that the limit value is $\mathrm{Ef}\,|_{n_{\mathrm{pr}}=n_{\mathrm{pos}}=5} = 0.67232$, and some values close to asymptote are $\mathrm{Ef}^{\copyright}\,|_{n_{\mathrm{bl}}=1000} = 0.6359$, $\mathrm{Ef}^{\copyright}\,|_{n_{\mathrm{bl}}=2000} = 0.6451$, $\mathrm{Ef}^{\copyright}\,|_{n_{\mathrm{bl}}=3000} = 0.6497$, $\mathrm{Ef}^{\copyright}\,|_{n_{\mathrm{bl}}=5000} = 0.6541$.

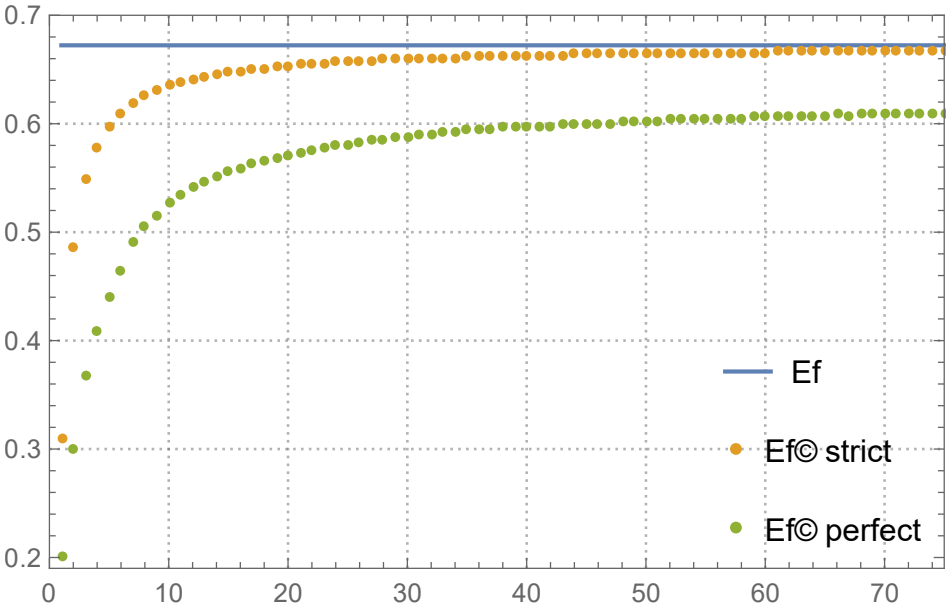

Figure 4. Dependencies of $\mathrm{Ef}^{\copyright}$ on the number of blocks for $n_{\mathrm{st}} = n_{\mathrm{pr}} = n_{\mathrm{pos}} = 5$.

In practice, it is important to select a right value of n_{pos} when other parameters are fixed. In Figure 5, dependencies of Ef and Ef$^{©}$ (for both cases "b_{shape} is strict" and binary and "b_{shape} is perfect") on the number of positions are shown in the case when $n_{\text{bl}} = 100$, $n_{\text{st}} = 10$, $n_{\text{pr}} = 10$. Values for efficiencies are average over 300 random calculations.

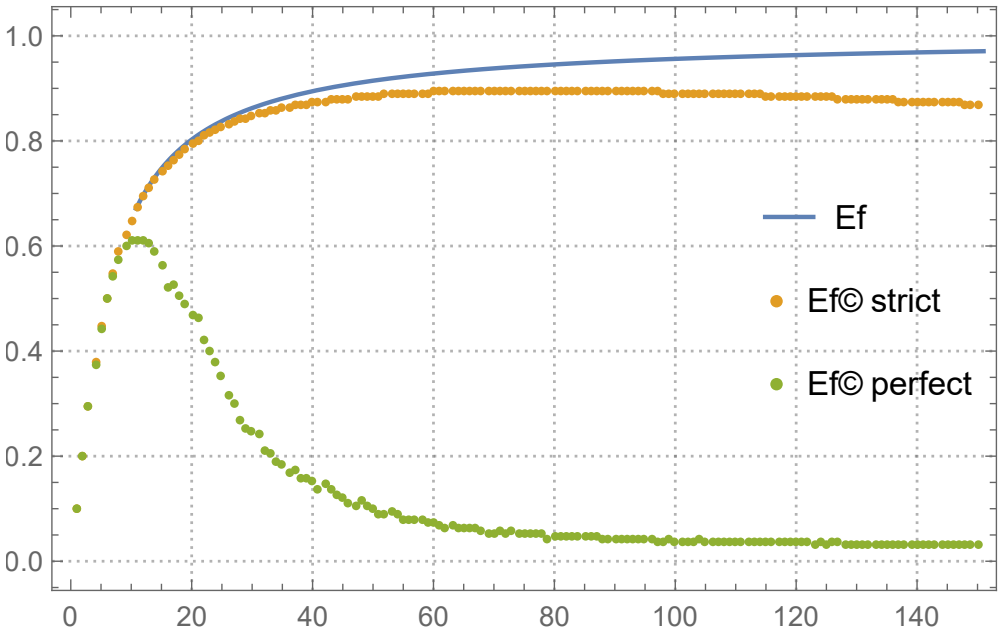

Figure 5. Dependencies of Ef and Ef$^{©}$ on the number of positions.

In Table 2, for the case "b_{shape} is perfect" and in Table 3 for the case "b_{shape} is strict", the arguments of maxima and the corresponding maximal values of blocks publishing efficiency Ef$^{©}$ are given, as functions of the number of positions n_{pos} for $n_{\text{bl}} = 200$ and various values of n_{st} and n_{pr}.

Table 2. ArgMax and Max of the function $n_{\text{bl}} \mapsto \text{Ef}^{©}$ for the case "b_{shape} is perfect".

$n_{\text{st}} \backslash n_{\text{pr}}$	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	7 0.89	8 0.83	8 0.78	8 0.72	8 0.67	9 0.62	9 0.58	9 0.55	9 0.52
2	4 0.85	5 0.74	5 0.67	6 0.62	6 0.58	7 0.56	7 0.53	7 0.51	8 0.49
3	3 0.80	4 0.70	5 0.65	5 0.60	6 0.57	6 0.55	7 0.53	7 0.51	8 0.50
4	3 0.78	4 0.70	5 0.65	6 0.61	6 0.58	7 0.57	8 0.56	9 0.54	9 0.53
5	3 0.77	4 0.71	5 0.66	6 0.63	7 0.61	7 0.59	8 0.56	9 0.55	9 0.53
6	3 0.79	4 0.72	5 0.67	6 0.65	7 0.62	7 0.60	7 0.58	8 0.57	9 0.55
7	3 0.79	4 0.71	6 0.68	6 0.65	7 0.63	7 0.61	8 0.60	9 0.59	10 0.59
8	3 0.79	4 0.73	6 0.70	6 0.67	7 0.65	8 0.64	9 0.62	10 0.61	10 0.60
9	3 0.80	5 0.74	6 0.70	6 0.68	8 0.66	8 0.65	10 0.64	10 0.61	11 0.61
10	3 0.81	5 0.75	6 0.72	7 0.70	8 0.68	9 0.66	10 0.64	10 0.63	11 0.62

Table 3. ArgMax and Max of $n_{\mathrm{bl}} \mapsto \mathrm{Ef}^{\copyright}$ for the case "b_{shape} is strict".

$n_{\mathrm{st}} \backslash n_{\mathrm{pr}}$	2	3	4	5
1	13/0.92	19/0.90	23–24/0.89	28–29/0.88
2	14/0.93	21–22/0.91	28/0.90	34–35/0.89
3	15/0.94	23–24/0.92	32/0.91	38–39/0.90
4	16/0.94	25–26/0.93	33–34/0.92	41–42/0.91
5	18/0.95	27–28/0.93	36–37/0.92	44–46/0.92

5.3. Simulation Model for Heights of Strict Binary Trees

Here, we consider an analog of the formula (22) for average heights h of published strict binary trees in the case when "b_{behavior} is stochastic". We consider a linear approximation

$$h(n_{\mathrm{st}}, n_{\mathrm{pr}}) \approx \alpha n_{\mathrm{st}} + \beta \log_2 n_{\mathrm{pr}} + \gamma.$$

Optimal parameters α, β, γ are obtained by means of the least squares method. For a fixed positive integer m, we consider all pairs of integers $(n_{\mathrm{st}}, \log_2 n_{\mathrm{pr}}) = (i, j)$ satisfying $1 \leqslant j \leqslant i \leqslant m$. Thus, the problem is to minimize the square of the error vector

$$r = \sum_{i=1}^{m} \sum_{j=1}^{i} \Big(\alpha i + \beta j + \gamma - h(i, 2^j) \Big)^2.$$

The critical point (α, β, γ) is the solution of the system of linear equations

$$\begin{cases} \frac{\partial r}{\partial \alpha} = 0 \\ \frac{\partial r}{\partial \beta} = 0 \\ \frac{\partial r}{\partial \gamma} = 0 \end{cases} \quad \Leftrightarrow \quad \sum_{i=1}^{m} \sum_{j=1}^{i} \begin{pmatrix} i \\ j \\ 1 \end{pmatrix} \left(\begin{pmatrix} i & j & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} - h(i, 2^j) \right) = 0. \tag{36}$$

The inverse matrix for this system is:

$$\left(\sum_{i=1}^{m} \sum_{j=1}^{i} \begin{pmatrix} i \\ j \\ 1 \end{pmatrix} \begin{pmatrix} i & j & 1 \end{pmatrix} \right)^{-1} = \frac{6}{(m+2)_4} \begin{pmatrix} 8 & -4 & -4m \\ -4 & 8 & -4 \\ -4m & -4 & 3m^2 + 3m + 2 \end{pmatrix}.$$

It can be calculated using Wolfram Mathematica. Enthusiasts can verify this using formulas for sums $\sum_{k=1}^{n} k^{(p)} = \frac{1}{p+1} n^{(p+1)}$ of rising factorial powers for $k^{(p)} = k(k+1) \cdots (k+p-1)$ or the Faulhaber's formulas for generalized harmonic numbers $H_{n,-p} := \sum_{k=1}^{n} k^p$.

The results of numerical experiments (coefficients for linear approximation and the standard deviation) are presented in Table 4.

Table 4. Linear approximations of average height of generated trees as functions of n_{st} and $\log_2 n_{\mathrm{pr}}$.

m	α	β	γ	$\sqrt{\frac{2r}{m(m+1)}}$
6	0.84	1.35	−0.75	0.072
7	0.81	1.40	−0.75	0.074
8	0.79	1.43	−0.74	0.075
9	0.77	1.46	−0.74	0.075
10	0.75	1.49	−0.73	0.075

For the case $m = 8$, the graphical presentation of average values $h(i, 2^j)$ and their linear approximation are given in Figure 6.

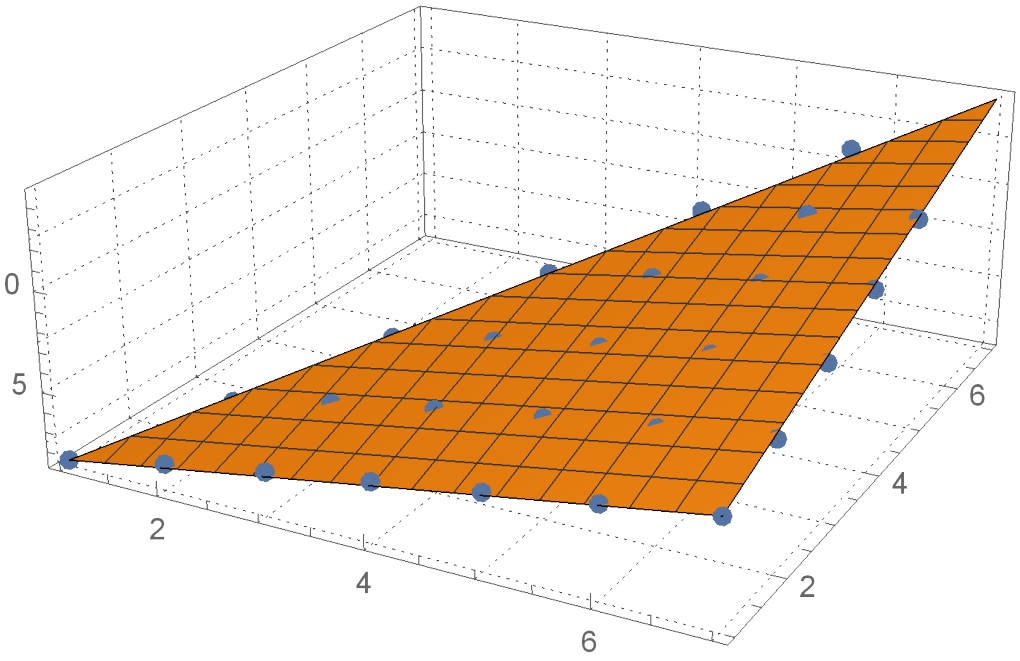

Figure 6. Average heights of generated trees and their linear approximation for $m = 8$.

6. Discussion

Latus consensus, proposed in [3,4], provides secure and stable sidechain operation, even under the complete distrust assumption in sidechains, and even when all participants in the sidechain consensus are malicious. We assume that the adversary in the sidechain may try to sensor the transactions of other participants inside the SC, or from the MC to the SC, or try to steal tokens in the SC, hide the state of the SC, or stop block production in the SC, but these actions must not be successful. For reliable operation under such assumptions, Latus consensus needs uninterruptible zk-SNARK-proof generation, which, in turn, demands essential computational power and cannot be performed by a single regular participant with commodity hardware. Projects Coda [8] or Polygon [28] implicitly involve a semi-trusted participant with powerful computational capabilities (such as cloud computation) to solve this task on time, but this approach may lead to a single failure point. To eliminate this problem we propose a new approach for efficient decentralization proof generation. According to our approach, decentralized generation is applied not only to blocks in the whole epoch but even to each block. Thus, proofs are generated with the involvement of a large number of independent participants, which are called provers. This provides a decentralization of the zk-SNARK sidechain operation in secure and reliable way.

In this article, we introduced the properties of this process as proof generation efficiency Ef and proof publishing efficiency $\mathrm{Ef}^{\copyright}$, which are investigated in Section 5.1.

For the generation of a single block considered in [18], the block forger initially selects the number of levels ℓ of a perfect binary tree that will be included into a block. In [18] (table 1), we find what number of provers is needed to build the perfect binary tree with $\ell = 4, 5, 6, 7, 8, 9$ levels during $n_{\mathrm{st}} = 9$ steps with probability $\geqslant 0.95$. In this limit case, the role of the efficiency garnered from this article results in the ratio of the number of useful proofs, which is the number of internal vertices $n_i(t)$ in the built tree to the total number of proofs $n_{\mathrm{st}} n_{\mathrm{pr}}$ produced by all provers during the block publishing process

$$\mathrm{Ef}_1 \approx \frac{n_i(t)}{n_{\mathrm{st}} n_{\mathrm{pr}}} = \frac{2^{\ell-1} - 1}{9 n_{\mathrm{pr}}}. \tag{37}$$

In Table 5, we compare the values of Ef_1 from (37) with the efficiencies $\mathrm{Ef}^{©}$ obtained by the emulation procedure when b_{shape} is perfect in three cases: $n_{\mathrm{bl}} = 10$, $n_{\mathrm{bl}} = 100$ and $n_{\mathrm{bl}} = n_{\mathrm{pr}}$, and when b_{shape} is strict and $n_{\mathrm{bl}} = 10$ (denoted $\mathrm{Ef}^{©}_{\mathrm{perfect}}$ and $\mathrm{Ef}^{©}_{\mathrm{strict}}$, respectively). The value of Ef calculated from (34) is the limit for both $\mathrm{Ef}^{©}_{\mathrm{perfect}}$ and $\mathrm{Ef}^{©}_{\mathrm{strict}}$ at $n_{\mathrm{bl}} \to \infty$.

Table 5. Comparison of efficiencies for $n_{\mathrm{st}} = 9$ steps and $n_{\mathrm{pos}} = n_{\mathrm{pr}}$.

n_{pr}	ℓ	$\mathbf{Ef_1}$	$\mathbf{Ef}^{©}_{\mathbf{perfect}}$ $n_{\mathbf{bl}} = 10$	$\mathbf{Ef}^{©}_{\mathbf{perfect}}$ $n_{\mathbf{bl}} = 100$	$\mathbf{Ef}^{©}_{\mathbf{perfect}}$ $n_{\mathbf{bl}} = n_{\mathbf{pr}}$	$\mathbf{Ef}^{©}_{\mathbf{strict}}$ $n_{\mathbf{bl}} = 10$	**Ef**
3	4	0.26	0.64	0.69	0.50	0.69	0.70
4	4	0.19	0.64	0.67	0.54	0.67	0.68
10	5	0.17	0.48	0.59	0.50	0.62	0.65
33	6	0.10	0.34	0.45	0.43	0.58	0.64
95	7	0.07	0.18	0.27	0.26	0.56	0.63
452	8	0.03	0.05	0.09	0.11	0.53	0.63
2176	9	0.01	0.02	0.03	0.04	0.50	0.63

So, one can see $\mathrm{Ef}^{©}/\mathrm{Ef}_1 \approx 3$ as the average . In the top half of Table 5, the values of $\mathrm{Ef}^{©}$ are closed to the best possible limit Ef. At the bottom, when the number of provers becomes extreme, $\mathrm{Ef}^{©}_{\mathrm{perfect}}$ still obtains an increment with respect to Ef_1; however, both values are too small. The convergence $\mathrm{Ef}^{©}_{\mathrm{perfect}} \to \mathrm{Ef}$ is slow and a very big buffer size is required in this situation. However, $\mathrm{Ef}^{©}_{\mathrm{strict}}$ remains closed to Ef.

7. Conclusions

We investigated characteristics of blocks and characteristics of a SC in a whole under various parameters of a SC, such as the number of blocks in an epoch, number of provers, number of available transactions, time of block creation, etc. We obtained theoretical results for a model with additional restrictions on provers' behavior and various experimental results for an extended model without such restrictions. These results are helpful when determining the parameters of a sidechain, such as epoch length, block creation time, proof incentives, etc. We show that in deterministic cases the sequences of published trees are ultimately periodic and ensure the highest possible efficiency of the proof creation process, because in this case there are no collisions in proof creation.

In stochastic cases, we obtain a universal measure of prover efficiencies taking values in the interval [0,1] and given by the explicit formula in one case, calculated by simulation models in other cases.

In the sense of efficiency defined, the optimal number of allowed prover positions for a step can be proposed for various sidechain parameters, such as number of provers, number of steps in a block, and so on. We also considered non-perfect binary tree utilization in a blockchain, and described benefits and restrictions for such trees' use. It turns out that we can achieve large efficiency using these very trees.

Our algorithm for the strict binary trees gives the following differences (compared with the case of perfect binary trees):

1. The buffer size is smaller;
2. Block publishing efficiency is higher;
3. The average height of the generated perfect binary trees is proportional to $\log(n_{\mathrm{st}} n_{\mathrm{pr}}) = \log n_{\mathrm{st}} + \log n_{\mathrm{pr}}$. The average height of the generated strict binary trees has a similar linear dependence on the logarithm of the number of provers $\log n_{\mathrm{pr}}$, but a different linear dependence to the number of steps n_{st} (not $\log n_{\mathrm{st}}$). So, we can consider the case of strict binary trees practically interesting only if the number of steps n_{st} is small.

All results from [18–20] and the present paper are related and, put together, give a comprehensive description of SC behavior under our chosen parameters, allowing us

to find the optimal sets of parameters that help to optimize functioning, in particular increasing throughput.

One of the most interesting results in this article shows that, under the condition of quick synchronization among block forgers, the throughput increases up to three times in practical settings and can be increased even more for extreme settings cases .

One topic of future research is the investigation of the efficiency of Verkle tree [29] and Curve tree structures [30], which can be used for creating proof trees instead of using different variants of Merkle trees.

Author Contributions: Conceptualization, R.O.; Software, H.N.; Validation, R.V.; Formal analysis, Y.B. and L.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Research Foundation of Ukraine under Grant 2020.01/0351.

Data Availability Statement: Presented data was generated using the program with its source code available at https://github.com/annanelasa/ProofStream (accessed on 9 December 2022).

Acknowledgments: We would like to thank Alberto Garoffolo for the fruitful discussion and comments.

Conflicts of Interest: The authors declare that they have no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

zk-SNARK	Zero-Knowledge Succinct Non-Interactive Argument of Knowledge
SC	Sidechain
MC	Mainchain
PoW	Proof of work
PoS	Proof of stake
iff	if and only if
PRO	product category

Appendix A. Monoid from Operad

Here we describe two constructions of the monoidal category from a plain operad and the monoid from the monoidal category. Necessary knowledge about category theory and about operads can be obtained in [31–35], respectively, while [36] serves as a begining introduction to both subjects. The monoids from Section 2.2 are results of these constructions applied to the magma and monoid operads.

Let us recall the direct definition a non-symmetric set operad.

Definition A1. *A* non-symmetric operad *is a sequence* $(P(n))_{n\geqslant 1}$ *of sets, whose elements are called n-ary operations, equipped with*

- *for all positive integers* $n, m_1, \ldots, m_n$*, a composition function*

$$\begin{aligned} \gamma_{n;m_1,\ldots,m_n} : P(n) \times P(m_1) \times \cdots \times P(m_n) &\to P(m_1 + \cdots + m_n) \\ (\theta, \theta_1, \ldots, \theta_n) &\mapsto \theta \circ (\theta_1, \ldots, \theta_n), \end{aligned} \tag{A1}$$

- *an element* $1 \in P(1)$ *called the identity,*

satisfying the following coherence axioms, associativity and identity:

$$\begin{aligned} &\theta \circ \Big(\theta_1 \circ (\theta_{1,1}, \ldots, \theta_{1,m_1}), \ldots, \theta_n \circ (\theta_{n,1}, \ldots, \theta_{n,m_n})\Big) \\ &\quad = \Big(\theta \circ (\theta_1, \ldots, \theta_n)\Big) \circ (\theta_{1,1}, \ldots, \theta_{1,m_1}, \ldots, \theta_{n,1}, \ldots, \theta_{n,m_n}), \end{aligned} \tag{A2}$$

$$\theta \circ (1, \ldots, 1) = \theta = 1 \circ \theta. \tag{A3}$$

A morphism of non-symmetric operads $P \to P'$ is a family of maps $\big(g_n : P(n) \to P'(n)\big)_{n\geqslant 1}$ compatible with compositions and units:

$$g_{m_1+\cdots+m_n}\gamma_{n;m_1,\ldots,m_n} = \gamma_{n;m_1,\ldots,m_n}(g_n \times g_{m_1} \times \cdots \times g_{m_n}), \qquad n, m_1, \ldots, m_n \in \mathbb{Z}_{>0},$$
$$g_1(1) = 1.$$

Example A1. *The components of a free magma with a single generator forms a non-symmetric magma operad $(\mathcal{M}_n)_{n\geqslant 1}$. If elements of $\mathcal{M}_n$ are interpreted as strict binary trees with n leaves, the composition $t \circ (t_1, \ldots, t_n)$ is obtained by gluing together of i-th leaf in t with the root of t_i for $i = 1, 2, \ldots n$, and the unit is zero height tree $\bullet$.*

The free semi-group with a single generator can be identified with the additive semigroup of positive integers $(\mathbb{Z}_{>0}, +)$. The non-symmetric semi-group operad has the single n-ary operation $(m_1, \ldots, m_n) \mapsto m_1 + \cdots + m_n$ for each $n > 0$.

The map $t \mapsto n_\ell(t)$ (number of leaves of strict binary tree) is the unique morphism between the magma operad and the semi-group operad.

Given a non-symmetric operad P, there exists a general functorial construction $P \mapsto \widehat{P}$ of the corresponding PRO. $\widehat{P}$ is a category whose objects are non-negative integers, morphisms are finite lists of operations:

$$m_1 + \cdots + m_n \xrightarrow{(\theta_1,\ldots,\theta_n)} n, \qquad \theta_i \in P(m_i),\ 1 \leqslant i \leqslant n$$

and for $\theta_i \in P(m_i)$, $1 \leqslant i \leqslant n$, the composition in $\widehat{P}$ is expressed via the operadic composition:

$$(\theta_1, \ldots, \theta_n) \circ (\theta_{1,1}, \ldots, \theta_{1,m_1}, \ldots, \theta_{n,1}, \ldots, \theta_{n,m_n}) := \big(\theta_1 \circ (\theta_{1,1}, \ldots, \theta_{1,m_1}), \ldots, \theta_n \circ (\theta_{n,1}, \ldots, \theta_{n,m_n})\big).$$

The category $\widehat{P}$ is equipped with a strict monoidal structure $(\oplus, 0)$ given by addition of objects and by concatenation of a list on morphisms:

$$\left(m \xrightarrow{(\theta_1,\ldots,\theta_n)} n\right) \oplus \left(m' \xrightarrow{(\theta'_1,\ldots,\theta'_{n'})} n'\right) := \left(m + m' \xrightarrow{(\theta_1,\ldots,\theta_n,\theta'_1,\ldots,\theta'_{n'})} n + n'\right).$$

Let us consider the pushout (i.e., the colimit) of the diagram of sets and functions:

$$\operatorname{Mor}\widehat{P} \xleftarrow{\mathrm{Id}} \operatorname{Mor}\widehat{P} \xrightarrow{-\oplus 1} \operatorname{Mor}\widehat{P},$$

involving the set $\operatorname{Mor}\widehat{P}$ of all morphisms in $\widehat{P}$. This is the factor-set $\operatorname{Mor}\widehat{P}/\sim$ by the transitive relation $\sim$ obtained by identifications $(\theta_1, \ldots, \theta_n) \sim (\theta_1, \ldots, \theta_n, 1)$ for each morphism $(\theta_1, \ldots, \theta_n)$ in $\widehat{P}$. Moreover, this factor-set can be naturally identified with a set of infinite sequences $(\theta_i)_{i\geqslant 0}$ with only finite number $\theta_i \neq 1$.

One can turn this set into a monoid. The composition of sequences θ and θ' with $\theta_i \in P(m_i)$ is described as

$$(\theta \circ \theta')_i = \theta_i \circ (\theta'_j)_{M_{i-1}\leqslant j<M_i}, \qquad M_i = \sum_{j=0}^{i} m_j.$$

The unit for this composition is the constant sequence $(1)_{i\geqslant 0}$.

The natural projection $\pi : \operatorname{Mor}\widehat{P} \to \operatorname{Mor}\widehat{P}/\sim$ is a functor with an additional property: $\pi(\theta \oplus 1) = \pi(\theta)$.

Appendix B. Software Implementation

The software is implemented in C♯ as a console application based on a single class with static procedures. Due to the peculiarities found during the studies, it turned out to

be convenient to implement 2×2 cases as separate procedures DP(), DS(), SP(), SS(), where the first (respectively second) letter in the name corresponds to the case when $b_{\text{behaviour}}$ is D[eterministic] or S[tochastic] (respectively b_{shape} is P[erfect] or S[trict]). In addition, we have procedures SPP(), SSP() which repeat simulations and with additional external loop over the number of positions. The Main() procedure calls one of the above 6 procedures depending on the input parameters, which corresponds to command lines ">ProofStream ..." with the following 4, 5, 6 or 8 parameters.

Two cases when $b_{\text{behaviour}}$ is deterministic:

- >ProofStream 1 0 n_{bl} n_{st} n_{pr}
  In the case b_{shape} is strict is the most simple. The corresponding calculations helped to formulate the results from Section 4.1 which all then was proved.
- >ProofStream 1 1 n_{st} n_{pr}
  In the case b_{shape} is perfect the output is the complete description of ultimately periodic sequences of buffer states and heights of published trees for special values of number of steps n_{st} and number of provers n_{pr}. Hypothesis 1–4 are based on numerous computation.

Two cases when $b_{\text{behaviour}}$ is stochastic require the additional parameter: the number of positions n_{pos}. They use a pseudo-random number generator with seeds obtained from cryptographic RNG.

- >ProofStream 0 0 n_{bl} n_{st} n_{pr} n_{pos}
- >ProofStream 0 1 n_{bl} n_{st} n_{pr} n_{pos}

Graphs in Figures 4–6 were obtained using Wolfram Mathematica. A part of data to plot as well as some result in Tables 2–4 are results of running loop over different parameter. In particular, the loop for(int $n_{\text{pos}} = \text{min}$; $n_{\text{pos}} < \text{max}$; $n_{\text{pos}} \mathrel{+}= \text{increment}$){...} is called by the command lines:

- >ProofStream 0 0 $n_{\text{simulations}}$ n_{bl} n_{st} n_{pr} min max increment
- >ProofStream 0 1 $n_{\text{simulations}}$ n_{bl} n_{st} n_{pr} min max increment

The initial C♯ code with a brief description and some calculation examples is available at the annanelasa/ProofStream repo on GitHub.

References

1. Back, A.; Corallo, M.; Dashjr, L.; Friedenbach, M.; Maxwell, G.; Miller, A.; Poelstra, A.; Timón, J.; Wuille, P. Enabling Blockchain Innovations with Pegged Sidechains. 2014. Available online: https://blockstream.com/sidechains.pdf (accessed on 27 February 2023).
2. Gaži, P.; Kiayias, A.; Zindros, D. Proof-of-work sidechains. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019; pp. 139–156. https://doi.org/10.1109/SP.2019.00040.
3. Garoffolo, A.; Kaidalov, D.; Oliynykov, R. Zendoo: A zk-SNARK Verifiable Cross-Chain Transfer Protocol Enabling Decoupled and Decentralized Sidechains. In Proceedings of the 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), Singapore, 29 November–1 December 2020; pp. 1257–1262. https://doi.org/10.1109/ICDCS47774.2020.00161.
4. Garoffolo, A.; Viglione, R. Sidechains: Decoupled Consensus Between Chains. *arXiv* **2018**, arXiv:1812.05441. https://doi.org/10.48550/arXiv.1812.05441.
5. Garay, J.; Kiayias, A.; Leonardos, N. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology—EUROCRYPT 2015, Part II*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9057, pp. 281–310. https://doi.org/10.1007/978-3-662-46803-6_10.
6. Ben-Sasson, E.; Chiesa, A.; Tromer, E.; Virza, M. Succinct Non-Interactive Zero Knowledge for a von Neumann architecture. In Proceedings of the 2014 23rd USENIX Conference on Security Symposium—SEC'14, San Diego, CA, USA, 20–22 August 2014; pp. 781–796. Available online: https://dl.acm.org/doi/abs/10.5555/2671225.2671275 (accessed on 27 February 2023).
7. Bowe, S.; Gabizon, A. Making Groth's zk-SNARK Simulation Extractable in the Random Oracle Model. 2018. Available online: https://ia.cr/2018/187 (accessed on 27 February 2023).
8. Bonneau, J.; Meckler, I.; Rao, V.; Shapiro, E. Coda: Decentralized Cryptocurrency at Scale. Cryptology ePrint Archive, Report 2020/352. 2020. Available online: https://ia.cr/2020/352 (accessed on 27 February 2023).
9. Matter Labs. zkSync Era Basics. 2023. Available online: https://era.zksync.io/docs/dev/fundamentals/zkSync.html (accessed on 27 February 2023).
10. Ochôa, I.S.; Silva, L.A.; de Mello, G.; Garcia, N.M.; de Paz Santana, J.F.; Leithardt, V.R.Q. A Cost Analysis of Implementing a Blockchain Architecture in a Smart Grid Scenario Using Sidechains. *Sensors* **2020**, *20*, 843. https://doi.org/10.3390/s20030843.

11. Zhou, J.; Wang, N.; Liu, A.; Wang, W.; Du, X. CBCS: A Scalable Consortium Blockchain Architecture Based on World State Collaborative Storage. *Electronics* **2023**, *12*, 735. https://doi.org/10.3390/electronics12030735.
12. Lee, N.Y. Hierarchical Multi-Blockchain System for Parallel Computation in Cryptocurrency Transfers and Smart Contracts. *Appl. Sci.* **2021**, *11*, 10173. https://doi.org/10.3390/app112110173.
13. Garoffolo, A.; Kaidalov, D.; Oliynykov, R. Trustless Cross-chain Communication for Zendoo Sidechains. Cryptology ePrint Archive, Report 2022/1179. 2022. Available online: https://ia.cr/2022/1179 (accessed on 27 February 2023).
14. Kiayias, A.; Zindros, D. Proof-of-Work Sidechains. Cryptology ePrint Archive, Report 2018/1048. 2018. Available online: https://ia.cr/2018/1048 (accessed on 27 February 2023).
15. Gaži, P.; Kiayias, A.; Zindros, D. Proof-of-Stake Sidechains. Cryptology ePrint Archive, Report 2018/1239. 2018. Available online: https://ia.cr/2018/1239 (accessed on 27 February 2023).
16. Kiayias, A.; Russell, A.; David, B.; Oliynykov, R. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO 2017, Part I*; Lecture Notes in Computer Science; Springer: Heidelberg, Germany, 2017; Volume 10401, pp. 357–388. https://doi.org/10.1007/978-3-319-63688-7_12.
17. Eagen, L. μCash: Transparent Anonymous Transactions. Cryptology ePrint Archive, Report 2022/1104. 2022. Available online: https://ia.cr/2022/1104 (accessed on 27 February 2023).
18. Bespalov, Y.; Garoffolo, A.; Kovalchuk, L.; Nelasa, H.; Oliynykov, R. Probability Models of Distributed Proof Generation for zk-SNARK-Based Blockchains. *Mathematics* **2021**, *9*, 3016. https://doi.org/10.3390/math9233016.
19. Bespalov, Y.; Garoffolo, A.; Kovalchuk, L.; Nelasa, H.; Oliynykov, R. Game-Theoretic View on Decentralized Proof Generation in zk-SNARK Based Sidechains. In *CEUR Workshop Proceedings, Cybersecurity Providing in Information and Telecommunication Systems (CPITS 2021), Kyiv, Ukraine, 28 January 2021*; RWTH Aachen University: Aachen, Germany, 2021; Volume 2923, pp. 47–59. Available online: http://ceur-ws.org/Vol-2923/ (accessed on 27 February 2023).
20. Bespalov, Y.; Kovalchuk, L.; Nelasa, H.; Oliynykov, R.; Garoffolo, A. Game theory analysis of incentive distribution for prompt generation of the proof tree in zk-SNARK based sidechains. In Proceedings of the 2022 IEEE International Carnahan Conference on Security Technology (ICCST), Valec u Hrotovic, Czech Republic, 7–9 September 2022; pp. 1–7. https://doi.org/10.1109/ICCST52959.2022.9896484.
21. Bonneau, J.; Meckler, I.; Rao, V.; Shapiro, E. Mina: Decentralized Cryptocurrency at Scale. 2020. Available online: https://minaprotocol.com/wp-content/uploads/technicalWhitepaper.pdf (accessed on 27 February 2023).
22. Cioabă, S.M.; Murty, M.R. *A First Course in Graph Theory and Combinatorics*; Texts and Readings in Mathematics 55; Cambridge University Press: Cambridge, UK, 2022. https://doi.org/10.1007/978-981-19-0957-3.
23. Bourbaki, N. *Algebra I. Chapters 1–3*; Springer-Verlag: Berlin/Heidelberg, Germany, 1998.
24. Reutenauer, C. *Free Lie Algebras*; London Mathematical Society Monographs New Series 7; Clarendon Press; Oxford University Press: Oxford, UK, 1993.
25. Stanley, R.P. *Catalan Numbers*; Cambridge University Press: Cambridge, UK, 2015. https://doi.org/10.1017/CBO9781139871495.
26. Johnson, N.L.; Kotz, S. *Urn Models and Their Applications*; John Wiley and Sons: New York, NY, USA, 1977.
27. O'Neill, B. The Classical Occupancy Distribution: Computation and Approximation. *Am. Stat.* **2021**, *75*, 364–375. https://doi.org/10.1080/00031305.2019.1699445.
28. Kanani, J.; Nailwal, S.; Arjun, A. Matic Whitepaper. 2020. Available online: https://github.com/maticnetwork/whitepaper (accessed on 27 February 2023).
29. Kuszmaul, J. Verkle Trees. In Proceedings of the Eighth Annual PRIMES Conference, 19–20 May 2018. Available online: https://math.mit.edu/research/highschool/primes/materials/2018/Kuszmaul.pdf (accessed on 27 February 2023).
30. Campanelli, M.; Hall-Andersen, M.; Kamp, S.H. Curve Trees: Practical and Transparent Zero-Knowledge Accumulators. Cryptology ePrint Archive, Report 2022/756. 2022. Available online: https://ia.cr/2022/756 (accessed on 27 February 2023).
31. Mac Lane, S. *Categories for the Working Mathematician*, 2nd ed.; Graduate Texts in Mathematics 5; Springer-Verlag: Berlin/Heidelberg, Germany, 1998.
32. Awodey, S. *Category Theory*, 2nd ed.; Oxforg Logic Guides 52; Oxford University Press: Oxford, UK, 2010.
33. Leinster, T. *Basic Category Theory*; Cambridge Studies in Advanced Mathematics 143; Cambridge University Press: Cambridge, UK, 2014.
34. Markl, M.; Shnider, S.; Stasheff, J.D. *Operads in Algebra, Topology and Physics*; Mathematical Surveys and Monographs 96; AMS: Providence, RI, USA, 2002. https://doi.org/10.1090/surv/096.
35. Méndez, M.A. *Set Operads in Combinatorics and Computer Science*; SpringerBriefs in Mathematics, Springer-Verlag: Berlin/Heidelberg, Germany, 2015. https://doi.org/10.1007/978-3-319-11713-3.
36. Spivak, D.I. *Category Theory for the Sciences*; MIT Press: Cambridge, MA, USA, 2014.