Roots in the semiring of finite deterministic dynamical systems

François Doré[1], Kévin Perrot[2], Antonio E. Porreca[2], Sara Riva[3], and Marius Rolland[2]

[1] Université Côte d'Azur, CNRS, I3S, France
[2] Aix-Marseille Université, CNRS, LIS, Marseille, France
[3] Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Abstract. Finite discrete-time dynamical systems (FDDS) model phenomena that evolve deterministically in discrete time. It is possible to define sum and product operations on these systems (disjoint union and direct product, respectively) giving a commutative semiring. This algebraic structure led to several works employing polynomial equations to model hypotheses on phenomena modelled using FDDS. To solve these equations, algorithms for performing the division and computing k-th roots are needed. In this paper, we propose two polynomial algorithms for these tasks, under the condition that the result is a connected FDDS. This ultimately leads to an efficient solution to equations of the type $AX^k = B$ for connected X. These results are some of the important final steps for solving more general polynomial equations on FDDS.

Keywords: discrete dynamical systems, root of graph direct product

1 Introduction

Finite discrete-time dynamical systems (FDDS) are pairs (X, f) where X is a finite set of states and $f : X \to X$ is a transition function (where no ambiguity arises, we will usually denote (X, f) simply as X). These systems emerge from the analysis of concrete models such as Boolean networks [10,11] and are applied to biology [17,16,1] to represent, for example, genetic regulatory networks or epidemic models. We can find them also in chemistry [7], to represent the evolution over discrete time of chemical reactions, or information theory [9].

We can identify dynamical systems with their transition graph, which have uniform outgoing degree one (these are also known as functional digraphs). Their general shape is a collection of cycles with a finite number of directed trees (with arcs pointing towards the root, *i.e.,* in-trees) anchored to them by the root. The nodes inside the cycles are periodic states, while the others are transient states.

The set $(\mathbb{D}, +, \times)$ of FDDS taken up to isomorphism with the disjoint union as a *sum* operation (corresponding to the alternative execution of two systems) and the *direct product* [12] (corresponding to synchronous execution) is a commutative semiring [2]. However, this semiring is not factorial, *i.e.,* a FFDS admits, in general, multiple factorizations into irreducibles. For this reason, the structure of product is more complex compared to other semirings such as the natural numbers, and its understanding remains limited. We are still unable to characterize or efficiently detect the FDDS obtained by parallel execution of smaller FDDS.

Some literature analyzes this problem limited to periodic behaviours, *i.e.,* to FDDS with permutations as their transition function [3,8,4]. Studying these restricted FDDS is justified by the fact that they correspond to the stable, asymptotic behaviour of the system. However, transient behaviour is more vast and various when modelling phenomena such as those from, for example, biology or physics. FDDS with a single fixed point have also been investigated [14] focusing more on the transient behaviours. Nevertheless, we cannot investigate general FDDS through a simple combination of these two techniques.

A direction for reducing the complexity of the decomposition problem is finding an efficient algorithm for equations of the form $AX = B$, *i.e.,* for dividing FDDS. The problem is trivially in NP, but we do not know its exact complexity (*e.g.*, NP-hard, GI, or P). However, [5] proved that we can solve these equations in polynomial time if A and B are certain classes of *permutations*, *i.e.,* FDDS without transient states. Nevertheless, the complexity of more general cases is unknown even for permutations.

Another direction is to propose an efficient algorithm for the computation of roots over FDDS. Since [14], we are aware of the uniqueness of the solution of k-th roots, but once again we do not know the exact complexity of the problem beyond a trivial NP upper bound.

In this paper, we will exploit the notion of unroll introduced in [14] to address the division and the root problems in the specific case where X is connected (*i.e.,* the graph of X contains just one connected component). More precisely, we start by showing that we can compute in polynomial time a FDDS X such that $AX = B$, if any exists. We also show that we can compute in polynomial time, given an FDDS A and a strictly positive integer k, a connected FDDS X such that $X^k = A$, if any exists. These two last contributions naturally lead to a solution to the more general equation $AX^k = B$.

2 Definitions

In the following, we will refer to the in-trees constituting the transient behaviour of FDDS just as *trees* for simplicity. An FDDS has a set of weakly connected components, each containing a unique cycle. In the following, we will refer to FDDS with only one component as *connected*.

In literature, two operations over FDDS have been considered: the *sum* (the disjoint union of the components of two systems) and the *product* (direct product [12] of their transition graphs). Let us recall that, given two digraphs $A = (V, E)$ and $B = (V', E')$, their product $A \times B$ is a digraph where the set of nodes is $V \times V'$ and the set of edges is $\{((v, v'), (u, u')) \mid (v, u) \in E, (v', u') \in E'\}$. When applied to the transition graphs of two connected FDDS with cycle lengths respectively p and p', this operation generates $\gcd(p, p')$ components with cycles of length $\mathrm{lcm}(p, p')$ [2,12].

Let us recall the notion of unroll of dynamical systems introduced in [14]. We will denote trees and forests using bold letters (in lower and upper case respectively) to distinguish them from FDDS.

Definition 1 (Unroll). *Let A be an FDDS (X, f). For each state $u \in X$ and $k \in \mathbb{N}$, we denote by $f^{-k}(u) = \{v \in X \mid f^k(v) = u\}$ the set of k-th preimages of u. For each u in a cycle of A, we call the* unroll tree of A in u *the infinite tree $\mathbf{t}_u = (V, E)$ having vertices $V = \{(s, k) \mid s \in f^{-k}(u), k \in \mathbb{N}\}$ and edges $E = \big\{\big((v, k), (f(v), k-1)\big)\big\} \subseteq V^2$. We call* unroll of A, *denoted $\mathcal{U}(A)$, the set of its unroll trees.*

Unroll trees have exactly one infinite branch on which the trees representing transient behaviour hook and repeat periodically. Remark that the forest given by the unroll of a connected FDDS may contain isomorphic trees and this results from symmetries in the original graph.

This transformation from an FDDS to its unroll has already proved successful in studying operations (particularly the product operation) at the level of transient behaviours. Indeed, the sum (disjoint union) of two unrolls corresponds to the unroll of the sum of the FDDS; formally, $\mathcal{U}(A) + \mathcal{U}(A') = \mathcal{U}(A + A')$. For the product, it has been shown that it is possible to define an equivalent product over unrolls for which $\mathcal{U}(A) \times \mathcal{U}(A') = \mathcal{U}(A \times A')$. Here and in the following, the equality sign will denote graph isomorphism.

Let us formally define the product of trees to be applied over the unroll of two FDDS. Since it is known that the product distributes over the different trees of the two unrolls [6], it suffices to define the product between two trees. Intuitively, this product is the direct product applied layer by layer. To define it, we let $depth(v)$ be the distance of the node from the root of the tree.

Definition 2 (Product of trees). *Consider two trees $\mathbf{t}_1 = (V_1, E_1)$ and $\mathbf{t}_2 = (V_2, E_2)$ with roots r_1 and r_2, respectively. Their* product *is the tree $\mathbf{t}_1 \times \mathbf{t}_2 = (V, E)$ such that $V = \{(v, u) \in V_1 \times V_2 \mid depth(v) = depth(u)\}$ and $E = \{((v, u), (v', u')) \mid (v, u) \in V, (v, v') \in E_1, (u, u') \in E_2\}$.*

In the following, we use a total order $\leq$ on finite trees introduced in [14], which is compatible with the product, that is, if $\mathbf{t}_1 \leq \mathbf{t}_2$ then $\mathbf{t}_1\mathbf{t} \leq \mathbf{t}_2\mathbf{t}$ for all tree $\mathbf{t}$. Let us briefly recall that this ordering is based on a vector obtained from concatenating the incoming degrees of nodes visited through a BFS. During graph traversal, child nodes (preimages in our case), are sorted recursively according to this very order, resulting in a deterministic computation of the vector.

We will also need the notion of *depth* for finite trees and forests. The depth of a finite tree is the length of its longest branch. For a forest, it is the maximum depth of its trees. In the case of unrolls, which have infinite paths, we can adopt the notion of depth of a dynamical system (that is, the largest depth among the trees rooted in one of its periodic states). For an unroll tree $\mathbf{t}$, its depth is the depth of a connected FDDS A such that $\mathbf{t} \in \mathcal{U}(A)$. See Figure 1.

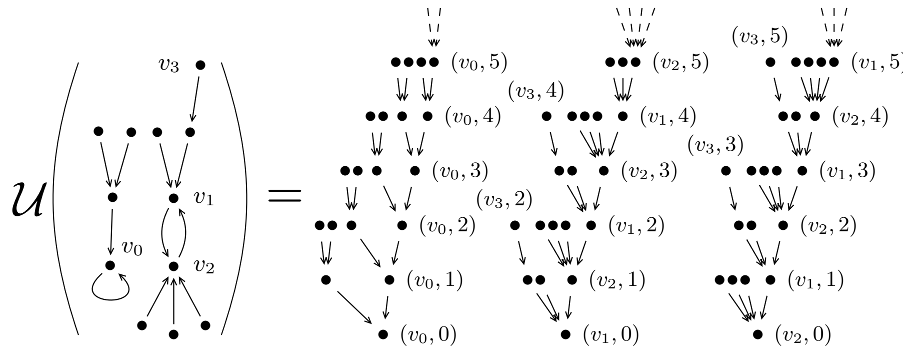

Fig. 1. The unroll $\mathcal{U}(A)$ of a disconnected FDDS A. Only the first 6 levels of $\mathcal{U}(A)$ are shown. Both the FDDS and its unroll have depth 2.

We now recall three operations defined in [14] that will be useful later. Given a forest $\mathbf{F}$, we denote by $\mathcal{D}(\mathbf{F})$ the multi-set of trees rooted in the predecessors of the roots of $\mathbf{F}$. Then, we denote by $\mathcal{R}(\mathbf{F})$ the tree such that $\mathcal{D}(\mathcal{R}(\mathbf{F})) = \mathbf{F}$. Intuitively, this second operation connects the trees to a new common root. Finally, given a positive integer k, we denote $\mathcal{C}(\mathbf{t}, k)$ the induced sub-tree of $\mathbf{t}$ composed by the vertices with a depth less or equal to k. Let us generalize the same operation applied to a forest $\mathbf{F} = \mathbf{t}_1 + ... + \mathbf{t}_n$ as $\mathcal{C}(\mathbf{F}, k) = \mathcal{C}(\mathbf{t}_1, k) + ... + \mathcal{C}(\mathbf{t}_n, k)$.

3 Complexity of FDDS division with connected quotient

In this section we establish an upper bound to the complexity of division over FDDS. More formally, our problem is to decide if, given two FDDS A and B, there exists a connected FDDS X such that $AX = B$. To achieve this, we will initially prove that cancellation holds over unrolls, *i.e.,* that $\mathbf{EX} = \mathbf{EY}$ implies $\mathbf{X} = \mathbf{Y}$ for unrolls $\mathbf{E}, \mathbf{X}, \mathbf{Y}$. Later, we will extend the algorithm proposed in [14, Figure 6] to handle more general unrolls (rather than just those consisting of a single tree), ultimately leading to our result.

We begin by considering the case of forests containing a finite number of finite trees; we will refer to them as *finite tree forests*. We will later generalise the reasoning to forests such as unrolls.

Lemma 1. *Let* $\mathbf{A}$*,* $\mathbf{X}$*, and* $\mathbf{B}$ *be finite tree forests. Then,* $\mathbf{AX} = \mathbf{B}$ *if and only if* $\mathcal{R}(\mathbf{A})\mathcal{R}(\mathbf{X}) = \mathcal{R}(\mathbf{B})$.

Proof. ($\Leftarrow$) Assume $\mathcal{R}(\mathbf{A})\mathcal{R}(\mathbf{X}) = \mathcal{R}(\mathbf{B})$. Then,$\mathcal{D}(\mathcal{R}(\mathbf{A})\mathcal{R}(\mathbf{X})) = \mathcal{D}(\mathcal{R}(\mathbf{B})) = \mathbf{B}$. Moreover, since $\mathcal{R}(\mathbf{A})$ and $\mathcal{R}(\mathbf{X})$ are finite trees, by [14, Lemma 7] we have:

$$\mathcal{D}(\mathcal{R}(\mathbf{A})\mathcal{R}(\mathbf{X})) = \mathcal{D}(\mathcal{R}(\mathbf{A}))\mathcal{D}(\mathcal{R}(\mathbf{X})) = \mathbf{AX}.$$

($\Rightarrow$) We can show the other direction by a similar reasoning. $\square$

Thanks to this lemma, we can generalise Lemma 21 of [14] as follows.

Lemma 2. *Let* $\mathbf{A}$*,* $\mathbf{X}$*, and* $\mathbf{Y}$ *be finite tree forests. Then* $\mathbf{AX} = \mathbf{AY}$ *if and only if* $\mathcal{C}(\mathbf{X}, \mathit{depth}(\mathbf{A})) = \mathcal{C}(\mathbf{Y}, \mathit{depth}(\mathbf{A}))$*.*

Proof. ($\Leftarrow$) By the definition of tree product, all nodes of $\mathbf{X}$ (resp., $\mathbf{Y}$) of depth larger than $\mathit{depth}(\mathbf{A})$ do not impact the product $\mathbf{AX}$ (resp., $\mathbf{AY}$). Thus, we have $\mathbf{AX} = \mathbf{A}\mathcal{C}(\mathbf{X}, \mathit{depth}(\mathbf{A}))$ and $\mathbf{AY} = \mathbf{A}\mathcal{C}(\mathbf{Y}, \mathit{depth}(\mathbf{A}))$. Since $\mathcal{C}(\mathbf{X}, \mathit{depth}(\mathbf{A})) = \mathcal{C}(\mathbf{Y}, \mathit{depth}(\mathbf{A}))$, we conclude that $\mathbf{AX} = \mathbf{AY}$.

($\Rightarrow$) Suppose $\mathbf{AX} = \mathbf{AY}$. By Lemma 1, we have $\mathcal{R}(\mathbf{A})\mathcal{R}(\mathbf{X}) = \mathcal{R}(\mathbf{A})\mathcal{R}(\mathbf{Y})$. Since $\mathcal{R}(\mathbf{A})$, $\mathcal{R}(\mathbf{X})$, and $\mathcal{R}(\mathbf{Y})$ are finite trees, we deduce [14, Lemma 21]

$$\mathcal{C}(\mathcal{R}(\mathbf{X}), \mathit{depth}(\mathcal{R}(\mathbf{A}))) = \mathcal{C}(\mathcal{R}(\mathbf{Y}), \mathit{depth}(\mathcal{R}(\mathbf{A}))) \tag{1}$$

For all forest $\mathbf{F}$ and $d > 0$, we have that $\mathcal{D}(\mathcal{C}(\mathbf{F}, d))$ is the multiset containing the subtrees rooted on the predecessors of the roots of $\mathcal{C}(\mathbf{F}, d)$. It is therefore the same multiset as that which is composed of the subtrees rooted on the predecessors of the roots of $\mathbf{F}$ cut at depth $d-1$. It follows that $\mathcal{C}(\mathcal{D}(\mathbf{F}), d-1) = \mathcal{D}(\mathcal{C}(\mathbf{F}, d))$. In particular, for $\mathbf{F} = \mathcal{R}(\mathbf{X})$ and $d = \mathit{depth}(\mathbf{A}) + 1 = \mathit{depth}(\mathcal{R}(\mathbf{A}))$, we have

$$\mathcal{D}(\mathcal{C}(\mathcal{R}(\mathbf{X}), \mathit{depth}(\mathcal{R}(\mathbf{A})))) = \mathcal{C}(\mathbf{X}, \mathit{depth}(\mathbf{A})).$$

Likewise, $\mathcal{D}(\mathcal{C}(\mathcal{R}(\mathbf{Y}), \mathit{depth}(\mathcal{R}(\mathbf{A})))) = \mathcal{C}(\mathbf{Y}, \mathit{depth}(\mathbf{A}))$. By applying $\mathcal{D}(\cdot)$ to both sides of (1), we conclude $\mathcal{C}(\mathbf{X}, \mathit{depth}(\mathbf{A})) = \mathcal{C}(\mathbf{Y}, \mathit{depth}(\mathbf{A}))$. □

Lemma 2 is a sort of cancellation property subject to a depth condition. The first step to prove cancellation over unrolls is proving the equivalence between the notion of divisibility of unrolls and divisibility over deep enough finite cuts.

Proposition 1. *Let* A*,* X*, and* B *be FDDS with* α *equal to the number of unroll trees of* $\mathcal{U}(B)$*. Let* $n \geq \alpha + \mathit{depth}(\mathcal{U}(B))$*. Then*

$$\mathcal{U}(A)\mathcal{U}(X) = \mathcal{U}(B) \text{ if and only if } \mathcal{C}(\mathcal{U}(A), n)\mathcal{C}(\mathcal{U}(X), n) = \mathcal{C}(\mathcal{U}(B), n)$$

To prove Proposition 1, we can apply the same reasoning of [14, Lemma 38] (see appendix for the details of the proof).

We remark that the cut operation over $\mathcal{U}(B)$ at a depth n generates a forest where the size of each tree is in $\mathcal{O}(m^2)$ and the total size is in $\mathcal{O}(m^3)$ with m the size of B (*i.e.,* the number of nodes), since the chosen n is at most m. Now, we can prove the main result of this section.

Theorem 1. *For unrolls* $\mathbf{A}$*,* $\mathbf{X}$*,* $\mathbf{Y}$ *we have* $\mathbf{AX} = \mathbf{AY}$ *if and only if* $\mathbf{X} = \mathbf{Y}$*.*

Proof. Let α be the number of trees in $\mathbf{AX}$ and $n \geq \alpha + \mathit{depth}(\mathbf{AX})$ be an integer. By Proposition 1, $\mathbf{AX} = \mathbf{AY}$ if and only if $\mathcal{C}(\mathbf{A}, n)\mathcal{C}(\mathbf{X}, n) = \mathcal{C}(\mathbf{A}, n)\mathcal{C}(\mathbf{Y}, n)$. In addition, by Lemma 2, $\mathcal{C}(\mathbf{A}, n)\mathcal{C}(\mathbf{X}, n) = \mathcal{C}(\mathbf{A}, n)\mathcal{C}(\mathbf{Y}, n)$ if and only if $\mathcal{C}(\mathbf{X}, n) = \mathcal{C}(\mathbf{Y}, n)$. By Proposition 1, the theorem follows. □

Let us introduce the notion of *periodic pattern* of an unroll tree. Recall that an unroll tree $\mathbf{t}$ has exactly one infinite branch on which the trees $(\mathbf{t}_0, \mathbf{t}_1, \ldots)$ representing transient behaviour hook and repeat periodically. Let p be a positive integer. A periodic pattern with period p of $\mathbf{t}$ is a sequence of p finite trees $(\mathbf{t}_0, \ldots, \mathbf{t}_{p-1})$ rooted on the infinite branch such that, for all $i \in \mathbb{N}$ we have $\mathbf{t}_i = \mathbf{t}_{i \bmod p}$. Let us point out that the idea here is to obtain a set of trees such that we represent all different behaviours repeating in all unroll trees, obtaining a finite representation.

For connected FDDS, since its period p is the number of trees in its unroll, we can reconstruct the FDDS itself from a periodic pattern $(\mathbf{t}_0, \ldots, \mathbf{t}_{p-1})$ of one of its unroll trees $\mathbf{t}_u$ by adding edges between $\mathbf{t}_i$ and $\mathbf{t}_{(i+1) \bmod p}$ for all i. We call this operation the *roll of* $\mathbf{t}_u$ *of period* p. The following lemma shows that we can recover the periodic pattern of an unroll tree from a deep enough cut.

Lemma 3. *Let A be a connected FDDS of period p, $\mathbf{t}$ be an unroll tree of $\mathcal{U}(A)$, and $n \geq p + depth(\mathcal{U}(A))$. Let $(v_n, \ldots, v_0)$ be a directed path in $\mathcal{C}(\mathbf{t}, n)$ such that $depth(v_n) = n$ and v_0 is the root of the tree. Then, nodes $v_p, \ldots, v_0$ necessarily come from the infinite branch of $\mathbf{t}$.*

Proof. We assume, by contradiction, that at least one of the nodes $v_p, \ldots, v_0$ does not come from the infinite branch of $\mathbf{t}$. Let v_a be the node of $(v_{p-1}, \ldots, v_0)$ with maximal depth coming from the infinite branch of $\mathbf{t}$; there always is at least one of them, namely the root v_0. We have $depth(v_n) \leq depth(v_a) + depth(\mathbf{t})$. However, we assumed $depth(v_a) < p$, thus $depth(v_n) < p + depth(\mathbf{t})$. Since, $depth(v_n) = n$, we have $n < p + depth(\mathbf{t}) = p + depth(\mathcal{U}(A))$ which is a contradiction. □

We can finally describe a division algorithm for FDDS working under the hypothesis that the quotient is connected.

Algorithm 1. *Given two FDDS A and B, where $\mathcal{U}(B)$ has α trees, we can compute X such that X is a connected FDDS and $AX = B$ (if any exists) by*

1. *cutting $\mathcal{U}(A)$ and $\mathcal{U}(B)$ at depth $n = \alpha + depth(\mathcal{U}(B))$*
2. *computing $\mathbf{x}$ with the division algorithm [14] to divide the trees $\mathcal{R}(\mathcal{C}(\mathcal{U}(B), n))$ by $\mathcal{R}(\mathcal{C}(\mathcal{U}(A), n))$*
3. *computing the connected FDDS X as the roll of period p of any tree of $\mathcal{D}(\mathbf{x})$, where p is equal to the number of trees in $\mathcal{D}(\mathbf{x})$*
4. *and verifying if X multiplied by A is isomorphic to B.*

Since the depth where we cut is large enough, Proposition 1, Lemma 1 and the correctness of the division algorithm of [14] imply that the tree $\mathbf{x}$ computed in Step 2 of Algorithm 1 satisfies $\mathcal{C}(\mathcal{U}(A), n)\mathcal{D}(\mathbf{x}) = \mathcal{C}(\mathcal{U}(B), n)$. By the definition of unroll, since we only search for connected FDDS, if $\mathcal{D}(\mathbf{x})$ is the cut of an unroll then the rolls of each tree of $\mathcal{D}(\mathbf{x})$ at period p are isomorphic. Furthermore, Lemma 3 ensures that we can roll each tree in $\mathcal{D}(\mathbf{x})$. However, $\mathcal{D}(\mathbf{x})$ is not necessarily the cut of an unroll and it is possible that there exists an FDDS X such that $\mathcal{D}(\mathbf{x}) = \mathcal{C}(\mathcal{U}(X), n)$ but $AX \neq B$ (an example can be seen in Figure 2). As a consequence, Step 4 of Algorithm 1 is mandatory to ensure its correctness.

Theorem 2. *Algorithm 1 runs in $\mathcal{O}(m^9)$ time, where m is the size of its inputs.*

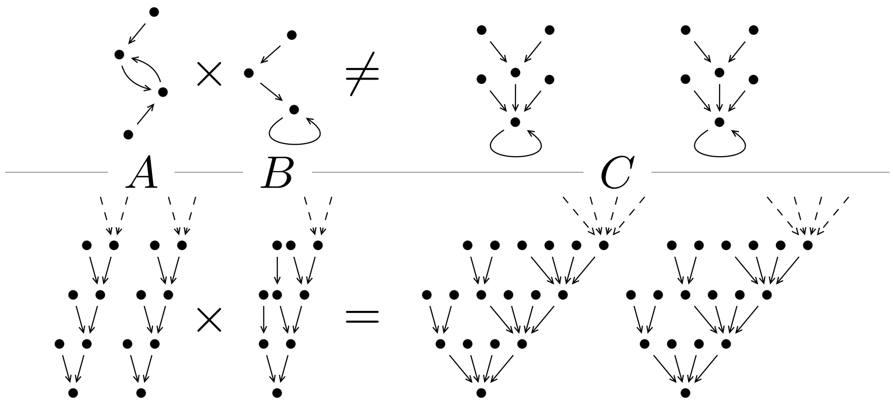

Fig. 2. Three FDDSs A, B and C such that $AB \neq C$ but $\mathcal{U}(A)\mathcal{U}(B) = \mathcal{U}(C)$. Here the symbol $\times$ denotes the product of FDDSs on the top, and of forests on the bottom.

Proof. The cuts of depth n of the unrolls of A and B can be computed in $\mathcal{O}(m^3)$ time and the size of the result is $\mathcal{O}(m^3)$. In fact, we can construct $\mathcal{C}(\mathcal{U}(A), n)$ and $\mathcal{C}(\mathcal{U}(B), n)$ backwards from their roots up to depth n; the size of $\mathcal{C}(\mathcal{U}(A), n)$ is bounded by the size of $\mathcal{C}(\mathcal{U}(B), n)$, which is $\mathcal{O}(m^3)$. By analysing the division tree algorithm in Figure 6 of [14], we can check that it can be executed in cubic time. Moreover, since its inputs have size $\mathcal{O}(m^3)$, step 2 of Algorithm 1 requires $\mathcal{O}(m^9)$ time. The roll procedure of a tree can be computed by a traversal, requiring $\mathcal{O}(m^2)$ time. Finally, the product of two FDDS is quadratic-time on its input but linear-time on its output. However, in our case, the size of the output of AX is bounded by the size of B; hence, the product can be computed in $\mathcal{O}(m)$ time. Finally, the isomorphism test requires $\mathcal{O}(m)$ [13]. $\square$

4 Complexity of computing k-th roots of unrolls

The purpose of this section is to study the problem of computing connected roots on FDDS, particularly on transients. Let $\mathbf{A} = \mathbf{t}_1 + \ldots + \mathbf{t}_n$ be a forest and k a positive integer. Then $\mathbf{A}^k = \sum_{k_1+\ldots+k_n=k} \binom{k}{k_1,\ldots,k_n} \prod_{i=1}^{n} \mathbf{t}_i^{k_i}$; furthermore, since the sum of forests is their disjoint union, each forest (in particular $\mathbf{A}^k$) can be written as a sum of trees in a unique way (up to reordering of the terms). The injectivity of k-th roots, in the semiring of unrolls, has been proved in [14]. Here, we study this problem from an algorithmic and complexity point of view, and find a polynomial-time upper bound for the computation of k-th roots.

We begin by studying the structure of a forest of finite trees raised to the k-th power. Indeed, if we suppose $\mathbf{X} = \mathbf{t}_1 + \ldots + \mathbf{t}_n$ with $\mathbf{t}_i \leq \mathbf{t}_{i+1}$, we want to be able to identify the smallest tree of $\mathbf{X}^k$ from the product $\mathbf{t}_i \times \prod_{j=1}^{n} \mathbf{t}_j$. Moreover, we want to be able to identify it for all $\mathbf{t}_i$.

Hereafter, we consider $\mathbf{a}^0$ to be equivalent to the simple oriented path with length equivalent to the depth of $\mathbf{a}$ (the same is true for forests).

Lemma 4. *Let* $\mathbf{X}$ *be a forest of the form* $\mathbf{X} = \mathbf{t}_1 + \ldots + \mathbf{t}_n$ *(with* $\mathbf{t}_i \leq \mathbf{t}_{i+1}$*) and* k *a positive integer. For any tree* $\mathbf{t}_i$ *of depth* d_i *in* $\mathbf{X}$*, the smallest tree* $\mathbf{t}_s$ *of depth* d_i *with factor* $\mathbf{t}_i$ *in* $\mathbf{X}^k$ *is isomorphic to* $\mathbf{t}_m^{k-1}\mathbf{t}_i$*, where* $\mathbf{t}_m$ *is the smallest tree of* $\mathbf{X}$ *with depth at least* d_i*.*

Proof. Let us assume that the smallest tree $\mathbf{t}_s$ of depth d_i with factor $\mathbf{t}_i$ in $\mathbf{X}^k$ is not isomorphic to $\mathbf{t}_m^{k-1}\mathbf{t}_i$. Two cases are possible. Either $\mathbf{t}_s$ contains a third factor other than $\mathbf{t}_m$ and $\mathbf{t}_i$, or it is of the form $\mathbf{t}_m^{k-k_i}\mathbf{t}_i^{k_i}$, with $k_i > 1$.

In the former case, let us suppose that there exists $a \in \{1, \ldots, i-1\} \setminus \{m\}$ and $k_a > 0$ such that $\mathbf{t}_a \neq \mathbf{t}_m$ and $\mathbf{t}_s$ is isomorphic to $\mathbf{t}_i^{k_i}\mathbf{t}_a^{k_a}\mathbf{t}_m^{k_m}$. Remark that, according to [14, Lemma 10], the smallest tree of depth d_i with factor $\mathbf{t}_i$ in $\mathbf{X}^k$ necessarily has all its factors of depth at least d_i. For this reason, we can assume $depth(\mathbf{t}_a) \geq d_i$ without loss of generality. However, since $\mathbf{t}_m < \mathbf{t}_a$, we have $\mathbf{t}_m^{k_m+1}\mathbf{t}_a^{k_a-1} < \mathbf{t}_m^{k_m}\mathbf{t}_a^{k_a}$. Thus, we have that $\mathbf{t}_i^{k_i}\mathbf{t}_m^{k_m+1}\mathbf{t}_a^{k_a-1} < \mathbf{t}_i^{k_i}\mathbf{t}_m^{k_m}\mathbf{t}_a^{k_a}$. This brings us into contradiction with the minimality of $\mathbf{t}_s$.

In the second case, we assume that $\mathbf{t}_s$ is isomorphic to $\mathbf{t}_m^{k-k_i}\mathbf{t}_i^{k_i}$ with $k_i > 1$. By hypothesis, we have $\mathbf{t}_i \geq \mathbf{t}_m$. If we consider the case of $\mathbf{t}_m < \mathbf{t}_i$, we have $\mathbf{t}_m^{k-k_i+1}\mathbf{t}_i^{k_i-1} < \mathbf{t}_m^{k-k_i}\mathbf{t}_i^{k_i}$. Once again, this is in contradiction with the minimality of $\mathbf{t}_s$. In the case of $\mathbf{t}_m = \mathbf{t}_i$, we have $\mathbf{t}_m^{k-k_i+1}\mathbf{t}_i^{k_i-1} = \mathbf{t}_m^{k-k_i}\mathbf{t}_i^{k_i}$. But we supposed $\mathbf{t}_s$ not isomorphic to $\mathbf{t}_m^{k-1}\mathbf{t}_i$. This concludes the proof. $\square$

Before describing an algorithmic technique for computing roots over unrolls (*i.e.,* forests), we need a last technical lemma.

Lemma 5. *Let* $\mathbf{x}$ *and* $\mathbf{a}$ *be two finite trees such that* $\mathbf{x}^k = \mathbf{a}$*, and* k *a positive integer. Then,* $\mathcal{D}(\mathbf{a}) = \mathcal{D}(\mathbf{x})^k$*.*

Proof. Since $\mathbf{x}$ is a tree, for all $i \leq k$, $\mathbf{x}^i$ is also a tree. According to [14, Lemma 7], we have $\mathcal{D}(\mathbf{a}) = \mathcal{D}(\mathbf{x}^k) = \mathcal{D}(\mathbf{x})^k$. $\square$

We now introduce an algorithmic procedure to compute the roots over forests based on an induction over decreasing depths in which, each time, we reconstruct part of the solution considering the smallest tree with at least a specific depth (according to Lemmas 4 and 5).

Theorem 3. *Given a forest* $\mathbf{A}$ *and* k *a strictly positive integer, we can compute* $\mathbf{X}$ *such that* $\mathbf{X}^k = \mathbf{A}$ *with Algorithm 2.*

In Algorithm 2, the main idea is to extract iteratively the minimal tree among the tallest ones in $\mathbf{A}$ (*i.e.,* $\mathbf{t}_s$). This tree will be used to reconstruct one of the trees of $\mathbf{X}$ (*i.e.,* $\mathbf{t}_i$). This can be done in two ways according to two possible scenarios. In the first case, $\mathbf{t}_s$ is smaller than the smallest one already reconstructed (*i.e.,* $\mathbf{t}_m$) raised to the power k. If this is the case, we compute a new tree in $\mathbf{X}$ through a recursive call to our `root` function. In the second case, the extracted tree is greater than $\mathbf{t}_m^k$. This means, by Lemma 4, that it is a product of the smallest reconstructed (*i.e.,* $\mathbf{t}_m$) one and a new one (*i.e.,* $\mathbf{t}_i$). In this case, the latter can be computed by the `divide` algorithm of [14]. After the reconstruction of a tree $\mathbf{t}_i$ of $\mathbf{X}$, we remove from $\mathbf{A}$ all the trees obtainable from products of already

Algorithm 2 `root`

Require: $\mathbf{A}$ a forest, k an integer

```
if A is a path then
    return A
end if
R ← ∅
t_m ← ∅
F ← A
while F ≠ ∅ do
    F ← A \ R^k.
    t_s ← min{t | t ∈ F, depth(t) = depth(F)}
    if t_m = ∅ or t_m^k > t_s then
        t_i ← R(root(D(t_s), k))
        t_m ← t_i
    else
        t_i ← divide(t_s, t_m^{k-1})
    end if
    if t_i = ⊥ or (R + t_i)^k ⊈ A then
        return ⊥
    end if
    R ← R + t_i
end while
return R
```

computed trees in $\mathbf{X}$. This allows us to extract progressively shorter trees $\mathbf{t}_s$ from $\mathbf{A}$ and to compute consequently shorter trees of $\mathbf{X}$. When we remove all trees in $\mathbf{A}$ obtainable from trees $\mathbf{t}_i$ with depth at least d_i in $\mathbf{X}$, this leaves us only trees with depth at most d_i. Since for each depth, the number of trees of this depth is finite, the algorithm necessarily halts.

Let us consider an example. In Figure 3, in order to compute the left side from the right one, the first tree considered is $\mathbf{t}_1^2$, the single tallest one. The latter can be used to compute $\mathbf{t}_1$ recursively. Next, the smallest one among the remaining ones is $\mathbf{t}_0^2$, which is smaller than $\mathbf{t}_1^2$. Thus, we can compute $\mathbf{t}_0$ again through recursion. Finally, the last tree extracted, after removing the trees with exclusively $\mathbf{t}_0$ and $\mathbf{t}_1$ as factors, is $\mathbf{t}_0\mathbf{t}_2$. Since this time $\mathbf{t}_0^2$ is smaller, we can get the third and final tree $\mathbf{t}_2$ by dividing it by the smallest computed tree yet.

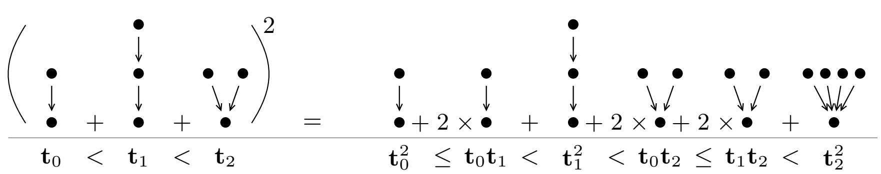

Fig. 3. Order of the trees in the square of a forest.

Theorem 4. *Algorithm 2 runs in $\mathcal{O}(m^3)$ time if k is at most $\lfloor \log_2 m \rfloor$, where m is the size of $\mathbf{A}$.*

Proof. If $\mathbf{A}$ is a path, then the algorithm halts in linear time $\mathcal{O}(m)$ on line 2.

Otherwise, there exists a level i of $\mathbf{A}$ containing $\beta \geq 2$ nodes. In order to justify the upper bound on k, suppose $\mathbf{R}^k = \mathbf{A}$. Then, level i of $\mathbf{R}$ contains $\sqrt[k]{\beta}$ nodes. The smallest integer greater than 1 having a k-th root is 2^k, thus $\beta \geq 2^k$. Since $\beta \leq m$, we have $k \leq \lfloor \log_2 m \rfloor$.

Lines 1–6 take linear time $\mathcal{O}(m)$. The while loop of lines 7–20 is executed, in the worst case, once per tree of the k-th root $\mathbf{R}$, *i.e.,* a number of times equal to the k-th root of the number of trees in $\mathbf{A}$. Line 7 takes $\mathcal{O}(1)$ time. The product of trees requires linear time in its output size. Consequently, $\mathbf{R}^k$ can be computed in $\mathcal{O}(m \log m)$ time. Moreover, since we can remove $\mathbf{R}^k$ from $\mathbf{A}$ in quadratic time, we deduce that line 8 takes $\mathcal{O}(m^2)$ time. Since the search of $\mathbf{t}_s$ consists of a simple traversal, we deduce that line 9 takes $\mathcal{O}(m)$ time. Line 10 takes $\mathcal{O}(m \log m)$ time for computing $\mathbf{t}_m^k$. If no recursive call is made, line 14 is executed in time $\mathcal{O}(m_s^3)$, where m_s is the size of $\mathbf{t}_s$. The runtime of lines 16–20 is dominated by line 16, which takes $\mathcal{O}(m^2)$ as line 8.

Since each tree $\mathbf{t}_s$ in $\mathbf{A}$ is used at most once, we have $\sum m_s^3 \leq m^3$; as a consequence, the most expensive lines of the algorithm (namely, 8, 14, and 16) have a total runtime of $\mathcal{O}(m^3)$ across all iterations of the while loop.

We still need to take into account the recursive calls of line 11. By taking once again into account the bound $\sum m_s^3 \leq m^3$, the total runtime of these recursive calls is also $\mathcal{O}(m^3)$. We conclude that Algorithm 2 runs in time $\mathcal{O}(m^3)$. □

Corollary 1. *Let $\mathbf{A}$ be a forest. Then, it is possible to decide in polynomial time if there exists a forest $\mathbf{X}$ and an integer $k > 1$ such that $\mathbf{X}^k = \mathbf{A}$.*

Proof. Since k is bounded by the logarithm of the size of $\mathbf{A}$ and, according to Theorem 4, we can compute the root in $\mathcal{O}(m^3)$, we can test all k (up to the bound) and check if there exists a $\mathbf{X}$ such that $\mathbf{X}^k = \mathbf{A}$ in $\mathcal{O}(m^3 \log m)$, where m denotes once again the size of $\mathbf{A}$. □

According to Corollary 1, we easily conclude that the corresponding enumeration problem of finding all solutions $\mathbf{X}$ (for all powers k) is in EnumP, since the verification of a solution can be done in polynomial time and the size of a solution is polynomial in the size of the input. Moreover, the problem is in the class DelayP since the time elapsed between the computation of one solution (for a certain k) and the next is polynomial. We refer the reader to [15] for more information about enumeration complexity classes.

Now that we have a technique to compute the root of forests, let us think in terms of unrolls of FDDS. Consider an FDDS A and its unroll $\mathbf{A} = \mathcal{U}(A)$. According to Proposition 1, we can compute the FDDS X such that $A = X^k$ by considering the forest $\mathbf{F} = \mathcal{C}(\mathbf{A}, n)$ where α is the number of trees in $\mathbf{F}$ and $n = \alpha + \mathit{depth}(\mathbf{A})$. Again, this depth allows us to ensure that all the transient dynamics of the dynamical system are represented in the different trees. Applying the root algorithm on $\mathbf{F}$, we obtain the result of the root as a forest of finite trees. However, this is just a candidate solution for the corresponding problem over the initial FDDS (for the same reasoning as in the case of division). In order to test the result of the root algorithm, as before, we realised the roll of one tree in the solution to period p with p as the number of trees in the result. Then to decide if X is truly the k-th root of A, we verify if $X^k = A$ where X is the result of the roll operation. That is possible because the algorithm is designed to study connected solutions. Indeed, the following holds.

Corollary 2. *Let A be a FDDS, it is possible to decide if there exists a connected FDDS X and an integer $k > 1$ such that $X^k = A$ in polynomial time.*

By combining the division algorithm with the root algorithm, we are now able to study equations of the form $AX^k = B$. Given FDDS A and B and $k > 0$, we can first compute the result $\mathbf{Y}$ of the division of $\mathcal{C}(\mathcal{U}(B), n)$ by $\mathcal{C}(\mathcal{U}(A), n)$, where α is the number of trees in $\mathcal{U}(B)$ and $n = \alpha + depth(B)$. Then, we compute the k-th root $\mathbf{X}$ of $\mathbf{Y}$. After that, we make the roll of one tree of $\mathbf{X}$ in period p, with p the number of trees in $\mathbf{X}$. Then, using the roll result X, we just need to verify if $AX^k = B$. Once again, the solutions found by this method are only the connected ones, and further non-connected solutions are also possible.

5 Conclusions

In this article we have proven the cancellation property for products of unrolls and established that the division of FDDSs is polynomial-time when searching for connected quotients only. Furthermore, we have proven that calculating the k-th root of a FDDSs is polynomial-time if the solution is connected. Finally, we have shown that solving equations of the form $AX^k = B$ is polynomial if X is connected. However, numerous questions remain unanswered.

The main direction for further investigation involves removing the connectivity condition. Although the cancellation property of unrolls we proved and the new polynomial-time algorithm for the division suggest that the primary challenge for FDDS division lies in the cycles rather than the transients, the same cannot be said for the computation of the k-th root of FDDS. Another intriguing direction is solving general polynomial equations $P(X_1, \dots, X_n) = B$ with a constant right-hand side B. While this appears to be at least as challenging as division, some specific cases, such as when the polynomial is injective, could yield more direct results. Furthermore, the results of this work can improve the state of the art of the solution of $P(X_1, \dots, X_n) = B$ where polynomial P is a sum of univariate monomials [4]. Indeed, a technique to solve (and enumerate the solutions) of this type of equation in a finite number of systems of equations of the form $AX^k = B$ has been introduced. Thus, our result, which is more efficient than previously known techniques, can have a positive impact on the complexity of the proposed pipeline. It would also be interesting to investigate whether our techniques also apply to finding nontrivial solutions to equations of the form $XY = B$ with X and Y connected, which would make it possible to improve our knowledge of the problem of irreducibility.

Acknowledgments. SR was supported by the French Agence Nationale pour la Recherche (ANR) in the scope of the project "REBON" (grant number ANR-23-CE45-0008), and KP, AEP and MR by the EU project MSCA-SE-101131549 "ACANCOS".

References

1. Bernot, G., Comet, J.P., Richard, A., Chaves, M., Gouzé, J.L., Dayan, F.: Modeling in computational biology and biomedicine. In: Modeling and Analysis of Gene Regulatory Networks: A Multidisciplinary Endeavor, pp. 47–80. Springer (2012)
2. Dennunzio, A., Dorigatti, V., Formenti, E., Manzoni, L., Porreca, A.E.: Polynomial equations over finite, discrete-time dynamical systems. In: Cellular Automata, 13th International Conference on Cellular Automata for Research and Industry, ACRI 2018. Lecture Notes in Computer Science, vol. 11115, pp. 298–306. Springer (2018)
3. Dennunzio, A., Formenti, E., Margara, L., Montmirail, V., Riva, S.: Solving equations on discrete dynamical systems. In: Computational Intelligence Methods for Bioinformatics and Biostatistics, 16th International Meeting, CIBB 2019. Lecture Notes in Computer Science, vol. 12313, pp. 119–132. Springer (2019)
4. Dennunzio, A., Formenti, E., Margara, L., Riva, S.: An algorithmic pipeline for solving equations over discrete dynamical systems modelling hypothesis on real phenomena. Journal of Computational Science **66**, 101932 (2023)
5. Dennunzio, A., Formenti, E., Margara, L., Riva, S.: A note on solving basic equations over the semiring of functional digraphs. arXiv e-prints (2024), `https://arxiv.org/abs/2402.16923`
6. Doré, F., Formenti, E., Porreca, A.E., Riva, S.: Decomposition and factorisation of transients in functional graphs. Theoretical Computer Science **999**, 114514 (2024)
7. Ehrenfeucht, A., Rozenberg, G.: Reaction systems. Fundamenta Informaticae **75**, 263–280 (2007)
8. Formenti, E., Régin, J.C., Riva, S.: MDDs boost equation solving on discrete dynamical systems. In: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 196–213. Springer (2021)
9. Gadouleau, M., Riis, S.: Graph-theoretical constructions for graph entropy and network coding based communications. IEEE Transactions on Information Theory **57**(10), 6703–6717 (2011)
10. Gershenson, C.: Introduction to random boolean networks. arXiv e-prints (2004), `https://doi.org/10.48550/arXiv.nlin/0408006`
11. Goles, E., Martìnez, S.: Neural and Automata Networks: Dynamical Behavior and Applications. Kluwer Academic Publishers (1990)
12. Hammack, R., Imrich, W., Klavžar, S.: Handbook of Product Graphs. Discrete Mathematics and Its Applications, CRC Press, second edn. (2011)
13. Hopcroft, J.E., Wong, J.K.: Linear time algorithm for isomorphism of planar graphs (preliminary report). In: Proceedings of the sixth annual ACM symposium on Theory of computing. pp. 172–184 (1974)
14. Naquin, E., Gadouleau, M.: Factorisation in the semiring of finite dynamical systems. Theoretical Computer Science **998**, 114509 (2024)
15. Strozecki, Y.: Enumeration Complexity: Incremental Time, Delay and Space (2021), HDR thesis, Université de Versailles Saint-Quentin-en-Yvelines
16. Thomas, R.: Boolean formalization of genetic control circuits. Journal of Theoretical Biology **42**(3), 563–585 (1973)
17. Thomas, R., D'Ari, R.: Biological Feedback. CRC Press (1990)

Appendix

Proposition 1 *Let A, X, and B be FDDS with α equal to the number of unroll trees of $\mathcal{U}(B)$. Let $n \geq \alpha + depth(\mathcal{U}(B))$. Then*

$$\mathcal{U}(A)\mathcal{U}(X) = \mathcal{U}(B) \text{ if and only if } \mathcal{C}(\mathcal{U}(A), n)\mathcal{C}(\mathcal{U}(X), n) = \mathcal{C}(\mathcal{U}(B), n).$$

Proof. ($\Rightarrow$) If $\mathcal{U}(A)\mathcal{U}(X) = \mathcal{U}(B)$ then $\mathcal{C}(\mathcal{U}(A)\mathcal{U}(X), n) = \mathcal{C}(\mathcal{U}(B), n)$ for all $n \geq 0$. And since, $\mathcal{C}(\mathcal{U}(A)\mathcal{U}(X), n) = \mathcal{C}(\mathcal{U}(A), n)\mathcal{C}(\mathcal{U}(X), n)$, one direction follow.

($\Leftarrow$) For the other direction, we employ the same logic as in the proof of the Lemma 38 of [14] *i.e.,* extending an isomorphism of the unrolls cut to depth n to an isomorphism of the whole unrolls without cuts.

For this proof, we partially change the unrolls definition; more precisely, we change the set of nodes in each unroll tree. Indeed, we need to explicitly set (in the second coordinate) the root of each tree while in the former definition, the root is left implicit. Thus, as in the original definition, for each periodic state u we define an unroll tree $\mathbf{t}_u = (V, E)$ as having vertices $V = \{(s, u, k) \mid s \in f^{-k}(u), k \in \mathbb{N}\}$ and edges $E = \big\{\big((v, u, k), (f(v), u, k-1)\big)\big\} \subseteq V^2$ with f the transition function of the dynamical system. Remark that this produces an unroll tree having root $(u, u, 0)$. Let $\psi : V(\mathcal{C}(\mathcal{U}(B), n)) \to V(\mathcal{C}(\mathcal{U}(A), n)\mathcal{C}(\mathcal{U}(X), n))$ be a forest product isomorphism for the product $\mathcal{C}(\mathcal{U}(A), n)\mathcal{C}(\mathcal{U}(X), n) = \mathcal{C}(\mathcal{U}(B), n)$. Let $d : V(B)^2 \to \mathbb{N} \cup \{-1\}$ be the function associating each pair (u, v) to the length of the shortest directed path from u to v, if it exists in B, otherwise -1. We call D the maximum value $d(u, v)$ with $(u, v) \in V(B)^2 \cup V(A)^2 \cup V(X)^2$. Let us point out that $n > D$.

We extend ψ to $\phi : V(\mathcal{U}(B)) \to V(\mathcal{U}(A)\mathcal{U}(X))$ such that, for all $(b, r, h) \in V(\mathcal{U}(B))$ where $h > n$, we have $\phi(b, r, h) = ((a, r_1, h), (x, r_2, h))$ if and only if $\psi(b, r, d) = ((a, r_1, d), (x, r_2, d))$ where

$$d = \max(d(b, r), d((a, x), (r_1, r_2))) = \max(d(b, r), d(a, r_1), d(x, r_2))$$

where (a, x) and (r_1, r_2) are states of the FDDS AX. Remark that ϕ is a well-defined function, since (b, r, d) belongs to the domain of ψ, as $d \leq D < n$.

Now we prove that ϕ is a valid forest product isomorphism. First, we show the bijectivity of ϕ. The surjectivity of ϕ is an immediate consequence of the surjectivity of ψ. As for its injectivity, suppose that $\phi(b, r, h) = \phi(b', r', h')$. We denote $\phi(b, r, h) = ((a, r_1, h), (x, r_2, h))$ and $\phi(b', r', h') = ((a', r'_1, h'), (x', r'_2, h'))$. Thus $(a, x, r_1, r_2, h) = (a', x', r'_1, r'_2, h')$.

By the definition of ϕ, we have

$$\psi(b, r, d) = ((a, r_1, d), (x, r_2, d))$$

and

$$\begin{aligned}\psi(b', r', d') &= ((a', r'_1, d'), (x', r'_2, d')) \\ &= ((a, r_1, d'), (x, r_2, d')).\end{aligned}$$

By proving that $d = d'$, we obtain $\psi(b, r, d) = \psi(b', r', d')$ and, by injectivity of ψ, we deduce $(b, r, d') = (b', r', d')$ and, in particular, $b = b'$ and $r = r'$; since we already know that $h = h'$, the injectivity of ϕ follows.

Since ψ is a forest product isomorphism, we deduce that (b, r, d) and (b', r', d') are two nodes of the same tree. Indeed, the two nodes $((a, r_1, d), (x, r_2, d))$ and $((a, r_1, d'), (x, r_2, d'))$ belong to the same tree since they have the same root coordinate. Thus, we deduce that $r = r'$.

Moreover, since ψ is a forest product isomorphism, the distance between $((a, r_1, d), (x, r_2, d))$ and infinite branch of its tree (cut to depth n) equals the distance between (b, r, d) and the infinite branch of its tree (cut to depth n). And since this distance is the depth of node (a, x) in AX and b in B, we deduce that $depth_{AX}((a, x)) = depth_B(b)$. For the same reason, $depth_{AX}((a, x)) = depth_B(b')$. So $depth_B(b) = depth_B(b')$. Besides, by the definition of unroll, h is the depth of (b, r, h) in the unroll tree, and we deduce that (b, r, h) and (b', r, h) have the same depth. This implies that $d(b, r) = d(b', r')$. Hence $d = d'$ and, as a consequence, the injectivity of ϕ follows.

Now, we show that $\phi(b, r, h)$ is a root if and only if (b, r, h) is a root. Since ψ is a forest product isomorphism, we have (a, r_1, d) and (x, r_2, d) are roots if and only if (b, r, d) is a root. In addition, the depth of any root is 0, so $d = 0$. So, we conclude that $\phi(b, r, h)$ is a root if and only if $h = 0$ and (b, r, h) is a root.

Finally, we need to show that for all

$$((a, r_1, h), (x, r_2, h)) \ , \ ((a', r'_1, h'), (x', r'_2, h')) \in V(\mathcal{U}(A)\mathcal{U}(X))$$

we have

$$\Big(\phi^{-1}((a, r_1, h), (x, r_2, h)) \ , \ \phi^{-1}((a', r'_1, h'), (x', r'_2, h'))\Big) \in E(\mathcal{U}(B))$$

if and only if

$$\Big((a, r_1, h) \ , \ (a', r'_1, h')\Big) \in E(\mathcal{U}(A)) \text{ and } \Big((x, r_2, h) \ , \ (x', r'_2, h')\Big) \in E(\mathcal{U}(X)).$$

Since ψ is a forest product isomorphism, we have $\big((b, r, d), (b', r', d')\big) \in E(\mathcal{C}(\mathcal{U}(B), n))$ if and only if

$$\Big(((a, r_1, d), (x, r_2, d)) \ , \ ((a', r'_1, d'), (x', r'_2, d'))\Big) \in E(\mathcal{C}(\mathcal{U}(A), n)\mathcal{C}(\mathcal{U}(X), n)).$$

So, by the definition of ϕ that is, if and only if

$$\Big(((a, r_1, h), (x, r_2, h)) \ , \ ((a', r'_1, h'), (x', r'_2, h'))\Big) \in E(\mathcal{U}(A)\mathcal{U}(X))$$

which is equivalent to $\big((a, r_1, h), (a', r'_1, h')\big) \in E(\mathcal{U}(A))$ and $\big((x, r_2, h), (x', r'_2, h')\big) \in E(\mathcal{U}(X))$ by the Definition 2 of tree product. This proves that $\mathcal{U}(B) = \mathcal{U}(A)\mathcal{U}(X)$. □