$\mathsf{FO}^2(<,+1,\sim)$ on data trees, data tree automata and branching vector addition systems

Florent Jacquemard, Luc Segoufin and Jérémie Dimino
INRIA and ENS Cachan

Abstract

A data tree is an unranked ordered tree where each node carries a label from a finite alphabet and a datum from some infinite domain. We consider the two variable first order logic $\mathsf{FO}^2(<,+1,\sim)$ over data trees. Here $+1$ refers to the child and the next sibling relations while $<$ refers to the descendant and following sibling relations. Moreover $\sim$ is a binary predicate testing data equality. We exhibit an automata model, denoted DTA$^\#$, that is more expressive than $\mathsf{FO}^2(<,+1,\sim)$ but such that emptiness of DTA$^\#$ and satisfiability of $\mathsf{FO}^2(<,+1,\sim)$ are inter-reducible. This is proved via a model of counter tree automata, denoted EBVASS, that extends Branching Vector Addition Systems with States (BVASS) with extra features for merging counters. We show that, as decision problems, reachability for EBVASS, satisfiability of $\mathsf{FO}^2(<,+1,\sim)$ and emptiness of DTA$^\#$ are equivalent.

Introduction

A data tree is an unranked ordered tree where each node carries a label from a finite alphabet and a datum from some infinite domain. Together with the special case of data words, they have been considered in the realm of program verification, as they are suitable to model the behavior of concurrent, communicating or timed systems, where data can represent e.g. process identifiers or time stamps [1, 6, 7]. Data trees are also a convenient model for XML documents [4], where data represent attribute values or text contents. Therefore finding decidable logics for this model is a central problem as it has applications in most reasoning tasks in database or in verification.

Several logical formalisms and models of automata over data trees have been proposed. Many of them were introduced in relationship with XPath, the standard formalism to express properties of XML documents. Although satisfiability of XPath in the presence of data values is undecidable, automata models were introduced for showing decidability of several data-aware fragments [12, 4, 11, 10, 13].

As advocated in [4], the logic $\mathsf{FO}^2(<,+1,\sim)$ can be seen as a relevant fragment of XPath. Here $\mathsf{FO}^2(<,+1,\sim)$ refers to the two-variable fragment of first order logic over unranked ordered data trees, with predicates for the child and the next sibling relations ($+1$), predicates for the descendant and following sibling relations ($<$) and a predicate for testing data equality between two nodes ($\sim$). Over data words, $\mathsf{FO}^2(<,+1,\sim)$ was shown decidable by a reduction to Petri Nets or, equivalently, Vector Addition Systems with States (VASS) [5]. It is also shown in [4] that reachability for Branching Vector Addition Systems with States, BVASS, reduces to satisfiability of $\mathsf{FO}^2(<,+1,\sim)$ over data trees. The model of BVASS, extends VASS with a natural branching feature for running on trees, see [15] for a survey of the various formalisms equivalent to BVASS. As the reachability of BVASS is a long standing open problem, showing decidability of $\mathsf{FO}^2(<,+1,\sim)$ seems unlikely in the near future.

This paper is a continuation of the work of [5, 4]. We introduce a model of counter automata, denoted EBVASS, and show that satisfiability of $\mathsf{FO}^2(<,+1,\sim)$ is inter-reducible to reachability in EBVASS. This model extends BVASS by allowing new features for merging counters. In a BVASS the value of a counter at a node x in a tree is the sum of the values of that counter at the children of x, plus or minus some constant specified by the transition relation. In EBVASS a constraint can be added enforcing the following behavior at node x: one of the counters of its left child and one of the counters of its right child are decreased by

the same arbitrary number n, then the sum is performed as for BVASS, and finally, one of the resulting counters is increased by n.

The reduction from $\mathsf{FO}^2(<,+1,\sim)$ to EBVASS goes via a new model of data tree automata, denoted DTA$^\#$. Our first result (Section 2) shows that languages of data trees definable in $\mathsf{FO}^2(<,+1,\sim)$ are also recognizable by DTA$^\#$. Moreover the construction of the automaton from the formula is effective. Our automata model is a non-trivial extension from data words to data trees of the Data Automata (DA) model of [5], chosen with care in order to be powerful enough to capture the logic but also not too powerful in order to match its computational power. The obvious extensions of DA to data trees are either too weak to capture $\mathsf{FO}^2(<,+1,\sim)$ or too expressive and undecidable (see Proposition 1). Here we weaken the strongest of these extensions, still capturing the expressive power of $\mathsf{FO}^2(<,+1,\sim)$, and with an associated emptiness problem equivalent to satisfiability of $\mathsf{FO}^2(<,+1,\sim)$.

Our second result (Section 3) shows that the emptiness problem for DTA$^\#$ reduces to the reachability problem for EBVASS. Finally we show in Section 4 that the latter problem can be reduced to the satisfiability of $\mathsf{FO}^2(<,+1,\sim)$, closing the loop. Altogether, this implies that showing (un)decidability of any of these problems would show (un)decidability of the three of them. Although this question of (un)decidability remains open, the equivalence shown in this paper between the decidability of these three problems, the definition of the intermediate model DTA$^\#$ and the techniques used for proving the interreductions bring a better understanding of the three problems, and in particular of the emptiness of the branching vector addition systems with states.

Related work. There are many other works introducing automata or logical formalism for data words or data trees. Some of them are shown decidable using counter automata, see for instance [9, 13]. The link between counter automata and data automata is not surprising as the corresponding models only compare data values via equality. Hence they are invariant by permutation of the data domain and therefore, often, it is enough to count the number of data values satisfying some properties instead of knowing their precise values.

1 Preliminaries

In this paper $\mathbb{A}$ or $\mathbb{B}$ denote finite alphabets while $\mathbb{D}$ denotes an infinite data domain. We use $\mathbb{E}$ or $\mathbb{F}$ when we do not care whether the alphabet is finite or not.

Unranked ordered data forests. We work with finite unranked ordered trees and forests over an alphabet $\mathbb{E}$, defined inductively as follows: for any $a \in \mathbb{E}$, a is a tree. If $\boldsymbol{t}_1, \cdots, \boldsymbol{t}_k$ is a finite non empty sequence of trees then $\boldsymbol{t}_1 + \cdots + \boldsymbol{t}_k$ is a forest. If $\boldsymbol{s}$ is a forest and $a \in \mathbb{E}$, then $a(\boldsymbol{s})$ is a tree. The set of trees and forests over $\mathbb{E}$ are respectively denoted $\mathsf{Trees}(\mathbb{E})$ and $\mathsf{Forests}(\mathbb{E})$. A tree is called unary (resp. binary) when every node has at most one (resp. two) children. We use standard terminology for trees and forests defining nodes, roots, leaves, parents, children, ancestors, descendants, following and preceding siblings.

Given a forest $\boldsymbol{t} \in \mathsf{Forests}(\mathbb{E})$, and a node x of $\boldsymbol{t}$, we denote by $\boldsymbol{t}(x)$ the label of x in $\boldsymbol{t}$.

We say that two forests $\boldsymbol{t}_1 \in \mathsf{Forests}(\mathbb{E}_1)$ and $\boldsymbol{t}_2 \in \mathsf{Forests}(\mathbb{E}_2)$ *have the same domain* if there is a bijection from the nodes of $\boldsymbol{t}_1$ to the nodes of $\boldsymbol{t}_2$ that respects the parent and the next-sibling relations. In this case we identify the nodes of $\boldsymbol{t}_1$ with the nodes of $\boldsymbol{t}_2$ and the difference between $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ lies only in the label associated to each node. Given two forests $\boldsymbol{t}_1 \in \mathsf{Forests}(\mathbb{E}_1)$, $\boldsymbol{t}_2 \in \mathsf{Forests}(\mathbb{E}_2)$ having the same domain, we define $\boldsymbol{t}_1 \otimes \boldsymbol{t}_2 \in \mathsf{Forests}(\mathbb{E}_1 \times \mathbb{E}_2)$ as the forest over the same domain and such that for all node x, $\boldsymbol{t}_1 \otimes \boldsymbol{t}_2(x) = \langle \boldsymbol{t}_1(x), \boldsymbol{t}_2(x) \rangle$.

The set of *data forests* over a finite alphabet $\mathbb{A}$ and an infinite data domain $\mathbb{D}$ is defined as $\mathsf{Forests}(\mathbb{A}\times\mathbb{D})$. Note that every data forest $\boldsymbol{t} \in \mathsf{Forests}(\mathbb{A}\times\mathbb{D})$ can be decomposed into two forests $\boldsymbol{a} \in \mathsf{Forests}(\mathbb{A})$ and $\boldsymbol{d} \in \mathsf{Forests}(\mathbb{D})$ such that $\boldsymbol{t} = \boldsymbol{a} \otimes \boldsymbol{d}$.

Logics on data forests. A data forests of $\mathsf{Forests}(\mathbb{A}\times\mathbb{D})$ is seen as relational model for first order logic. The domain of the model is the set of nodes in the forest. There is a unary relation $a(x)$ for all $a \in \mathbb{A}$ containing the nodes of label a. There is a binary relation $x \sim y$ containing all pairs of nodes carrying the same data value of $\mathbb{D}$, and binary relations $\mathsf{E}_{\rightarrow}(x,y)$ (y is the sibling immediately next to x), $\mathsf{E}_{\downarrow}(x,y)$ (x is the parent of y), and $\mathsf{E}_{\rightrightarrows}$, $\mathsf{E}_{\downdownarrows}$ which are the non reflexive transitive closures respectively of $\mathsf{E}_{\rightarrow}$ and

$\mathsf{E}_{\downarrow}$, minus respectively $\mathsf{E}_{\rightarrow}$ and $\mathsf{E}_{\downarrow}$ (i.e. they define two or more navigation steps). The reason for this non standard definition of $\mathsf{E}_{\rightrightarrows}$ and $\mathsf{E}_{\downarrow\downarrow}$ is that it will be convenient that equality, $\mathsf{E}_{\rightarrow}$, $\mathsf{E}_{\downarrow}$, $\mathsf{E}_{\rightrightarrows}$ and $\mathsf{E}_{\downarrow\downarrow}$ are disjoint binary relations. We will often make use of the macro, $x \not\approx y$, and say that x and y are *incomparable*, when none of $x = y$, $\mathsf{E}_{\rightarrow}(x, y)$, $\mathsf{E}_{\downarrow}(x, y)$, $\mathsf{E}_{\rightrightarrows}(x, y)$ and $\mathsf{E}_{\downarrow\downarrow}(x, y)$ holds.

Let $\mathsf{FO}^2(<, +1, \sim)$ be the set of first order sentences with two variables built on top of the above predicates. Typical examples of properties definable in $\mathsf{FO}^2(<, +1, \sim)$ are key constraints (all nodes of label a have different data values), $\forall x \forall y\ a(x) \wedge a(y) \wedge x \sim y \rightarrow x = y$, and downward inclusion constraints (every node x of label a has a node y of label b in its subtree with the same data value), $\forall x \exists y\ a(x) \rightarrow (b(y) \wedge x \sim y \wedge (\mathsf{E}_{\downarrow\downarrow}(x, y) \vee \mathsf{E}_{\downarrow}(x, y))$.

We also consider the extension $\mathsf{EMSO}^2(<, +1, \sim)$ of $\mathsf{FO}^2(<, +1, \sim)$ with existentially quantified monadic second order variables. Every formula of $\mathsf{EMSO}^2(<, +1, \sim)$ has the form $\exists R_1 \ldots \exists R_n\ \phi$ where ϕ is a $\mathsf{FO}^2(<, +1, \sim)$ formula called the *core*, involving the variables $R_1, \ldots, R_n$ as unary predicates. The extension to full monadic second order logic is denoted $\mathsf{MSO}(<, +1, \sim)$.

We write $\mathsf{MSO}(<, +1)$ for formulas not using the $\sim$ predicates, formulas ignoring the data values, i.e. classical formulas over forests.

Automata models for forests. We will informally refer to automata and transducers for forests and unranked trees over a finite alphabet. The particular choice of a model of automata is not relevant here and we refer to [8] for a detailed description. A set of forests accepted by an automaton is called a *regular language* and regular languages are exactly those definable in $\mathsf{MSO}(<, +1)$.

We denote by $\mathbb{E}_{\#}$ the extension of an alphabet $\mathbb{E}$ with a new symbol $\#$ that does not occur in $\mathbb{E}$. It is well known that forests of $\mathsf{Forests}(\mathbb{E})$ can be transformed into binary trees in $\mathsf{Trees}(\mathbb{E}_{\#})$ using the first-child/right-sibling encoding, denoted by $fcns$, and formally defined as follows (for $a \in \mathbb{E}$ and $\boldsymbol{s}, \boldsymbol{s}' \in \mathsf{Forests}(\mathbb{E})$):

$$\begin{array}{rcl} fcns(a) & = & a(\# + \#) \\ fcns(a(\boldsymbol{s})) & = & a(fcns(\boldsymbol{s}) + \#) \\ fcns(a + \boldsymbol{s}) & = & a(\# + fcns(\boldsymbol{s})) \\ fcns(a(\boldsymbol{s}) + \boldsymbol{s}') & = & a(fcns(\boldsymbol{s}) + fcns(\boldsymbol{s}')). \end{array}$$

This transformation effectively preserves regularity: for each automaton $\mathcal{B}$ computing on $\mathsf{Forests}(\mathbb{E})$ there exists an automaton $\mathcal{B}'$ on binary trees of $\mathsf{Trees}(\mathbb{E}_{\#})$, effectively computable from $\mathcal{B}$, recognizing exactly the $fcns$ encoding of the forests recognized by $\mathcal{B}$. This automaton $\mathcal{B}'$ is called the *$fcns$ view* of $\mathcal{B}$.

Automata models for data forests. Given a data forest $\boldsymbol{t} = \boldsymbol{b} \otimes \boldsymbol{d} \in \mathsf{Forests}(\mathbb{B} \times \mathbb{D})$ and a data value $d \in \mathbb{D}$, the *class forest* $\boldsymbol{t}[d]$ of $\boldsymbol{t}$ associated to the datum d is the forest of $\mathsf{Forests}(\mathbb{B}_{\#})$ having the same domain as $\boldsymbol{t}$ and such that $\boldsymbol{t}[d](x) = \boldsymbol{b}(x)$ if $\boldsymbol{d}(x) = d$ and $\boldsymbol{t}[d](x) = \#$ otherwise.

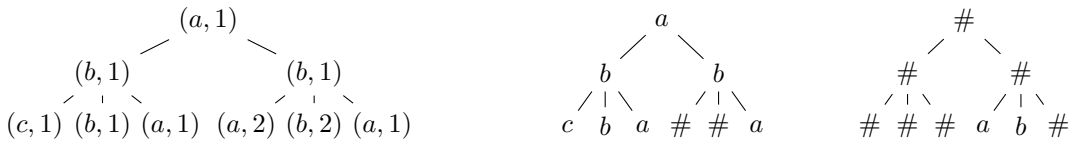

Fig. 1: A forest $\boldsymbol{t}$ followed by its class forests $\boldsymbol{t}[1]$ and $\boldsymbol{t}[2]$

We now define two models of automata over data trees. The first and most general one is a straightforward generalization to forests of the automata model over data words of [5]. The second one adds a restriction in order to avoid undecidability.

General Data Forest Automata model: DTA. A DTA is a pair $(\mathcal{A}, \mathcal{B})$ where $\mathcal{A}$ is a non-deterministic letter-to-letter transducer taking as input a forest in $\mathsf{Forests}(\mathbb{A})$ and returning a forest in $\mathsf{Forests}(\mathbb{B})$ with the same domain, while $\mathcal{B}$ is a forest automaton taking as input a forest in $\mathsf{Forests}(\mathbb{B}_{\#})$. Intuitively a DTA

works as follows on a forest $\boldsymbol{t} = \boldsymbol{a} \otimes \boldsymbol{d}$: first the transducer $\mathcal{A}$ relabels the nodes of $\boldsymbol{a}$ into $\boldsymbol{b}$ and the forest automaton $\mathcal{B}$ has to accept all class forests of $\boldsymbol{b} \otimes \boldsymbol{d}$.

More formally a data forest $\boldsymbol{t} = \boldsymbol{a} \otimes \boldsymbol{d} \in \mathsf{Forests}(\mathbb{A} \times \mathbb{D})$ is accepted by $(\mathcal{A}, \mathcal{B})$ iff

1. there exists $\boldsymbol{b} \in \mathsf{Forests}(\mathbb{B})$ such that $\boldsymbol{b}$ is a possible output of $\mathcal{A}$ on $\boldsymbol{a}$ and,
2. for all $d \in \mathbb{D}$, the class forest $(\boldsymbol{b} \otimes \boldsymbol{d})[d] \in \mathsf{Forests}(\mathbb{B}_{\#})$ is accepted by $\mathcal{B}$.

Over data words this model was shown decidable [5]. Unfortunately it is undecidable over data trees.

Proposition 1. *Emptiness of* DTA *is undecidable.*

Proof. We show that DTA can simulate the Class Automata of [3]. This latter model has an undecidable emptiness problem, already when restricted to data words, *i.e.* forests of the form $\langle a_1, d_1\rangle + \ldots + \langle a_m, d_m\rangle$). It captures indeed the class of languages of words - without data - recognized by counter automata.

We assume given two finite alphabets $\mathbb{A}$ and $\mathbb{B}$, writing the latter *in extenso* as $\mathbb{B} = \{b_1, \ldots, b_n\}$. A class automaton over $\mathbb{A} \times \mathbb{D}$ is a pair $\mathcal{C} = (\mathcal{A}, \mathcal{B})$ where $\mathcal{A}$ is a non-deterministic letter-to-letter word transducer from $\mathbb{A}$ into $\mathbb{B}$ and $\mathcal{B}$ is a word automaton taking as input words over the alphabet $\mathbb{B} \times \{0, 1\}$. In order to define the acceptation of data words by class automata, we shall use a notion of class word associated to a data word $\boldsymbol{w} = \boldsymbol{b} \otimes \boldsymbol{d}$ and a value $d \in \mathbb{D}$, denoted $\boldsymbol{w}[\![d]\!]$, defined as the word having the same domain as $\boldsymbol{w}$ and such that, for every node x of $\boldsymbol{w}$, $\boldsymbol{w}[\![d]\!](x) = \langle \boldsymbol{b}(x), 1\rangle$ if $\boldsymbol{d}(x) = d$ and $\boldsymbol{w}[\![d]\!](x) = \langle \boldsymbol{b}(x), 0\rangle$ otherwise. A data word $\boldsymbol{w} = \boldsymbol{a} \otimes \boldsymbol{d}$ is accepted by $\mathcal{C}$ iff

1. there exists a word $\boldsymbol{b}$ over $\mathbb{B}$ such that $\boldsymbol{b}$ is a possible output of $\mathcal{A}$ on $\boldsymbol{a}$ and,
2. for all $d \in \mathbb{D}$, the class word $(\boldsymbol{b} \otimes \boldsymbol{d})[\![d]\!]$ is accepted by $\mathcal{B}$.

The difference between Data Automata and Class Automata is that the $\mathcal{B}$ part in the Class automata has access to the label of the nodes that are not in the class, while it sees only $\#$ in the Data Automata case. This extra power implies undecidability.

Given a class automaton $\mathcal{C} = (\mathcal{A}, \mathcal{B})$ over $\mathbb{A} \times \mathbb{D}$, we construct a DTA $\mathcal{C}'$ such that $\mathcal{C}$ accepts a data word iff $\mathcal{C}'$ accepts a data tree. The idea of the reduction is that we replace each letter b_i by a tree of depth i. Hence, even if b_i is replaced by $\#$ during the run of $\mathcal{C}'$ (conversion to class word), this label can still be recovered.

Let $\mathbb{O}$ be a new alphabet containing the two symbols b and $\#$. For any symbol s and $1 \leq i \leq n$, let s^i be the unary data tree of depth i defined recursively by: $s^1 = s$ and $s^{i+1} = s(s^i)$. We associate to a data word $\boldsymbol{w} = \langle b_{i_1}, d_1\rangle + \ldots + \langle b_{i_m}, d_m\rangle$ a binary data tree $\hat{\boldsymbol{w}} \in \mathsf{Trees}(\mathbb{O} \times \mathbb{D})$ defined by $\hat{\boldsymbol{w}} = \langle b, d_1\rangle^{i_1+1} + \ldots + \langle b, d_m\rangle^{i_m+1}$.

From the word automaton $\mathcal{B}$ we construct a forest automaton $\mathcal{B}'$ accepting exactly the set of class forests $\hat{\boldsymbol{w}}[d]$ such that $\boldsymbol{w}[\![d]\!]$ is accepted by $\mathcal{B}$, for all $d \in \mathbb{D}$. For simplicity we describe the $fcns$ view of $\mathcal{B}'$. The automaton $\mathcal{B}'$ contains the states of $\mathcal{B}$ plus some additional states $p_{i,j}$ with $1 \leq i \leq n$ and $j \in \{0, 1\}$, such that $p_{i,0}$ and $p_{i,1}$ accept respectively the singletons languages $\{\#^i\}$ and $\{b^i\}$, with transitions of the form $\# \to p_{0,0}$, $b \to p_{0,1}$, and $(p_{i,0}, \#, p_{0,0}) \to p_{i+1,0}$, $(p_{i,1}, b, p_{0,0}) \to p_{i+1,1}$ for all $0 \leq i \leq n-1$. We also add to $\mathcal{B}'$ a transition $\# \to q_0$, where q_0 is the initial state of $\mathcal{B}$. Finally, for each transition of $\mathcal{B}$ of the form $(q, \langle b_i, 0\rangle) \to q'$, (respectively $(q, \langle b_i, 1\rangle) \to q'$), where q and q' are states of $\mathcal{B}$ and $1 \leq i \leq n$, $\mathcal{B}'$ contains a transition $(p_{i,0}, \#, q) \to q'$ (respectively $(p_{i,1}, b, q) \to q'$).
Defining the set of final states of $\mathcal{B}'$ as the final states of $\mathcal{B}$, we get the desired $\mathcal{B}'$.

From there it is now easy to construct an $\mathcal{A}'$ such that the DTA $(\mathcal{A}', \mathcal{B}')$ accepts a data forest iff the class automaton $\mathcal{C} = (\mathcal{A}, \mathcal{B})$ accepts a data word. □

Restricted Data Forest Automata model: DTA$^{\#}$. The second data tree automata model DTA$^{\#}$ we consider is defined as for DTA with a restriction on $\mathcal{B}$. The restriction makes sure that $\mathcal{B}$ ignores repeated and contiguous occurrences of $\#$ symbols. This ensures that, after class projection, not only the automata cannot see the label of a node not in the class, but also can not see the shape of hidden subtrees. We give two equivalent definitions for the restriction on $\mathcal{B}$. The first one is is a syntactic restriction on the $fcns$

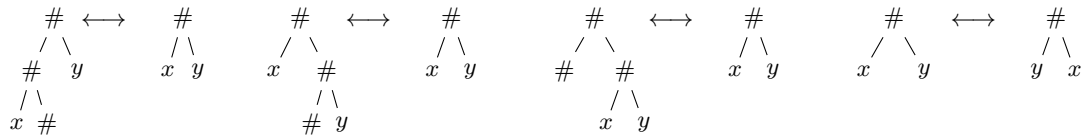

Fig. 2: Closure rules for $\mathcal{B}'$.

view $\mathcal{B}'$ of $\mathcal{B}$, while the second one is based on a closure property that must be satisfied by the language recognized by $\mathcal{B}$.

For the definition of the syntactic restriction on $\mathcal{B}'$, let us assume without loss of generality that the states of $\mathcal{B}'$ permit to distinguish the last symbol read by $\mathcal{B}'$. More precisely, we assume that the set of states of $\mathcal{B}'$ is split into two kinds: the $\#$-states and the non-$\#$-states. The states of the first kind are reached by $\mathcal{B}'$ on nodes labeled with symbol $\#$, while the states of the second kind are reached by $\mathcal{B}'$ on nodes with label in $\mathbb{B}$. We say that $\mathcal{B}'$ is *$\#$-stuttering* if $\mathcal{B}'$ is deterministic and has a specific $\#$-state $p_\#$ that it must reach on all leaves of label $\#$, and verifies the following properties:

1. if a transition rule of the form $(p_1, \#, p_\#) \to p_2$ is applied at a $\#$-node that is the left-child of another $\#$-node, then $p_1 = p_2$
2. if a transition rule of the form $(p_\#, \#, p_1) \to p_2$ is applied at a $\#$-node that is the right-child of another $\#$-node, then $p_1 = p_2$
3. all transition rules of the form $(p_\#, \#, p_1) \to p_2$ with p_1 a $\#$-state verify $p_1 = p_2$.
4. all transition rules of the form $(p_1, \#, p_2) \to p$ are "commutative", i.e. $(p_2, \#, p_1) \to p$ must then also be a rule.

Then a DTA$^\#$ $(\mathcal{A}, \mathcal{B})$ is a DTA such that the $fcns$ view $\mathcal{B}'$ of $\mathcal{B}$ is $\#$-stuttering.

These conditions have a semantical counterpart. A language of $L \subseteq \mathsf{Forests}(\mathbb{B})$ is called *$\#$-stuttering* iff it is closed under the rules depicted in Figure 3. Intuitively these rules should be understood as follows: if a tree in L as a part matching the left-hand side, then replacing this part by the right-hand side yields a tree also in L, and the other way round.

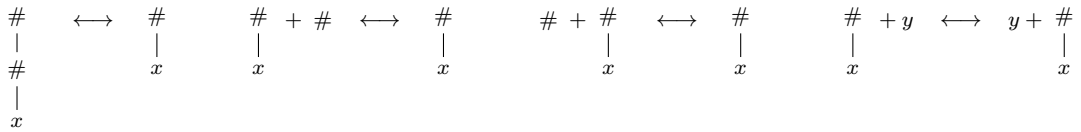

Fig. 3: Closure rules for $\mathcal{B}$.

For instance if L is $\#$-stuttering and contain the trees $\boldsymbol{t}[1]$ and $\boldsymbol{t}[2]$ of Figure 1, then it should also contain the trees in Figure 4.

Typical examples of $\#$-stuttering languages are those testing that no two nodes of label a occur in $\boldsymbol{t}[d]$ (key constraint) or that each node of label a has a descendant of label b in $\boldsymbol{t}[d]$ (inclusion constraint). Typical examples of languages that are not $\#$-stuttering are those counting the number of nodes of label $\#$. Note that $\#$-stuttering languages are closed under union and intersection.

The rules depicted in Figure 3 have an equivalent counterpart on the $fcns$ view depicted in Figure 2. From these definitions, it is immediate to see that for a language $L \subseteq \mathsf{Forests}(\mathbb{B})$, the following properties are equivalent

- L is $\#$-stuttering,
- $fcns(L)$ is closed under the rules in Figure 2,

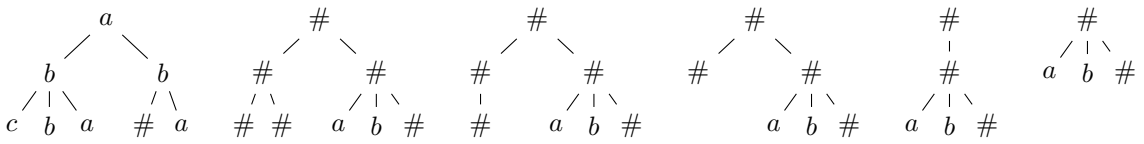

Fig. 4: Closure of $\{\boldsymbol{t}[1], \boldsymbol{t}[2]\}$ of Figure 1.

- there exists an $\#$-stuttering first automaton $\mathcal{B}'$ recognizing $fcns(L)$.

Hence, a DTA $(\mathcal{A}, \mathcal{B})$ is a DTA$^{\#}$ iff the language recognized by $\mathcal{B}$ is $\#$-stuttering.

We conclude this section with the following lemma whose proof is a straightforward Cartesian product construction. We use the term *letter projection* for a relabeling function defined as $h : \mathbb{A}' \to \mathbb{A}$, where $\mathbb{A}$ and $\mathbb{A}'$ are alphabets.

Lemma 2. *The class of* DTA$^{\#}$ *languages is closed under union, intersection and letter projection.*

2 From $\mathsf{FO}^2(<, +1, \sim)$ to DTA$^{\#}$

In this section we show the following result.

Theorem 3. *Given a formula ϕ in $\mathsf{FO}^2(<, +1, \sim)$, there exists a* DTA$^{\#}$*, effectively computable from ϕ, accepting exactly the set of data forests satisfying ϕ.*

The proof works in two steps. During the first step we provide a normal form for sentences of $\mathsf{FO}^2(<, +1, \sim)$ that is essentially an $\mathsf{EMSO}^2(<, +1, \sim)$ formula whose core is a conjunction of simple formula of $\mathsf{FO}^2(<, +1, \sim)$. In a second step, we show that each of the conjunct can be translated into a DTA$^{\#}$, and we conclude using composition of these automata by intersection, see Lemma 2.

2.1 Intermediate Normal Form

We show first that every $\mathsf{FO}^2(<, +1, \sim)$ formula ϕ can be transformed into an equivalent $\mathsf{EMSO}^2(<, +1, \sim)$ formula in *intermediate normal form*:

$$\exists R_1 \cdots \exists R_k \bigwedge_i \chi_i$$

where each χ_i has one of the following forms:

$$\forall x \forall y \quad \alpha(x) \wedge \beta(y) \wedge \delta(x, y) \to \gamma(x, y) \tag{1}$$

$$\forall x \exists y \quad \alpha(x) \to (\beta(y) \wedge \delta(x, y) \wedge \epsilon(x, y)) \tag{2}$$

where each of α and β is a *type*: conjunction of monadic predicates or their negation (these unary predicates are either from $\mathbb{A}$ or from $R_1, \ldots, R_k$, *i.e.* introduced by the existentially quantified variables), $\delta(x, y)$ is either $x \sim y$ or $x \not\sim y$, $\gamma(x, y)$ is one of $\neg \mathsf{E}_{\rightrightarrows}(x, y)$, $\neg \mathsf{E}_{\downdownarrows}(x, y)$ or $\neg(x \bowtie y)$, and $\epsilon(x, y)$ is one of $x = y$, $\mathsf{E}_{\to}(x, y)$, $\mathsf{E}_{\to}(y, x)$, $\mathsf{E}_{\downarrow}(x, y)$, $\mathsf{E}_{\downarrow}(y, x)$, $\mathsf{E}_{\rightrightarrows}(x, y)$, $\mathsf{E}_{\rightrightarrows}(y, x)$, $\mathsf{E}_{\downdownarrows}(x, y)$, $\mathsf{E}_{\downdownarrows}(y, x)$, $x \bowtie y$ or *false*.

This normal form is obtained by simple syntactical manipulation similar to the one given in [5] for the data words case, and detailed below.

Scott normal form. We first transform the formula ϕ into Scott Normal Form obtaining an $\mathsf{EMSO}^2(<, +1, \sim)$ formula of the form:

$$\psi = \exists R_1 \ldots \exists R_m \; \forall x \forall y \; \chi \wedge \bigwedge_i \forall x \exists y \; \chi_i$$

where χ and every χ_i are quantifier free, and $R_1, \ldots R_m$ are new unary predicates. This transformation is folklore: a new unary predicate R_θ is introduced for each subformula $\theta(x)$ with one free variable for marking the nodes where the subformula holds. The subformula $\theta(x)$ is then replaced by $R_\theta(x)$ and a conjunct $\forall x\big(R_\theta(x) \leftrightarrow \theta(x)\big)$ is added. Eventually this yields the desired normal form.

From Scott to intermediate normal form. We show next that every conjunct of the core of the formula ψ in Scott Normal Form can be replaced by an equivalent conjunction of formulas of the form (1) or (2), possibly by adding new monadic quantifications upfront.

Case $\forall x\forall y\ \chi$. Recall that with our definition, the binary relations $\mathsf{E}_{\rightarrow}$, $\mathsf{E}_{\rightrightarrows}$, $\mathsf{E}_{\downarrow}$, $\mathsf{E}_{\downdownarrows}$, $\precsim$ and $=$ are pairwise disjoint. Hence we can rewrite $\forall x\forall y\ \chi$ into an equivalent $\mathsf{FO}^2(<, +1, \sim)$ formula in the following form,

$$\forall x\forall y \left(\begin{array}{ccccccccccccc} & x=y & \rightarrow & \psi_{=}(x,y) & \wedge & \mathsf{E}_{\rightarrow}(x,y) & \rightarrow & \psi_{\rightarrow}(x,y) & \wedge & \mathsf{E}_{\rightrightarrows}(x,y) & \rightarrow & \psi_{\rightrightarrows}(x,y) \\ \wedge & x \precsim y & \rightarrow & \psi_{\precsim}(x,y)\big) & \wedge & \mathsf{E}_{\downarrow}(x,y) & \rightarrow & \psi_{\downarrow}(x,y) & \wedge & \mathsf{E}_{\downdownarrows}(x,y) & \rightarrow & \psi_{\downdownarrows}(x,y) \end{array} \right)$$

where every subformula ψ_* is quantifier free and only involves the predicate $\sim$ together with monadic predicates. They can be obtained from χ in the obvious way. The resulting formula is equivalent to the conjunction

$$\begin{array}{lll} & \forall x\exists y & \big(x = y \wedge \psi_{=}(x,y)\big) \\ \wedge & \forall x\exists y & \big(\neg\mathsf{last}(x) \rightarrow (\mathsf{E}_{\rightarrow}(x,y) \wedge \psi_{\rightarrow}(x,y))\big) \\ \wedge & \forall x\exists y & \big(\neg\mathsf{leaf}(x) \rightarrow (\mathsf{E}_{\downarrow}(x,y) \wedge \psi_{\downarrow}(x,y))\big) \\ \wedge & \forall x\forall y & \mathsf{E}_{\rightrightarrows}(x,y) \rightarrow \psi_{\rightrightarrows}(x,y) \\ \wedge & \forall x\forall y & \mathsf{E}_{\downdownarrows}(x,y) \rightarrow \psi_{\downdownarrows}(x,y) \\ \wedge & \forall x\forall y & x \precsim y \rightarrow \psi_{\precsim}(x,y) \end{array}$$

where $\mathsf{leaf}(x)$ is a new predicate denoting the leaves of the forest and $\mathsf{last}(x)$ is also a new predicate denoting nodes having no right sibling. The predicate leaf is specified by the following formulas, that have the desired form.

$$\begin{array}{ll} \forall x\forall y & \mathsf{E}_{\downarrow}(x,y) \wedge \mathsf{leaf}(x) \rightarrow \mathit{false} \\ \forall x\exists y & \neg\mathsf{leaf}(x) \rightarrow \mathsf{E}_{\downarrow}(x,y) \end{array}$$

Similar formulas specify the predicate last.

The first three conjuncts, with quantifier prefix $\forall x\exists y$, will be treated later when dealing with the second case.

For the next three conjuncts, putting $\neg\psi_{\rightrightarrows}$, $\neg\psi_{\downdownarrows}$, $\neg\psi_{\precsim}$ in disjunctive normal form (with an exponential blowup), we rewrite $\psi_{\rightrightarrows}$, $\psi_{\downdownarrows}$, $\psi_{\precsim}$ as a conjunction of formulas of the form $\neg(\alpha(x) \wedge \beta(y) \wedge \delta(x,y))$, where α, β, and δ are as in (1). By distribution of conjunction over implication, and by contraposition, we obtain for the 3 cases an equivalent conjunction of formulas of the following form (matching the desired form (1))

$$\begin{array}{ll} \forall x\forall y & \alpha(x) \wedge \beta(y) \wedge \delta(x,y) \rightarrow \neg\mathsf{E}_{\rightrightarrows}(x,y) \\ \forall x\forall y & \alpha(x) \wedge \beta(y) \wedge \delta(x,y) \rightarrow \neg\mathsf{E}_{\downdownarrows}(x,y) \\ \forall x\forall y & \alpha(x) \wedge \beta(y) \wedge \delta(x,y) \rightarrow \neg(x \precsim y) \end{array}$$

Case $\forall x\exists y\ \chi$. We first transform χ (with an exponential blowup) into an equivalent disjunction of the form

$$\chi' = \bigvee_j \alpha_j(x) \wedge \beta_j(y) \wedge \delta_j(x,y) \wedge \epsilon_j(x,y)$$

where α_j, β_j, δ_j and ϵ_j are as in (2). Next, in order to eliminate the disjunctions, we add a new monadic second-order variables $R_{\chi,j}$, that we existentially quantify upfront of the global formula, and transform $\forall x\exists y\ \chi'$ into the conjunction

$$\bigwedge_j \forall x\exists y\ (\alpha_j(x) \wedge R_{\chi,j}(x)) \rightarrow (\beta_j(y) \wedge \delta_j(x,y) \wedge \epsilon_j(x,y)) \wedge \forall x\exists y\ \big(\bigvee_j R_{\chi,j}(x)\big)$$

The first conjuncts express that if $R_{\chi,j}(x)$ holds, then there exists a node y such that the corresponding conjunct of χ' holds, and the last conjunct expresses that for all node x, at least one of the $R_{\chi,j}(x)$ must hold and can be rewritten as $\forall x\exists y\ \big(\bigwedge \neg R_{\chi,j}(x) \rightarrow \mathit{false}\big)$. Now all the conjuncts are as in (2) and we are done.

2.2 Case analysis for constructing DTA$^{\#}$ from intermediate normal forms

We now show how to transform a formula in intermediate normal form into a DTA$^{\#}$. Let $\mathbb{A}$ be the initial alphabet and let $\mathbb{A}'$ be the new alphabet formed by combining letters of $\mathbb{A}$ with the newly quantified monadic predicates $R_1, \ldots, R_k$. By closure of DTA$^{\#}$ under intersection and letter projection (Lemma 2), it is enough to construct a DTA$^{\#}$ automaton for each simple formula of the form (1) or (2), accepting the data forests in $\mathsf{Forests}(\mathbb{A}' \times \mathbb{D})$ satisfying the formula.

We do a case analysis depending on the atoms involved in the formula of the form (1) or (2). For each case we construct a DTA$^{\#}$ $(\mathcal{A}, \mathcal{B})$ recognizing the set of data forests satisfying the formula. The construction borrows several ideas from the data word case [5], but some extra work is needed as the tree structure is more complicated. In the discussion below, a node whose label satisfies the type α will be called an α-*node*. Many of the cases build on generic constructions that we described in the following remark.

Remark 1. *A* DTA$^{\#}$ $(\mathcal{A}, \mathcal{B})$ *can be used to distinguish one specific data value. We will then say that* $(\mathcal{A}, \mathcal{B})$ marks the data value of a node x using the new color c*. This can be done as follows. The transducer* $\mathcal{A}$ *marks the node* x *with a specific new color* c'*. At the same time it guesses all the nodes sharing the same data value as* x *and marks each of them with a new color* c*. Then, the forest automaton* $\mathcal{B}$ *checks, for every data value, that either none of the nodes are marked with* c *or* c'*, or that all nodes not labeled with* $\#$ *are marked with* c *or* c' *and that* c' *occurs exactly once in the same class forest. Note that this defines a* $\#$*-stuttering language. It is now clear that for the run to be accepting,* $\mathcal{A}$ *must color exactly one data value and that all the nodes carrying this data value must be marked with* c *and* c'*. The transducer* $\mathcal{A}$ *can then build on this fact for checking other properties.*

A generic example of the usefulness of this remark is given below. Once a data value is marked with a color c, then a property of the form $\forall x \forall y\ \alpha(x) \wedge \beta(y) \wedge x \not\sim y \rightarrow \gamma(x, y)$ is a conjunction of $\forall x \forall y\ \alpha(x) \wedge c(x) \wedge \beta(y) \wedge \neg c(y) \rightarrow \gamma(x, y)$ with $\forall x \forall y\ \alpha(x) \wedge \neg c(x) \wedge \beta(y) \wedge x \not\sim y \rightarrow \gamma(x, y)$. The first part, $\forall x \forall y\ \alpha(x) \wedge c(x) \wedge \beta(y) \wedge \neg c(y) \rightarrow \gamma(x, y)$ is now a regular property and can therefore be tested by $\mathcal{A}$. Hence it is enough to consider the case where x does not carry the marked data value. The same reasoning holds if two data values are marked or if the formula starts with a $\forall x \exists y$ quantification. We will use this fact implicitly in the case analysis below.

Given a data forest, a *vertical path* is a set of nodes containing exactly one leaf and all its ancestors and nothing else. A *horizontal path* is a set of nodes containing one node together with all its siblings and nothing else.

We start with formulas of the form (1).

Case 1: $\forall x \forall y\ \alpha(x) \wedge \beta(y) \wedge x \sim y \rightarrow \gamma(x, y)$, where $\gamma(x, y)$ is as in (1). These formulas express a property of pairs of nodes with the same data value. These are $\#$-stuttering languages that can be tested by the forest automaton $\mathcal{B}$ solely (*i.e.* with $\mathcal{A}$ does nothing).

Case 2: $\forall x \forall y\ \alpha(x) \wedge \beta(y) \wedge x \not\sim y \rightarrow \neg \mathsf{E}_{\rightrightarrows}(x, y)$. This formula expresses that a data forest cannot contain an α-node having a β-node with a different data value as a sibling to its right, except if it is the next-sibling. Let X be an horizontal path in a data forest $\boldsymbol{t}$ containing at least one α-node, and let x be the leftmost α-node in X. Let d be the data value of x. Consider an α-node x' and a β-node y' that make the formula false within X, in particular we have $\mathsf{E}_{\rightrightarrows}(x', y')$. Then, if y' has a data value different from d we already have $\mathsf{E}_{\rightrightarrows}(x, y')$ and the formula is also false for the pair (x, y'). Hence the validity of the formula within X can be tested over pairs (x', y') such that either x' or y' has data value d.
With this discussion in mind we construct $(\mathcal{A}, \mathcal{B})$ as follows. In every horizontal path X containing one α-node, the transducer $\mathcal{A}$ identify the leftmost occurrence x of an α-node in X, and marks it with a new color c', and marks all the nodes of X with the same data as x with a color c. As in Remark 1, the forest automaton $\mathcal{B}$ checks that the guesses are correct, *i.e.* it accepts only forests in which every horizontal path X satisfy one of the following conditions: X contains one occurrence of the color c' and all other nodes of X not labeled with $\#$ are marked with c, or X contains none of the colors c and c' at all. All these properties define regular and $\#$-stuttering languages, and hence can be checked by a forest automaton $\mathcal{B}$.

Assuming this, the transducer $\mathcal{A}$ rejects if there are some unmarked β-nodes occurring as a right sibling (except for the next-sibling) of a marked α-node or there is an unmarked α-node as left sibling, except for the previous sibling, of a marked β-node. As explained in Remark 1, this is a regular property.

Case 3: $\forall x \forall y\ \alpha(x) \wedge \beta(y) \wedge x \not\sim y \rightarrow \neg \mathsf{E}_{\downarrow\downarrow}(x, y)$. The property expressed by this formula is similar to the previous case, replacing the right sibling relationship with the descendant relationship.

Let X be a vertical path in a data forest $\boldsymbol{t}$ containing at least one α-node, and let x be the α-node in X the closest to the root. Let d be the data value of x. Consider an α-node x' and a β-node y' that make the formula false within X, in particular we have $\mathsf{E}_{\downarrow\downarrow}(x', y')$. Then, if y' has a data value different from d we already have $\mathsf{E}_{\downarrow\downarrow}(x, y')$ and the formula is also false for the pair (x, y'). Hence the validity of the formula within X can be tested over pairs (x', y') such that either x' or y' has data value d.
The construction of $(\mathcal{A}, \mathcal{B})$ is similar to the previous case, except that different vertical paths may share some nodes. The transducer $\mathcal{A}$ marks all the α-nodes that have no α-node as ancestor, with a new color c'. Then, for every node x marked c', $\mathcal{A}$ guesses all the nodes inside the subtree rooted at x having the same data value as x and mark them with a new color c. As in Remark 1, the forest automaton $\mathcal{B}$ checks that the guesses of colors are correct for each vertical path (see also the previous case).

Assuming this, the transducer $\mathcal{A}$ rejects if there are some unmarked β-nodes that is a descendant, but not a child, of a marked α-node or there is an unmarked α-node as an ancestor, except for the parent, of a marked β-node. As explained in Remark 1, this is a regular property.

Case 4: $\forall x \forall y\ \alpha(x) \wedge \beta(y) \wedge x \not\sim y \rightarrow \neg(x \approx y)$. The formula expresses that every two nodes of type respectively α and β and with different data values cannot be incomparable. Recall that two nodes are incomparable if they are not parents and not siblings.

Subcase 4.1: There exists two α-nodes that are incomparable.

Let x_1 and x_2 be two incomparable α-nodes and let z be their least common ancestor. We can choose x_1 and x_2 such that none of the α-nodes are incomparable with z or sibling of z, because if this was not the case then there is an α-node x_3 incomparable with z or sibling of z, and therefore x_3 is incomparable with x_1, and we can replace x_2 with x_3, continuing with their least common ancestor, a node which is strictly higher than z. Let z_1 and z_2 be the children of z that are respectively ancestors of x_1 and x_2. Note that by construction, $z_1 \neq z_2$. If $x_1 = z_1$ and there is an α-node x_3 in the subtree of z, different from x_1 and incomparable with z_2, then we replace x_1 by x_3 and proceed. In other words we ensure that if $x_1 = z_1$ then there is no α-node incomparable with z_2 in the subtree of z. We do the same trick to enforce that if $x_2 = z_2$ then there is no α-node incomparable with z_1 in the subtree of z. Notice that we cannot have at the same time $x_1 = z_1$ and $x_2 = z_2$ because we assumed x_1 and x_2 to be incomparable. All these properties can be specified in MSO and therefore can be tested by a forest automaton. Let d_1 and d_2 be the respective data values of x_1 and x_2 (possibly $d_1 = d_2$).

Consider now a β-node y whose data value is neither d_1 nor d_2. If y is incomparable with z or sibling of z, then the formula cannot be true as it is contradicted by (x_1, y). If y is an ancestor of z then, as no α-node is incomparable with z, none is incomparable with y. Hence the formula can only be true with such y. Assume now that y is inside the subtree of z. If $y = z_1$ and $x_2 \neq z_2$, then the formula is contradicted by (x_2, y). If $y = z_1$ and $x_2 = z_2$, then, by hypothesis, there is no α-node incomparable with y in the subtree of z, and there is no α-node incomparable with y outside the subtree of z, and altogether, and the formula holds for y. If $y \neq z_1$ and y is a descendant of z_1, then the formula is contradicted by (x_2, y). The cases where y is descendant of y_2 are symmetric: in this case, the formula can only be true if $y = y_2$ and $x_1 = z_1$. In the remaining cases y is in the subtree of z and not in the subtrees of y_1 and y_2, making the formula false. Indeed, in each of these cases, either (x_1, y) or (x_2, y) contradicts the formula. To summarize, the only cases making the formula true are when y is an ancestor of z, or $y = z_1 \wedge x_2 = z_2$, or $y = z_2 \wedge x_1 = z_1$.

With this discussion in mind, this case can be solved as follows: The transducer $\mathcal{A}$ guesses the nodes of x_1, x_2, z_1, z_2 and z and checks that they satisfy the appropriate properties. Moreover, $\mathcal{A}$ guesses whether $d_1 = d_2$ and marks accordingly the data values of x_1 and x_2 with one or two new colors. The forest automaton $\mathcal{B}$ will then check that the data values are marked appropriately as in Remark 1.

Moreover $\mathcal{A}$ checks that for all marked β-nodes there is no α-node incomparable with it and with a different data value, a regular property as explained in Remark 1. It now remains for $\mathcal{A}$ to check that every unmarked β-node y behaves according to the discussion above: y is an ancestor of z or $y = z_1$ and $x_2 = z_2$ or $y = z_2$ and $x_1 = z_1$. This is a regular property testable by $\mathcal{A}$.

Subcase 4.2: There are no two incomparable α-nodes.

Let x be an α-node such that no α-node is a descendant of x. By hypothesis, all α-nodes are either ancestors or siblings of x. Let d be the data value of x. We distinguish between several subcases depending on whether there are other α-nodes that are siblings of x or not.

If there is an α-node x' that is a sibling of x, then let let d' be its data value (possibly $d = d'$). Consider now a β-node y whose data value is neither d nor d'. Then, in order to make the formula true, y must be an ancestor or a sibling of x.

In this case, the transducer $\mathcal{A}$ guesses the nodes x and x' and marks the corresponding data values with one or two new colors (according to whether $d = d'$ or not). The forest automaton $\mathcal{B}$ will then check that the data values are marked correctly as explained in Remark 1. For the marked β-nodes, the property expressed by the formula is regular and can also be checked by $\mathcal{A}$. It remains for $\mathcal{A}$ to check that every unmarked β-nodes is either an ancestor of x or a sibling of x.

Now, if there are no α-nodes that are sibling of x, and y is a β-node whose data value is not d, then in order to make the formal true, y cannot be incomparable with x, and therefore, y can be an ancestor, a descendant or a sibling of x.

In this second case, the transducer $\mathcal{A}$ guesses the node x, marks its data value using a new color. The forest automaton $\mathcal{B}$ will then check that the data values were marked correctly as explained in Remark 1. The transducer $\mathcal{A}$ checks that all marked β-nodes make the formula true (a regular property), and that all unmarked β-nodes are not incomparable with x.

We now turn to formulas of the form (2).

Case 5: $\forall x \exists y\ \alpha(x) \rightarrow (\beta(y) \wedge x \sim y \wedge \epsilon(x,y))$, where $\epsilon(x,y)$ is as in (2). These formulas express properties of nodes with the same data. Moreover they express a regular property over all $\boldsymbol{t}[d]$. Therefore can be treated by the forest automaton $\mathcal{B}$ as for the case 1.

Case 6: $\forall x \exists y\ \alpha(x) \rightarrow (\beta(y) \wedge x \not\sim y \wedge \mathsf{E}_{\rightarrow}(x,y))$. This formula expresses that every α-node has a next sibling of type β with a different data value. The transducer $\mathcal{A}$ marks every α-node x, with a new color c and checks that the next-sibling of x is a β-node. The forest automaton $\mathcal{B}$ accepts only the forests such that for every node marked with c, its right sibling is labeled with $\#$.

Cases 7, 8, 9: The formulae of form $\forall x \exists y\ \alpha(x) \rightarrow (\beta(y) \wedge x \not\sim y \wedge \epsilon(x,y))$ where $\epsilon(x,y)$ is one of $\mathsf{E}_{\rightarrow}(y,x)$, $\mathsf{E}_{\downarrow}(x,y)$, $\mathsf{E}_{\downarrow}(y,x)$ are treated similarly.

Case 10: $\forall x \exists y\ \alpha(x) \rightarrow (\beta(y) \wedge x \not\sim y \wedge \mathsf{E}_{\rightrightarrows}(x,y))$. This formula expresses that every α-node must have a β-node as a right sibling, but not as its next-sibling, and with a different data value.

Let X be an horizontal path. Let y be the rightmost β-node of X and d be its data value. Consider now an α-node x of X with a data value different from d. Then either x is at the left of the previous-sibling of y, and y can serve as the desired witness, or x has no witness and the formula is false.

The transducer $\mathcal{A}$, for each horizontal path X containing an α-node, marks its rightmost β-node y with a new color c', guesses all the nodes of X with the same data value as y and marks them with a new color c. Then it checks that every unmarked α-node of X occurs at the left of the previous-sibling of y. The forest automaton $\mathcal{B}$ checks that the guesses are correct as in Remark 1: for each horizontal paths, either all elements are marked with c or c', or none.

Cases 11, 12, 13: The constructions for the formulae $\forall x \exists y\ \alpha(x) \rightarrow (\beta(y) \wedge x \not\sim y \wedge \epsilon(x,y))$ where $\epsilon(x,y)$ is one of $\mathsf{E}_{\rightrightarrows}(y,x))$, $\mathsf{E}_{\downdownarrows}(x,y))$, and $\mathsf{E}_{\downdownarrows}(y,x))$ are similar.

Case 14: $\forall x \exists y\ \alpha(x) \rightarrow (\beta(y) \wedge x \not\sim y \wedge x \not\approx y)$. This formula expresses that every α-node must have a incomparable β-node with a different data value.

Subcase 14.1: There exists two β-nodes that are incomparable.

Let y_1 and y_2 be two incomparable β-nodes and let z be their least common ancestor. Using the same reasoning as in subcase 4.1, we can choose y_1 and y_2 such that none of the β-nodes is incomparable with z or a sibling of z. Let z_1 and z_2 be the children of z that are the ancestors of y_1 and y_2 respectively. By construction, $z_1 \neq z_2$. Using the same trick as in subcase 4.1, we can ensure that if $y_1 = z_1$ then there is no β-node incomparable with z_2, and if $y_2 = z_2$ then there is no β-node incomparable with z_1. Moreover, we cannot have at the same time $y_1 = z_1$ and $y_2 = z_2$. Recall that all these properties can be tested by a forest automaton. Let d_1 and d_2 be the respective data values of y_1 and y_2 (possibly $d_1 = d_2$).

Consider now an α-node x whose data value is neither d_1 nor d_2. If x is incomparable with z or a sibling of z, then y_1 is a witness for x. If x is an ancestor of z then by hypothesis there is no β-node incomparable with x and hence the formula is false. Assume now that x is in the subtree rooted at z. If $x = z_1$ and $y_2 \neq z_2$, then y_2 is a β-node incomparable with x with a different data value, hence a witness for x in the formula. If $x = z_1$ and $y_2 = z_2$, then by hypothesis, there is no β-node incomparable with x in the subtree of z, and since there are neither β-nodes incomparable with x outside the subtree of z, the formula must be false. If $x \neq z_1$ and x is a descendant of z_1, then y_2 is a witness for x. The cases where x is a descendant of z_2 are symmetric. In the remaining cases, x is in the subtree of z and not a descendant of z_1 or z_2. In each of these cases, either y_1 or y_2 is a witness for x.

With this discussion in mind, this case can be solved as follows: The transducer $\mathcal{A}$ guesses the nodes of y_1, y_2, z_1, z_2 and z and check that they satisfy the appropriate properties. Moreover, $\mathcal{A}$ guesses whether $d_1 = d_2$ and marks accordingly the data values of z_1 and z_2 with one or two new colors. The forest automaton $\mathcal{B}$ will then check that the data values are marked appropriately, as in Remark 1. Moreover $\mathcal{A}$ checks that for every marked α-node, there exists a β-node making the formula true. It remains for $\mathcal{A}$ to check that no unmarked α-node occurs above z, that if $y_1 = z_1$ then z_2 is not an unmarked α-node and that if $y_2 = z_2$ then z_1 is not an unmarked α-node.

Subcase 14.2: There are no two β-nodes that are incomparable.

Let y be an β-node such that no β-node is a descendant of y. By hypothesis, all β-nodes are either ancestors or siblings of y. Let d be the data value of y. We distinguish between several subcases depending on whether there are β-nodes that are siblings of y or not.

If there exists a β-node y' that is a sibling of y, let d' be its data value (possibly $d = d'$). Consider an α-node x whose data value is neither d nor d'. If x is incomparable with y, then we have have a witness (y) for x. If x is an ancestor or a sibling of y, then the formula cannot be true, because by hypothesis every β-node cannot be incomparable with x. If x is a descendant of y, then y' makes the formula true for that x.

Consider now the case where there are no β-node that are sibling of y. Note that y can have β-nodes among its ancestors. Let x be a α-node that has data value different from d. If x is not incomparable with y then the formula must be false. Otherwise, y is a witness for x.

The transducer $\mathcal{A}$ guesses the β-node y and marks its data value using a new color. Then it checks whether there is an β-node y' that is a sibling of y. If yes, it guesses whether the value at y' is the same as the value at y or not, and marks the data value of y' using a new color. The forest automaton $\mathcal{B}$ will then check that the data values are marked appropriately. For marked α-nodes, $\mathcal{A}$ checks the regular property making the formula true. It now remains for $\mathcal{A}$ to check, in both cases, that every unmarked α-node x satisfy the appropriate condition described above, *i.e.* that x is incomparable with y or a descendant of y if there exists a sibling y' and that x is incomparable with y otherwise.

Case 15: $\forall x \exists y\ \alpha(x) \rightarrow \mathit{false}$. It is sufficient to test with $\mathcal{A}$ that no α-node is present in the forest.

3 From DTA$^{\#}$ to EBVASS

In this section we show that the emptiness problem of DTA$^{\#}$ can be reduced to the reachability of a counter tree automata model that extends BVASS, denoted EBVASS. It runs on binary trees over a finite alphabet.

We start by defining the counter tree automata model and then we present the reduction.

3.1 Definition of EBVASS

An EBVASS is a tree automaton equipped with counters. It runs over binary trees over a finite alphabet $\mathbb{A}$. It can increase or decrease its counters but cannot perform a zero test. For BVASS, when going up in the tree, the new value of each counter is the sum of its values at the left and right child. An EBVASS can change this behavior using simple arithmetical constraints.

More precisely an EBVASS is a tuple $(Q, \mathbb{A}, q_0, k, \delta, \chi)$ where $\mathbb{A}$ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, $k \in \mathbb{N}$ is the number of counters, χ is a finite set of constraints of the form $\mathsf{C}_{i_1} \ominus \mathsf{C}_{i_2} \to \mathsf{C}_i$with $1 \leq i_1, i_2, i \leq k$, and δ is the set of transitions which are of two kinds: ϵ-transitions (subset denoted δ_ϵ) and up-transitions (subset denoted δ_{u}).

An ϵ-transition is an element of $(Q \times \mathbb{A}) \times (Q \times U)$ where $U = \{I_i, D_i \ : \ 1 \leq i \leq k\}$ is the set of possible counter updates: D_i stands for *decrement counter* i and I_i stands for *increment counter* i. We view each element of U as a vector over $\{-1, 0, 1\}^k$ with only one non-zero position. An up-transition is an element of $(Q \times \mathbb{A} \times Q) \times Q$.

Informally, an ϵ-transition may change the current state and increment or decrement one of the counters. An up-transition depends on the label of the current node and, when the current node is an inner node, on the states reached at its left and right child. It defines a new state and the new value of each counter is the sum of the values of the corresponding counters of the children. Moreover, the behavior of up-transitions can be modified by the constraints χ. Informally a constraint of the form $\mathsf{C}_{i_1} \ominus \mathsf{C}_{i_2} \to \mathsf{C}_i$ modifies this process as follows: before performing the addition of the counters, two numbers n_1 and n_2 are guessed (they possibly have value 0), the counter i_1 of the left child is decreased by n_1, the counter i_2 of the left child is decreased by n_2, the counter i_1 of the right child is decreased by n_2, the counter i_2 of the right child is decreased by n_1 and, once the addition of the counters has been executed, the counter i is increased by $n_1 + n_2$. Note that n_1 and n_2 must be so that all intermediate values remain positive. We guess two values n_1 and n_2 instead of just one in order to capture the commutativity of #-stuttering languages (rule 4, page 5). In other words we cannot distinguish from the left child and the right child when applying the rule. We now make this more precise.

A *configuration* of an EBVASS is a pair (q, v) where $q \in Q$ and v is a valuation of the counters, seen as a vector of $\mathbb{N}^k$. The initial configuration is (q_0, v_0) where v_0 is the function setting all counters to 0. There is an *ϵ-transition* of label a from (q, v) to (q', v') if $(q, a, q', u) \in \delta_\epsilon$ and $v' = v + u$ (in particular this implies that $v + u \geq 0$). We write $(q, v) \xrightarrow{a}_\epsilon (q', v')$, if (q', v') can be reached from (q, v) via a finite sequence of ϵ-transitions of label a.

Given a binary tree $\boldsymbol{a} \in \mathsf{Trees}(\mathbb{A})$, a *run* ρ of a EBVASS is a function from nodes of $\boldsymbol{a}$ to configurations verifying for all leaf x, $\rho(x) = (q_0, v_0)$ and for all nodes x, x_1, x_2 of $\boldsymbol{a}$ with x_1 and x_2 the left and right child of x, and $\rho(x) = (q, v), \rho(x_1) = (q_1, v_1), \rho(x_2) = (q_2, v_2)$ there exists $(q_1', v_1'), (q_2', v_2')$ such that:

1. $(q_1, v_1) \xrightarrow{\boldsymbol{a}(x_1)}_\epsilon (q_1', v_1'), (q_2, v_2) \xrightarrow{\boldsymbol{a}(x_2)}_\epsilon (q_2', v_2')$,
2. $(q_1', \boldsymbol{a}(x), q_2', \ q) \in \delta_{\mathrm{u}}$,
3. for each constraint $\theta \in \chi$ of the form $\mathsf{C}_{i_1} \ominus \mathsf{C}_{i_2} \to \mathsf{C}_i$ there are two numbers n_θ^1 and n_θ^2 (they may be 0) and vectors $u_{\theta,1}, u_{\theta,2}, u_\theta \in \mathbb{N}^k$, having n_θ^1 and n_θ^2 at positions i_1, i_2 for $u_{\theta,1}$, having n_θ^2 and n_θ^1 at positions i_1, i_2 for $u_{\theta,2}$, $n_\theta^1 + n_\theta^2$ at position i for u_θ and all other positions set to zero,
4. $v_1'' = v_1' - \sum_{\theta \in \chi} u_{\theta,1} \geq 0$, and $v_2'' = v_2' - \sum_{\theta \in \chi} u_{\theta,2} \geq 0$, and $v = v_1'' + v_2'' + \sum_{\theta \in \chi} u_\theta$.

Without the constraints of χ we have the usual notion of BVASS [15].
The *reachability* problem for an EBVASS, on input $q \in Q$, asks whether there is a tree and a run on that tree reaching the configuration (q, v_0) at its root.

Example 1. *As a toy example, consider an* EBVASS $\mathcal{A}$ *with two counters characterizing the set of trees of the form* $c(\ldots c(\boldsymbol{t}_a, \boldsymbol{t}_b), d), \ldots d)$*, where all nodes of* $\boldsymbol{t}_a$ *(resp.* $\boldsymbol{t}_b$*) except leaves, are labeled by* a *(resp.* b*)*

and with the same number of a, b and c. In $\boldsymbol{t}_a$, $\mathcal{A}$ will increment C_1 each time it encounters an a-node, with up-transitions of the form $(q_\alpha, a, q_\alpha,\ q'_a)$, where q_α is either q_0 (the initial state) or q_a, and one ϵ-transition $(q'_a, a,\ q_a, I_1)$. It proceeds similarly in $\boldsymbol{t}_b$ with b-nodes. The up-transition $(q_a, c, q_b,\ q')$ together with the constraint $\mathsf{C}_1 \ominus \mathsf{C}_1 \to \mathsf{C}_2$ will guess an integer n, decrease C_1 by n in each child, sum the remainders in C_1 and add n to C_2 (which was null before the transition). Then, $\mathcal{A}$ decrements C_2 each time it meets a c, with one ϵ-transition $(q', c,\ q, D_2)$ and one up-transition $(q, c, q_0,\ q')$. It reaches the root in configuration (q, v_0) iff the tree has the expected form.

This language can be characterized by a BVASS as well. However, it does not seem possible in general to simulate directly a constraint $\mathsf{C}_{i_1} \ominus \mathsf{C}_{i_2} \to \mathsf{C}_i$ with BVASS transitions. One could imagine using an arbitrary number of ϵ-transitions decreasing the counters i_1 and i_2 while increasing counter i, after the merging operation summing up the counters. However, it is not clear how to do this while preserving the positiveness of the corresponding decrements before the merge (Step 4 above).

3.2 Reduction from DTA$^\#$ to EBVASS

Theorem 4. *The emptiness problem for* DTA$^\#$ *reduces to the reachability problem for* EBVASS.

Proof. We show that given a DTA$^\#$ $\mathcal{D}$, one can construct an EBVASS $\mathcal{E}$ with a distinguished state q such that for all $\boldsymbol{a} \in \mathsf{Forests}(\mathbb{A})$, there is a run of $\mathcal{E}$ on $fcns(\boldsymbol{a})$ reaching (q, v_0) at its root iff $\boldsymbol{a} \otimes \boldsymbol{d}$ is accepted by $\mathcal{D}$ for some $\boldsymbol{d}$.

Let $\mathcal{A} = (Q_\mathcal{A}, \mathbb{A}, \mathbb{B}, F_\mathcal{A}, \Delta_\mathcal{A})$ and $\mathcal{B} = (Q_\mathcal{B}, \mathbb{B}_\#, F_\mathcal{B}, \Delta_\mathcal{B})$ be the $fcns$ views of the two components of $\mathcal{D}$, computing on binary trees. Here $Q_\mathcal{A}$ and $Q_\mathcal{B}$ ($F_\mathcal{A}$, $F_\mathcal{B}$) are the respective state sets (final state sets) of $\mathcal{A}$ and $\mathcal{B}$, $\mathbb{A}$ is the input alphabet of $\mathcal{A}$, $\mathbb{B}$ is the output alphabet of $\mathcal{A}$ and input alphabet of $\mathcal{B}$ (with the additional symbol $\#$), and $\Delta_\mathcal{A}$, $\Delta_\mathcal{B}$ are the sets of transitions. We will use the notation $(r_1, a, r_2,\ r, b)$, for a transition of $\mathcal{A}$ from the states r_1, r_2 in the left and right child of a node of label a, renaming this node with b and moving up with state r, and the notation $(p_1, a, p_2) \to p$ for a transition of $\mathcal{B}$ from the states p_1, p_2 in the left and right child of a node of label a, moving up with state p.

The automaton $\mathcal{B}$ is assumed deterministic and complete, *i.e.* for every $\boldsymbol{b} \in \mathsf{Trees}(\mathbb{B}_\#)$, $\mathcal{B}$ evaluates into exactly one state of $Q_\mathcal{B}$, and $\#$-stuttering. We recall that it implies a distinction in the states of $\mathcal{B}$ between $\#$-states and non-$\#$-states, and the existence of a $\#$-state $p_\# \in Q_\mathcal{B}$ on which $\mathcal{B}$ evaluates the tree with a single node labeled with $\#$. In the following, we write explicitly the set of states of $\mathcal{B}$ as $Q_\mathcal{B} = \{p_\#, p_1, \ldots, p_m\}$.

For any data tree $\boldsymbol{t} \in \mathsf{Trees}(\mathbb{B}_\# \times \mathbb{D})$, and any data value d occurring in $\boldsymbol{t}$, the state of $Q_\mathcal{B}$ corresponding to the evaluation of $\mathcal{B}$ on the class forest $\boldsymbol{t}[d]$ is called *the $\mathcal{B}$-state associated to d in $\boldsymbol{t}$*. When d is the data value at the root of $\boldsymbol{t}$, this state is called the *$\mathcal{B}$-state* of $\boldsymbol{t}$. Note that for all $\boldsymbol{t}$, the $\mathcal{B}$-state of $\boldsymbol{t}$ exists and is unique, since $\mathcal{B}$ is assumed deterministic and complete, and that it is always a non-$\#$-state.

We now construct the expected EBVASS $\mathcal{E} = (Q, \mathbb{A}, q_0, k, \delta, \chi)$ with $k = m = |Q_\mathcal{B}| - 1$. Each configuration of $\mathcal{E}$ is of the form (q, v) where $q \in Q$ and v is a function associating to each counter its value in $\mathbb{N}$. We set $Q = Q_\mathcal{A} \times Q_\mathcal{B} \times Q_0$, where Q_0 is a finite set of auxiliary control states. The first and second components of a state $q \in Q$ are respectively called the $\mathcal{A}$-state and the $\mathcal{B}$-state of q. The transitions of the EBVASS $\mathcal{E}$ are constructed in order to ensure the following invariant:

($\star$) $\mathcal{E}$ reaches the configuration (q, v) at the root of a tree $\boldsymbol{a} \in \mathsf{Trees}(\mathbb{A})$ iff there exists a data tree $\boldsymbol{t} = \boldsymbol{a} \otimes \boldsymbol{d}$ and a possible output $\boldsymbol{b} \in \mathsf{Trees}(\mathbb{B})$ of $\mathcal{A}$ on $\boldsymbol{a}$ witnessed by a run of $\mathcal{A}$ whose value at the root of $\boldsymbol{a}$ is the $\mathcal{A}$-state of q, and moreover for all i, $1 \leq i \leq k$, v_i is the number of data values having p_i as associated $\mathcal{B}$-state in $\boldsymbol{b} \otimes \boldsymbol{d}$.

A consequence of ($\star$) is that: ($\diamond$) there is only one non-#-state $p_i \in Q_\mathcal{B}$ such that $v_i \neq 0$, and actually $v_i = 1$. We will refer to this state p_i as the *$\mathcal{B}$-state* of v, and the construction of $\mathcal{E}$ will ensure that p_i is also the $\mathcal{B}$-state of q.

The property ($\star$) clearly implies the expected result.

Notice that the property ($\star$) is invariant under permutations of $\mathbb{D}$. Hence if a tree $\boldsymbol{d}$ witnesses the property ($\star$), then any tree $\boldsymbol{d}'$ constructed from $\boldsymbol{d}$ by permuting the data values is also a witness for ($\star$). This observation will be useful for showing the correctness of the construction of $\mathcal{E}$.

Intuitively $\mathcal{E}$ works as follows. Let x be a node of $\boldsymbol{t}$ whose label is $a \in \mathbb{A}$. From configurations (q_1, v_1) and (q_2, v_2) reached by $\mathcal{E}$ respectively at the left and right children of x, x_1 and x_2, $\mathcal{E}$ does the following: it simulates $\mathcal{A}$ using a and the $\mathcal{A}$-states of q_1 and q_2, obtaining the new $\mathcal{A}$-state for x together with the relabeling $b \in \mathbb{B}$ of x. Now, $\mathcal{E}$ needs to simulate $\mathcal{B}$ on all data values, updating the counters appropriately. In order to do this, $\mathcal{E}$ has to "guess" the data value d of $\boldsymbol{d}$ at x, assuming that it has already guessed the subtrees $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ of $\boldsymbol{d}$ at x_1 and x_2. Of course, $\mathcal{E}$ cannot make a guess among an infinite set of choices, but it is enough for $\mathcal{E}$ to guess whether the data value is a new data value or not, *i.e.* whether d is present in $\boldsymbol{d}_1$ or $\boldsymbol{d}_2$. In the case d is already present, $\mathcal{E}$ guesses the $\mathcal{B}$-state associated to d in the subtrees $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ of $\boldsymbol{t}$ rooted at x_1 and x_2. Note that because $\mathcal{B}$ is deterministic and complete, these guesses imply whether d is the data value at x_1 or/and at x_2 in $\boldsymbol{d}$. Moreover $\mathcal{E}$ guesses whether the data value at x_1 occurs in the subtree $\boldsymbol{d}_2$ or not, and vice-versa for the data value at x_2. This makes finitely many guesses and we will see that this is enough for $\mathcal{E}$ to cover the computations on the class forests $(\boldsymbol{b} \otimes \boldsymbol{d})[e]$ for all data values e.

Let us now provide more details.

The simulation of $\mathcal{A}$ is straightforward: We ensure that for every ϵ-transition $(q, a,\ q', u)$ of $\mathcal{E}$, the $\mathcal{A}$-states of q and q' coincide, and that for every up-transition $(q_1, a, q_2,\ q)$ of $\mathcal{E}$, there exists a transition of $\mathcal{A}$ of the form $(r_1, a, r_2,\ r, b)$, for some $b \in \mathbb{B}$ such that r_1, r_2 and r are the respective $\mathcal{A}$-states of q_1, q_2 and q.

The "guesses" of $\mathcal{E}$ about the equalities between the data values at x, x_1 and x_2 are reflected by its transitions. $\mathcal{E}$ has a set of transitions that is associated to each case and will maintain the property $(\star)$, where q_1, a, q_2 and q are related as above in order to simulate $\mathcal{A}$.

Before embarking to the case analysis, a few words about correctness of the construction. It follows a generic construction that we now describe. One direction will be always immediate: if $\mathcal{D}$ has an accepting run on $\boldsymbol{t} = \boldsymbol{a} \otimes \boldsymbol{d}$ then $\mathcal{E}$ has an accepting run on $\boldsymbol{a}$ satisfying $(\star)$ because the $\mathcal{A}$-state and the $\mathcal{B}$-state of $\mathcal{E}$ always reflect the behavior of $\mathcal{D}$. We will therefore always omit this direction. The difficulty is to show the converse, i.e. from a run of $\mathcal{E}$ on $\boldsymbol{a}$ construct a tree $\boldsymbol{d}$ such that $\mathcal{D}$ has an accepting run on $\boldsymbol{a} \otimes \boldsymbol{d}$. It is shown by induction on the depth of the tree based on the inductive hypothesis $(\star)$.

Let us assume that $\mathcal{E}$ reached the configuration (q, v) at the root x of a tree $\boldsymbol{a}$.

If x is a leaf node, then by definition of EBVASS, q is the initial state q_0 of $\mathcal{E}$ and $v = v_0$ (the function setting all counters to 0), hence $(\star)$ holds.

If x is an inner node, then let $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ be the subtrees of $\boldsymbol{a}$ rooted at the left and right children of x, called x_1 and x_2. By induction on $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$, we have trees $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ such that there is a run of $\mathcal{D}$ on $\boldsymbol{t}_1 = \boldsymbol{a}_1 \otimes \boldsymbol{d}_1$ and $\boldsymbol{t}_2 = \boldsymbol{a}_2 \otimes \boldsymbol{d}_2$ satisfying $(\star)$. For each case, we will construct from $\boldsymbol{d}_1$, $\boldsymbol{d}_2$ and the transition of $\mathcal{E}$ giving (q, v) a tree $\boldsymbol{d} = d(\boldsymbol{d}_1, \boldsymbol{d}_2)$ such that $\mathcal{D}$ also has the expected run on $\boldsymbol{t} = \boldsymbol{a} \otimes \boldsymbol{d}$.

We set χ as the set of constraints of the form $\mathsf{C}_{j_1} \ominus \mathsf{C}_{j_2} \to \mathsf{C}_j$ such that there exists a transition $(p_{j_1}, \#, p_{j_2}) \to p_j$ of $\mathcal{B}$ where p_{j_1}, p_{j_2}, and p_j are $\#$-states in $Q_{\mathcal{B}} \setminus \{p_\#\}$. Note that the commutativity rule in the definition of $\#$-stuttering languages implies that whenever we have a constraint $\mathsf{C}_{j_1} \ominus \mathsf{C}_{j_2} \to \mathsf{C}_j$ then both $(p_{j_1}, \#, p_{j_2}) \to p_j$ and $(p_{j_2}, \#, p_{j_1}) \to p_j$ are rules of $\mathcal{B}$.

The transitions of $\mathcal{E}$ will take care of the simulation of the other transitions of $\mathcal{B}$, i.e. those involving at least one non-$\#$-state. In the transitions constructed in the following cases, the symbols $a \in \mathbb{A}$ and $b \in \mathbb{B}_\#$ are always assumed to follow the conditions described above for the simulation of $\mathcal{A}$.

1. *$\mathcal{E}$ guessed that the data value of the current node is equal to the data value of both its children.*
To handle this case, for each transition $\tau = (p_{i_1}, b, p_{i_2}) \to p_i$ of $\mathcal{B}$, where none of p_{i_1}, p_{i_2}, p_i are $\#$-states, $\mathcal{E}$ has the following transitions:

ϵ-transitions:

from a state q_1 of $\mathcal{B}$-state p_{i_1} it decreases counter i_1 and moves to a state q_τ^1
from a state q_2 of $\mathcal{B}$-state p_{i_2} it decreases counter i_2 and moves to state q_τ^2
from a state q_τ it increases counter i and move to a state q of $\mathcal{B}$-state p_i

up-transition: $(q_\tau^1, a, q_\tau^2, q_\tau)$.

The state q_τ^1 (resp. q_τ^2, q_τ) differs from q_1 (resp. q_2, q) only by its component in Q_0, that contains τ. We shall use the same convention for the states introduced in the following construction cases.

Correctness. Let us show that using an up-transition $(q^1_\tau, a, q^2_\tau, q_\tau)$ at the root of $\boldsymbol{a} \in \mathsf{Trees}(\mathbb{A})$ yields the desired property for the tree $\boldsymbol{t} = \boldsymbol{a} \otimes \boldsymbol{d}$ constructed by the generic computation. This up-transition can only occur if we had ϵ-transitions from q_1 to q^1_τ in the left subtree and from q_2 to q^2_τ in the right subtree where p_{i_1} and p_{i_2} are the $\mathcal{B}$-states of q_1 and q_2. Let x be the root of the tree $\boldsymbol{a} \otimes \boldsymbol{d}$ where this transition occurred. We have $\boldsymbol{a} = a(\boldsymbol{a}_1, \boldsymbol{a}_2)$. By induction hypothesis we have trees $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ such that there is a run of $\mathcal{D}$ on $\boldsymbol{t}_1 = \boldsymbol{a}_1 \otimes \boldsymbol{d}_1$ and $\boldsymbol{t}_2 = \boldsymbol{a}_2 \otimes \boldsymbol{d}_2$ satisfying $(\star)$.

We first apply a bijection on the labels of $\boldsymbol{d}_1$ in order for the data value of its root to match the one of the root of $\boldsymbol{d}_2$. Let d be this data value.

For each constraint $\theta = \mathsf{C}_{k_1} \ominus \mathsf{C}_{k_2} \to \mathsf{C}_k \in \chi$ we let n^1_θ and n^2_θ be the number used by the run of $\mathcal{E}$ when using the above up-transition. By induction hypothesis $(\star)$, and semantics of the constraints (making sure the counters are big enough) there are at least n^1_θ (resp. n^2_θ) distinct data values different from d (because the up-transition is applied *after* we decreased the counter k_1 by n^1_θ) in $\boldsymbol{d}_1$ having p_{k_1} (resp. p_{k_2}) as associated $\mathcal{B}$-state in $\boldsymbol{t}_1 = \boldsymbol{a}_1 \otimes \boldsymbol{d}_1$, and similarly for $\boldsymbol{t}_2 = \boldsymbol{a}_2 \otimes \boldsymbol{d}_2$. We pick such data values in each subtrees and call them the data values associated to θ. We do this for all constraints θ and we choose the associated data values such that they are all distinct. We now apply to $\boldsymbol{d}_2$ a permutation on the data values such that for all θ the data values associated to θ in $\boldsymbol{d}_2$ are identified with the ones for $\boldsymbol{d}_1$ and such that all other data values are distinct. In order to simplify the notations we call the resulting tree also $\boldsymbol{d}_2$. We then set $\boldsymbol{d}$ as $d(\boldsymbol{d}_1, \boldsymbol{d}_2)$ and $\boldsymbol{t}' = \boldsymbol{b} \otimes \boldsymbol{d}$ where $\boldsymbol{b}$ is an output of $\mathcal{A}$ on $\boldsymbol{a}$ compatible with the transition.

Let e be a data value occurring in $\boldsymbol{d}$.

If $e = d$, the root symbol of the class forest $\boldsymbol{t}'[e]$ is a and the counter i is increased by 1 by the last ϵ-transition. By induction hypothesis and its consequence $(\diamond)$, $v_i = 1$ and for all other non-#-state the corresponding value via v will be 0. Hence p_i is the new $\mathcal{B}$-state of v. It is also the $\mathcal{B}$-state of q by construction.

If $e \neq d$ we consider 3 subcases. If e occurs in both $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ then the class forest $\boldsymbol{t}'[e]$ has the form $\#(\boldsymbol{s}_1, \boldsymbol{s}_2)$ for some forests $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ containing each at least one symbol other than $\#$ (not at the root node). Let p_{j_1} and p_{j_2} be the states reached by $\mathcal{B}$ when evaluating $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$. They are the $\mathcal{B}$-states associated to e in $\boldsymbol{t}'_1$ and $\boldsymbol{t}'_2$, (resp. the left- and right subtrees of $\boldsymbol{t}'$), and both are $\#$-states in $Q_\mathcal{B} \setminus \{p_\#\}$. By construction of $\boldsymbol{d}$, there are at least $n_\theta = n^1_\theta + n^2_\theta$ such data values e, where $\theta = \mathsf{C}_{j_1} \ominus \mathsf{C}_{j_2} \to \mathsf{C}_j$ and p_j is the unique state of $\mathcal{B}$ such that $(p_{j_1}, \#, p_{j_2}) \to p_j$ is a transition of $\mathcal{B}$. These n_θ data values will contribute to an increase of v_j by n_θ as expected.

Assume now that e occurs in $\boldsymbol{d}_1$ but not in $\boldsymbol{d}_2$ (the remaining case being symmetrical). Then $\boldsymbol{t}'[e]$ has the form $\#(\boldsymbol{s}_1, \boldsymbol{s}_2)$ where $\boldsymbol{s}_1$ contains at least of symbol other than $\#$ (not at root node), and all nodes of $\boldsymbol{s}_2$ are labeled $\#$. By the hypothesis that $\mathcal{B}$ is $\#$-stuttering, the $\mathcal{B}$-state associated to e in $\boldsymbol{t}'$ is the same as the one associated to e in $\boldsymbol{t}'_1$, and the $\mathcal{B}$-state associated to e in $\boldsymbol{t}'_2$ is $p_\#$. This is consistent with the behavior of $\mathcal{E}$ that propagates upward the value of the counter corresponding to this state, after applying the constraints. Altogether this shows that $\boldsymbol{t} = \boldsymbol{a} \otimes \boldsymbol{d}$ verifies $(\star)$.

2. *$\mathcal{E}$ guessed that the data value d_1 of the current node is equal to the data value of its left child but different from the data value of its right child. Moreover $\mathcal{E}$ guessed that the $\mathcal{B}$-state associated to d_1 in the right subtree is p_{k_2}, and that the data value d_2 of the right child of the current node also appear in its left subtree with p_{k_1} as associated $\mathcal{B}$-state. Note that both p_{k_1} and p_{k_2} must be $\#$-states in $Q_\mathcal{B} \setminus \{p_\#\}$.*
To handle this case for all transitions $\tau = (p_{i_1}, b, p_{k_2}) \to p_i$ and $\tau' = (p_{k_1}, b, p_{i_2}) \to p_j$ of $\mathcal{B}$, where none of p_{i_1}, p_{i_2}, p_i are $\#$-states but p_j (like p_{k_1} and p_{k_2}) are $\#$ states, $\mathcal{E}$ has the following transitions:

ϵ-transitions:

from a state q_1 of $\mathcal{B}$-state p_{i_1} it decreases the counters i_1 and k_1 and moves to state $q^1_{\tau,\tau'}$
from a state q_2 of $\mathcal{B}$-state p_{i_2} it decreases the counters i_2 and k_2 and moves to state $q^2_{\tau,\tau'}$
from a state $q_{\tau,\tau'}$ it increases counters i and j and moves to a state q of $\mathcal{B}$-state p_i

up-transition: $(q^1_{\tau,\tau'}, a, q^2_{\tau,\tau'}, q_{\tau,\tau'})$.

Correctness. We argue as in the previous case with the following modifications. From $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ we first apply a bijection making sure that the data values d_1 and d_2 of their roots are different and that d_1 has $\mathcal{B}$-state p_{k_2} in $\boldsymbol{a}_2 \otimes \boldsymbol{d}_2$ and d_2 has $\mathcal{B}$-state p_{k_1} in $\boldsymbol{a}_1 \otimes \boldsymbol{d}_1$ (where $\boldsymbol{a} = a(\boldsymbol{a}_1, \boldsymbol{a}_2)$).

For each $\theta \in \chi$ we select the associated data values making sure they are neither d_1 nor d_2. The decrement in the ϵ-transitions make sure that this is always possible. We then perform the same identification as in the previous case. The same argument as above shows that the resulting tree $\boldsymbol{d} = d_1(\boldsymbol{d}_1, \boldsymbol{d}_2)$ has the desired properties.

3. *$\mathcal{E}$ guessed that the data value d_1 of the current node is equal to the data value of its left child but different from the data value of its right child. Moreover $\mathcal{E}$ guessed that d_1 also appear in the right subtree of the current node with p_{k_2} as associated $\mathcal{B}$-state, and that the data value d_2 of the right child of the current node does not appear in the left subtree. Note that p_{k_2} must be a $\#$-state in $Q_{\mathcal{B}} \setminus \{p_\#\}$.*
To handle this case for all transitions $\tau = (p_{i_1}, b, p_{k_2}) \to p_i$ and $\tau' = (p_\#, b, p_{i_2}) \to p_j$ of $\mathcal{B}$, where none of p_{i_1}, p_{i_2}, p_i are $\#$-states but p_{k_2} and p_j are $\#$ states, $\mathcal{E}$ has the following transitions:

ϵ-transitions:

from a state q_1 of $\mathcal{B}$-state p_{i_1} it decreases the counter i_1 and moves to state $q^1_{\tau,\tau'}$
from a state q_2 of $\mathcal{B}$-state p_{i_2} it decreases the counter i_2 and k_2 and moves to state $q^2_{\tau,\tau'}$
from a state $q_{\tau,\tau'}$ it increases the counters i and j and moves to a state q of $\mathcal{B}$-state p_i

up-transition: $(q^1_{\tau,\tau'}, a, q^2_{\tau,\tau'}, q_{\tau,\tau'})$.

Correctness. We argue as in the previous cases with the following modifications. From $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ we first apply a bijection making sure that the data values d_1 and d_2 of their roots are different and that d_1 has $\mathcal{B}$-state p_{k_2} in $\boldsymbol{a}_2 \otimes \boldsymbol{d}_2$ and d_2 does not appear in $\boldsymbol{d}_1$.

For each $\theta \in \chi$ we select the associated data values making sure they are neither d_1 nor d_2. The decrement in the ϵ-transitions make sure that this is always possible. We then perform the same identification as in the previous case. As before we show that the resulting tree $\boldsymbol{d} = d_1(\boldsymbol{d}_1, \boldsymbol{d}_2)$ has the desired properties.

4. *$\mathcal{E}$ guessed that the data value d of the current node is different from the ones of its children but appear in both subtrees with p_{k_1} and p_{k_2} as associated $\mathcal{B}$-states. Moreover $\mathcal{E}$ guessed that the data values of both children of the current node are equal. Note that p_{k_1} and p_{k_2} must be #-states in $Q_{\mathcal{B}} \setminus \{p_\#\}$.*
To handle this case for all transitions $\tau = (p_{k_1}, b, p_{k_2}) \to p_i$ and $\tau' = (p_{i_1}, b, p_{i_2}) \to p_j$ of $\mathcal{B}$, where none of p_{i_1}, p_{i_2}, p_i are $\#$-states but p_{k_1}, p_{k_2} and p_j are $\#$ states, $\mathcal{E}$ has the following transitions:

ϵ-transitions:

from a state q_1 of $\mathcal{B}$-state p_{i_1} it decreases the counters i_1 and k_1 and moves to state $q^1_{\tau,\tau'}$
from a state q_2 of $\mathcal{B}$-state p_{i_2} it decreases the counters i_2 and k_2 and moves to state $q^2_{\tau,\tau'}$
from a state $q_{\tau,\tau'}$ it increases the counters i and j and moves to a state q of $\mathcal{B}$-state p_i

up-transition: $(q^1_{\tau,\tau'}, a, q^2_{\tau,\tau'}, q_{\tau,\tau'})$.

Correctness. We argue as in the previous cases with the following modifications.

From $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ we first apply a bijection making sure that the data value d_1 of their roots are equal and that $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ share a common data value $d \neq d_1$ of $\mathcal{B}$-state p_{k_2} in $\boldsymbol{d}_2$ and $\mathcal{B}$-state p_{k_1} in $\boldsymbol{d}_1$.

For each $\theta \in \chi$ we select the associated data values making sure they are neither d_1 nor d. The decrement in the ϵ-transitions make sure that this is always possible. We then perform the same identification as in the previous case. The rest of the argument is similar after setting $\boldsymbol{d} = d(\boldsymbol{d}_1, \boldsymbol{d}_2)$.

5. *$\mathcal{E}$ guessed that the data value d of the current node is different from the ones of its children but appear in both subtrees with p_{k_1} and p_{k_2} as associated $\mathcal{B}$-states. Moreover $\mathcal{E}$ guessed that the data values of both children of the current node are distinct but appear in the other subtree with respective associated $\mathcal{B}$-state p_{ℓ_1} and p_{ℓ_2}. Note that p_{k_1}, p_{k_2}, p_{ℓ_1}, p_{ℓ_2} must be #-states.*

To handle this case for all transitions $\tau = (p_{k_1}, b, p_{k_2}) \to p_i$, $\tau_1 = (p_{i_1}, b, p_{l_1}) \to p_{j_1}$ and $\tau_2 = (p_{l_2}, b, p_{i_2}) \to p_{j_2}$ of $\mathcal{B}$, where none of p_{i_1}, p_{i_2}, p_i are $\#$-states but $p_{k_1}, p_{k_2}, p_{l_1}, p_{l_2}, p_{j_1}, p_{j_2}$ are $\#$ states, $\mathcal{E}$ has the following transitions:

ϵ-transitions:

from a state q_1 of $\mathcal{B}$-state p_{i_1} it decreases the counters i_1, k_1 and l_2 and moves to state q^1_{τ,τ_1,τ_2}
from a state q_2 of $\mathcal{B}$-state p_{i_2} it decreases the counters i_2, k_2 and l_1 and moves to state q^2_{τ,τ_1,τ_2}
from a state q_{τ,τ_1,τ_2} it increases the counters i, j_1 and j_2 and moves to a state q of $\mathcal{B}$-state p_i

up-transition: $(q^1_{\tau,\tau_1\,\tau_2}, a, q^2_{\tau,\tau_1,\tau_2}, q_{\tau,\tau_1,\tau_2})$.

Correctness. We argue as in the previous cases with the following modifications.

From $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ we first apply a bijection making sure that the data values d_1 and d_2 of their roots are distinct and that d_1 has $\mathcal{B}$-state p_{l_1} in $\boldsymbol{d}_2$ and d_2 has $\mathcal{B}$-state p_{l_2} in $\boldsymbol{d}_1$. Moreover $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ share a common data value d distinct from d_1 and d_2 of $\mathcal{B}$-state p_{k_2} in $\boldsymbol{d}_2$ and $\mathcal{B}$-state p_{k_1} in $\boldsymbol{d}_1$.

For each $\theta \in \chi$ we select the associated data values making sure that they are neither d, d_1 nor d_2. The decrement in the ϵ-transitions make sure that this is always possible. We then perform the same identification as in the previous case. The rest of the argument is similar after setting $\boldsymbol{d} = d(\boldsymbol{d}_1, \boldsymbol{d}_2)$.

6. *$\mathcal{E}$ guessed that the data value d of the current node is different from the ones of its children but appear in both subtrees with p_{k_1} and p_{k_2} as associated $\mathcal{B}$-states. Moreover $\mathcal{E}$ guessed that the data value of the right child of the current node appear in its left subtree with p_{ℓ_1} as associated $\mathcal{B}$-state and that the data value of the left child does not appear in the right subtree. Note that p_{k_1}, p_{k_2} and p_{ℓ_1} must be #-states in $Q_{\mathcal{B}} \setminus \{p_\#\}$.*
To handle this case for all transitions $\tau = (p_{k_1}, b, p_{k_2}) \to p_i$, $\tau_1 = (p_{l_1}, b, p_{p_2}) \to p_{j_1}$ and $\tau_2 = (p_{i_1}, b, p_\#) \to p_{j_2}$ of $\mathcal{B}$, where none of p_{i_1}, p_{i_2}, p_i are $\#$-states but $p_{k_1}, p_{k_2}, p_{l_1}, p_{j_1}, p_{j_2}$ are $\#$ states, $\mathcal{E}$ has the following transitions:

ϵ-transitions:

from a state q_1 of $\mathcal{B}$-state p_{i_1} it decreases the counters i_1, k_1 and l_1 and moves to state q^1_{τ,τ_1,τ_2}
from a state q_2 of $\mathcal{B}$-state p_{i_2} it decreases the counters i_2, k_2 and moves to state q^2_{τ,τ_1,τ_2}
from a state q_{τ,τ_1,τ_2} it increases the counters i, j_1 and j_2 and moves to a state q of $\mathcal{B}$-state p_i

up-transition: $(q^1_{\tau,\tau_1\,\tau_2}, a, q^2_{\tau,\tau_1,tau_2}, q_{\tau,\tau_1,\tau_2})$.

Correctness. We argue as in the previous cases with the following modifications.

From $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ we first apply a bijection making sure that the data values d_1 and d_2 of their roots are distinct and that d_1 does not appear in $\boldsymbol{d}_2$ and d_2 has $\mathcal{B}$-state p_{l_1} in $\boldsymbol{d}_1$. Moreover $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ share a common data value d distinct from d_1 and d_2 of $\mathcal{B}$-state p_{k_2} in $\boldsymbol{d}_2$ and $\mathcal{B}$-state p_{k_1} in $\boldsymbol{d}_1$.

For each $\theta \in \chi$ we select the associated data values making sure that they are neither d, d_1 nor d_2. The decrement in the ϵ-transitions make sure that this is always possible. We then perform the same identification as in the previous case. The rest of the argument is similar after setting $\boldsymbol{d} = d(\boldsymbol{d}_1, \boldsymbol{d}_2)$.

7. *$\mathcal{E}$ guessed that the data value d of the current node is different from the ones of its children but appears in both subtrees with p_{k_1} and p_{k_2} as associated $\mathcal{B}$-states. Moreover it guessed that the data values of both children of the current node do not appear elsewhere. Note that p_{k_1}, p_{k_2} must be #-states in $Q_{\mathcal{B}} \setminus \{p_\#\}$.*
To handle this case for all transitions $\tau = (p_{k_1}, b, p_{k_2}) \to p_i$, $\tau_1 = (p_\#, b, p_{p_2}) \to p_{j_1}$ and $\tau_2 = (p_{i_1}, b, p_\#) \to p_{j_2}$ of $\mathcal{B}$, where none of p_{i_1}, p_{i_2}, p_i are $\#$-states but $p_{k_1}, p_{k_2}, p_{j_1}, p_{j_2}$ are $\#$ states, $\mathcal{E}$ has the following transitions:

ϵ-transitions:

from a state q_1 of $\mathcal{B}$-state p_{i_1} it decreases the counters i_1, k_1 and moves to state q^1_{τ,τ_1,τ_2}
from a state q_2 of $\mathcal{B}$-state p_{i_2} it decreases the counters i_2, k_2 and moves to state q^2_{τ,τ_1,τ_2}
from a state q_{τ,τ_1,τ_2} it increases the counters i, j_1 and j_2 and moves to a state q of $\mathcal{B}$-state p_i

up-transition: $(q^1_{\tau,\tau_1\,\tau_2}, a, q^2_{\tau,\tau_1,tau_2}, q_{\tau,\tau_1,\tau_2})$.

Correctness. We argue as in the previous cases with the following modifications.

From $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ we first apply a bijection making sure that the data values d_1 and d_2 of their roots are distinct and that d_1 does not appear in $\boldsymbol{d}_2$ and d_2 does not appear in $\boldsymbol{d}_1$. Moreover $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ share a common data value d distinct from d_1 and d_2 of $\mathcal{B}$-state p_{k_2} in $\boldsymbol{d}_2$ and $\mathcal{B}$-state p_{k_1} in $\boldsymbol{d}_1$.

For each $\theta \in \chi$ we select the associated data values making sure that they are neither d, d_1 nor d_2. The decrement in the ϵ-transitions make sure that this is always possible. We then perform the same identification as in the previous case. The rest of the argument is similar after setting $\boldsymbol{d} = d(\boldsymbol{d}_1, \boldsymbol{d}_2)$.

8. *$\mathcal{E}$ guessed that the data value d of the current node is different from the ones of its children and does not appear in the subtrees. Moreover $\mathcal{E}$ guessed that the data values of both children of the current node are equal.*
To handle this case for all transitions $\tau = (p_\#, b, p_\#) \to p_i$, $\tau' = (p_{i_1}, b, p_{i_2}) \to p_j$ of $\mathcal{B}$, where none of p_{i_1}, p_{i_2}, p_i are $\#$-states, $\mathcal{E}$ has the following transitions:

ϵ-transitions:

from a state q_1 of $\mathcal{B}$-state p_{i_1} it decreases the counters i_1 and moves to state $q^1_{\tau,\tau'}$
from a state q_2 of $\mathcal{B}$-state p_{i_2} it decreases the counter i_2, and moves to state $q^2_{\tau,\tau'}$
from a state $q_{\tau,\tau'}$ it increases the counters i, and j and moves to a state q of $\mathcal{B}$-state p_i

up-transition: $(q^1_{\tau,\tau'}, a, q^2_{\tau,\tau'}, q_{\tau,\tau'})$.

Correctness. We argue as in the previous cases with the following modifications.

From $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ we first apply a bijection making sure that the data values of their roots are equal (let us call it d_1).

For each $\theta \in \chi$ we select the associated data values making sure it is not d_1. The decrement in the ϵ-transitions make sure that this is always possible. We then perform the same identification as in the previous case. The rest of the argument is similar after setting $\boldsymbol{d} = d(\boldsymbol{d}_1, \boldsymbol{d}_2)$, where d is a fresh new value.

9. *$\mathcal{E}$ guessed that the data value d of the current node is different from the ones of its children and does not appear in both subtrees. Moreover $\mathcal{E}$ guessed that the data values of both children of the current node are distinct but appear in the other subtree with respective associated $\mathcal{B}$-state p_{ℓ_1} and p_{ℓ_2}. Note that p_{ℓ_1}, p_{ℓ_2} must be $\#$-states in $Q_\mathcal{B} \setminus \{p_\#\}$.*
This case is treated as before with the expected transitions.

10. *$\mathcal{E}$ guessed that the data value d of the current node is different from the ones of its children and does not appear in both subtrees. Moreover $\mathcal{E}$ guessed that the data values of the right child of the current node appear in its left subtree with p_{ℓ_1} as associated $\mathcal{B}$-state and that the data value of the left child does not appear in the right subtree. Note that p_{ℓ_1} must be a $\#$-states in $Q_\mathcal{B} \setminus \{p_\#\}$.*
This case is treated as before with the expected transitions.

11. *We omit the symmetric cases.*

□

4 From EBVASS to $\mathsf{FO}^2(<, +1, \sim)$

We show in this section that reachability of EBVASS can be expressed as a sentence of $\mathsf{FO}^2(<, +1, \sim)$. This concludes the loop of reductions, showing that reachability for EBVASS, satisfiability of $\mathsf{FO}^2(<, +1, \sim)$ and emptiness of DTA$^\#$ are equivalent as decision problems. The proof essentially mimics the reduction from BVASS to $\mathsf{FO}^2(<, +1, \sim)$ described in [4] with extra material in order to handle the extra features.

Theorem 5. *The reachability problem for* EBVASS *reduces to the satisfiability problem for $\mathsf{FO}^2(<, +1, \sim)$.*

Proof. Given an EBVASS $\mathcal{E} = (Q, \mathbb{A}, q_0, k, \delta, \chi)$ and a state $q \in Q$, we compute a sentence $\phi \in \mathsf{FO}^2(<, +1, \sim)$ such that ϕ has a model iff the configuration (q, v_0) is reachable (where v_0 is the function setting all counters to 0). We associate to $\mathcal{E}$ the following finite alphabet $\mathbb{A}_\mathcal{E} = \delta \cup \{D_i, I_i \mid 1 \leq i \leq k\} \cup \{T_\theta, L_\theta, R_\theta \mid \theta \in \chi\}$. Intuitively D_i says that the counter i has been decreased, I_i says that the counter i has been increased, and the letters L_θ, R_θ and T_θ will be used to enforce the constraint θ. The formula ϕ will essentially accept binary data trees of $\mathsf{Trees}(\mathbb{A}_\mathcal{E} \times \mathbb{D})$ encoding runs of $\mathcal{E}$ (we will see that ϕ will also accepts some other trees).
We start by the encoding of a single transition $\mu \in \delta$.
If μ is an ϵ-transition then we encode it with two nodes x, y where y is the unique child of x and the label of x is μ while the label of y is D_i (resp. I_i) if μ was decreasing (resp. increasing) counter i.
If μ is an up-transition, then we encode it as a subtree of the following form:

- The root has label μ,
- below the root there is a (vertical) sequence of nodes of arity one whose labels form a word of $\sum_{\theta = \mathsf{C}_{i_1} \ominus \mathsf{C}_{i_2} \to \mathsf{C}_i \in \chi} (I_i T_\theta)^*$, where $\sum$ denotes concatenation,
- the last node of that sequence has arity two and two branches starts from that node: the left one and the right one,
- the sequence of labels of the left branch forms a word of $\sum_{\theta = \mathsf{C}_{i_1} \ominus \mathsf{C}_{i_2} \to \mathsf{C}_i \in \chi} (D_{i_1} L_\theta D_{i_2} R_\theta)^*$,
- the sequence of labels of the right branch forms a word of $\sum_{\theta = \mathsf{C}_{i_1} \ominus \mathsf{C}_{i_2} \to \mathsf{C}_i \in \chi} (D_{i_1} R_\theta D_{i_2} L_\theta)^*$,
- for all θ the number of occurrences of L_θ, R_θ and T_θ are the same.

If all these items but the last one are satisfied, we say that the resulting tree is a *pseudo-encoding* of the up-transition μ. Notice that pseudo-encodings of up-transitions form a regular tree language.
From there, the encoding of a run is obtained in the obvious way by concatenating encodings of transitions.

The formula ϕ essentially describes this construction. It first enforces that the tree has the desired shape:

- The tree is a repetition of a sequence of the form: a pseudo-encoding of one up-transition followed by the encodings of several ϵ-transitions,
- the sequencing is valid: if μ and ν are consecutive transitions in the tree then the starting state of one is the ending state of the other,
- the initial state q_0 can be found at the leaves and the state q is reached at the root.

Note that the above three conditions can be checked by a standard tree automaton over $\mathbb{A}_\mathcal{E}$, and therefore can be expressed in $\mathsf{EMSO}^2(<, +1)$. Therefore, by setting $\mathbb{A} = \mathbb{A}_c \times \mathbb{A}'$ for a suitable $\mathbb{A}'$ matching the existential part of the EMSO formula, the property above can be expressed in $\mathsf{FO}^2(<, +1)$.

The formula ϕ now needs to make sure that no counter ever get negative and that pseudo-encodings of up-transitions are actually real encodings. This is where data values enter into the play: The formula ϕ enforces that

1. no two nodes with label D_i can have the same data value, for $1 \leq i \leq k$,
2. same thing with the labels I_i,
3. for all $i \in [k]$, every node with label D_i has a descendant with label I_i and with the same data value,
4. for all $i \in [k]$, every node with label I_i has an ancestor with label D_i and with the same data value.

These four conditions enforce that the counters never get negative and that they are all set to 0 at the root. It remains to enforce that all pseudo-encodings can be transformed into real encodings. This is done with the following conditions.

5. no two nodes with label T_θ for $\theta \in \chi$ can have the same data value,
6. same thing with the labels L_θ and R_θ,
7. every node with label T_θ has a descendant with label L_θ and a descendant with label R_θ both with the same data value,
8. every node with label L_θ or R_θ has a ancestor with label T_θ and with the same data value,
9. two nodes of label R_θ and L_θ with the same data value cannot be comparable with the ancestor relationship.

It now remains to show that ϕ has the desired property.

Lemma 6. *ϕ has a model iff (q, v_0) is reachable by $\mathcal{E}$.*

Proof. **From reachability to models of ϕ.** Assume that (q, v_0) is reachable and let ρ be a run of $\mathcal{E}$ witnessing this fact. Let $\boldsymbol{a}$ be the tree constructed from ρ by concatenating the sequences of encodings of transitions of ρ as explained above. The binary tree $\boldsymbol{a}$ certainly satisfies the "regular" part of ϕ. We now assign the data values so that the remaining part of ϕ is satisfied. This is done in the obvious way: each time a counter i is decremented, as the resulting value is positive, this means that a matching increment was performed before. Similarly, each time a constraint θ is used in a transition μ, we assign one distinct data value per triple $L_\theta, R_\theta, T_\theta$ occurring in the encoding of μ. The formula was constructed to make the resulting tree a model of ϕ.

From models of ϕ to reachability. Assume now that $\boldsymbol{t} = \boldsymbol{a} \otimes \boldsymbol{d} \models \phi$. Unfortunately, it may happen that $\boldsymbol{a}$ does not encode a run of $\mathcal{E}$ because some section corresponds to a pseudo-encoding of an up-transition, instead of an expected real encoding. However, we show that from $\boldsymbol{t}$ we can construct another tree $\boldsymbol{t}' = \boldsymbol{a}' \otimes \boldsymbol{d}'$ such that $\boldsymbol{t}' \models \phi$ and $\boldsymbol{a}'$ encodes a real run of $\mathcal{E}$.

To see this, let us consider a node x of $\boldsymbol{t}$ with label T_θ, where $\theta = \mathsf{C}_{i_1} \ominus \mathsf{C}_{i_2} \to \mathsf{C}_i$, and let $d = \boldsymbol{d}(x)$. Let x_1 and x_2 be two descendants of x with respective labels L_θ and R_θ and such that $d = \boldsymbol{d}(x_1) = \boldsymbol{d}(x_2)$. Let z be the least common ancestor of x_1 and x_2. The existence of x_1 and x_2 is guaranteed by ϕ (conditions 5-7). The sentence ϕ also ensures that x is an ancestor of z (conditions 8-9). By construction the subtree at z must correspond to a pseudo-encoding of an up-transition μ'.

We now move (down) x and its parent (that must have label I_i) right above z within the coding of μ'. Similarly we move (up) x_1 and its parent (that must have label D_{i_1}) right below z, and similarly for x_2. The reader can verify that the resulting tree is still a model of ϕ: the regular conditions remain obviously satisfied. Conditions 1-4 are still valid because the node of label I_{i_1} matching the parent of y was already below the initial position of y and its new position is upward in the tree. Finally conditions 5-9 remain valid by construction.

Repeating this argument eventually yields a model $\boldsymbol{t}' = \boldsymbol{b} \otimes \boldsymbol{d}'$ of ϕ such that $\boldsymbol{b}$ is a correct sequencing of encodings of transitions a $\mathcal{E}$. This encoding is actually a real runs because conditions 1-4 of ϕ immediately enforces that no counter is ever negative. □

Theorem 5 is now immediate from Lemma 6. □

5 Conclusion

We have seen that satisfiability of $\mathsf{FO}^2(<, +1, \sim)$, emptiness of DTA$^\#$ and reachability of EBVASS are equivalent problems in terms of decidability. The main open problem is of course whether they are all decidable or not.

The use of the EBVASS constraints of the form $\mathsf{C}_{i_1} \ominus \mathsf{C}_{i_2} \to \mathsf{C}_i$ is crucial for the construction of Section 3. Their semantics cannot be directly simulated with the usual BVASS, but it is not clear whether

EBVASS are strictly more expressive that BVASS, and whether this extension is needed in order to capture the expressive power of $\mathsf{FO}^2(<, +1, \sim)$ on data trees.

In our definition of EBVASS the constraints of the form $\mathsf{C}_{i_1} \ominus \mathsf{C}_{i_2} \to \mathsf{C}_i$ have a "commutative" semantics. Without commutativity, i.e. the rule modifies only counter i_1 on the left child and counter i_2 on the right child, then the automata model is more powerful. In order to describes its runs as in the proof of Theorem 5, the logic needs to be able to enforce that a L_θ must be to the left of the R_θ with the same data value. This can be done by adding the document order predicate into the logic. A close inspection to the proof of Theorem 3 and Theorem 4 then shows that the extension of $\mathsf{FO}^2(<, +1, \sim)$ with the document order predicate can be captured by a DTA$^\#$ without the commutativity rule and that such automata can be captured by the non commutative version of EBVASS, hence closing again the loop.

In [2] it was shown that, over data words, the Data Automata model of [5] is more expressive than the Register Automata of [14]. It is not obvious that our automata model DTA$^\#$ extends the expressive power of the straightforward extension of register automata to data trees. This remains to be investigated.

References

[1] R. Alur, P. Černý, and S. Weinstein. Algorithmic analysis of array-accessing programs. *ACM Trans. Comput. Logic*, 13(3):27:1–27:29, Aug. 2012.

[2] H. Björklund and T. Schwentick. On notions of regularity for data languages. *Theoretical Computer Science*, 411(4-5):702–715, 2010.

[3] M. Bojańczyk and S. Lasota. An extension of data automata that captures XPath. In *Symp. on Logic in Computer Science (LICS'10)*, pages 243–252, 2010.

[4] M. Bojańczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. *Journal of the ACM*, 56(3):13:1–13:48, May 2009.

[5] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27, 2011.

[6] B. Bollig, A. Cyriac, P. Gastin, and K. Narayan Kumar. Model checking languages of data words. In *Int. Conf. on Foundations of Software Science and Computation Structures (FoSSaCS'12)*, volume 7213 of *LNCS*, pages 391–405. Springer, 2012.

[7] P. Bouyer. A logical characterization of data languages. *Inf. Process. Lett.*, 84(2):75–85, Oct. 2002.

[8] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on http://tata.gforge.inria.fr. release 12th October 2007.

[9] S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), Apr. 2009.

[10] D. Figueira. Satisfiability of downward XPath with data equality tests. In *Symp. on Principles of Database Systems (PODS'09)*, 2009.

[11] D. Figueira. Forward-XPath and extended register automata on data-trees. In *Intl. Conf. on Database Theory (ICDT'10)*, 2010.

[12] D. Figueira and L. Segoufin. Bottom-up automata on data trees and vertical XPath. In *Theoretical Aspects of Computer Science (STACS'11)*, 2011.

[13] M. Jurdzinski and R. Lazic. Alternating automata on data trees and XPath satisfiability. *ACM Transactions on Computational Logic*, 12(3):19, 2011.

[14] M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

[15] S. Schmitz. On the computational complexity of dominance links in grammatical formalisms. In *Annual Meeting of the Association for Computational Linguistics (ACL'10)*, pages 514–524. ACL Press, 2010.