



HAL
open science

Bottom-up automata on data trees and vertical XPath

Diego Figueira, Luc Segoufin

► **To cite this version:**

Diego Figueira, Luc Segoufin. Bottom-up automata on data trees and vertical XPath. Logical Methods in Computer Science, 2017, 13 (4), 10.23638/LMCS-13(4:5)2017 . hal-01631219

HAL Id: hal-01631219

<https://inria.hal.science/hal-01631219>

Submitted on 20 Feb 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

BOTTOM-UP AUTOMATA ON DATA TREES AND VERTICAL XPATH

DIEGO FIGUEIRA AND LUC SEGOUFIN

CNRS, LaBRI, and University of Edinburgh

INRIA and ENS Cachan, LSV

ABSTRACT. A data tree is a finite tree whose every node carries a label from a finite alphabet and a datum from some infinite domain. We introduce a new model of automata over unranked data trees with a decidable emptiness problem. It is essentially a bottom-up alternating automaton with one register that can store one data value and can be used to perform equality tests with the data values occurring within the subtree of the current node. We show that it captures the expressive power of the vertical fragment of XPath—containing the child, descendant, parent and ancestor axes—obtaining thus a decision procedure for its satisfiability problem.

1. INTRODUCTION

We study formalisms for data trees. A data tree is a finite tree where each position carries a label from a finite alphabet and a *datum* from some infinite domain. This structure has been considered in the realm of semistructured data, timed automata, program verification, and generally in systems manipulating data values. Finding decidable logics or automata models over data trees is an important quest when studying data-driven systems.

In particular data trees can model XML documents. There exist many formalisms to specify or query XML documents. For static analysis or optimization purposes it is often necessary to test whether two properties or queries over XML documents expressed in some formalism are equivalent. This problem usually boils down to a satisfiability question. One such formalism to express properties of XML documents is the logic XPath—the most widely used node selection language for XML. Although satisfiability of XPath in the presence of data values is undecidable, there are some known decidable data-aware fragments [Fig09, Fig10, Fig11, Fig13, BFG08, BMSS09]. Here, we investigate a rather big fragment that nonetheless is decidable. *Vertical XPath* is the fragment that contains all downward and upward axes, but no *horizontal* axis is allowed.

We introduce a novel automaton model that captures vertical XPath. We show that the automaton has a decidable emptiness problem and therefore that the satisfiability problem of vertical XPath is decidable. The **Bottom-Up Data Automata** (or BUDA) are bottom-up alternating tree automata with one register to store and compare data values. Further, these automata can compare the data value currently stored in the register with the data value of a descendant node, reached by a downward path satisfying a given regular property. Hence,

in some sense, it has a two-way behavior. However, they cannot test horizontal properties on the siblings of the tree, like “the root has exactly three children”.

Our main technical result shows the decidability of the emptiness problem of this automaton model. We show this through a reduction to the coverability problem of a well-structured transition system (WSTS [FS01]), that is, the problem of whether, given two elements x, y , an element greater or equal to y can be reached starting from x . Each BUDA automaton is associated with a transition system, in such a way that a derivation in this transition system corresponds to a run of the automaton, and vice-versa. The domain of the transition system consists in the *extended configurations* of the automaton, which contain all the information necessary to preserve from a (partial) bottom-up run of the automaton in a subtree in order to continue the simulation of the run from there. On the one hand, we show that BUDA can be simulated using an appropriate transition relation on sets of extended configurations. On the other hand, we exhibit a well-quasi-order (wqo) on those extended configurations and show that the transition relation is “monotone” relative to this wqo. This makes the coverability problem (and hence the emptiness problem) decidable.

Our decision algorithm is not primitive recursive. However it follows from [FS09] that there cannot be a primitive recursive decision algorithm for vertical XPath.

In terms of expressive power, we show that BUDA can express any node expression of the vertical fragment of XPath. Core-XPath (term coined in [GKP05]) is the fragment of XPath 1.0 that captures its navigational behavior, but cannot express any property involving data. It is easily shown to be decidable. The extension of this language with the possibility to make equality and inequality tests between data values is named Core-Data-XPath in [BMSS09], and it has an undecidable satisfiability problem [GF05]. By “vertical XPath” we denote the fragment of Core-Data-XPath that can only use the downward axes CHILD and DESCENDANT and the upward axes PARENT and ANCESTOR (no navigation among siblings is allowed). It follows from our work that vertical XPath is decidable, settling an open question [BK08, Question 5.10].

Related work. A model of *top-down* tree automata with one register and alternating control (ATRA) is introduced in [JL11], where the decidability of its emptiness problem is proved. ATRA are used to show the decidability of temporal logics extended with a “freeze” operator. This model of automata was extended in [Fig10] with the name ATRA(guess, spread) in order to prove the decidability of the *forward* fragment of XPath, allowing only axes navigating downward or rightward (NEXT-SIBLING and FOLLOWING-SIBLING). The BUDA and ATRA automata models are incomparable: ATRA can express all regular tree languages, but BUDA cannot; while BUDA can express unary inclusion dependency properties (like “the data values labeled by a is a subset of those labeled by b ”), but ATRA cannot. In order to capture vertical XPath, the switch from top-down to bottom-up seems necessary to express formulas with upward navigation, and this also makes the decidability of the emptiness problem considerably more difficult. In [Fig10], the decidability of the forward fragment of XPath is also obtained using a WSTS. However, the automata model and therefore also the transition system derived from it, are significantly different from BUDA and the transition system we derive from it. In particular they cannot traverse a tree in the same way.

Another decidable fragment of XPath on data trees is the *downward* fragment of XPath, strictly contained the vertical fragment treated here, where navigation can be done only through the child and descendant axes. This fragment is known to be decidable, EXPTIME-complete [Fig09, Fig12]. In [Fig13] it is shown the decidability of the satisfiability problem

for XPath where navigation can be done going downwards, rightwards or leftwards in the XML document but using only reflexive-transitive axes. That is, where navigation is done using the XPath axes \downarrow_* , \rightarrow^* , and $^*\leftarrow$. The complexity is of 3EXPSpace, and this in sharp contrast with the fact that having strict (non-reflexive) transitive axes makes the satisfiability problem undecidable.

The paper [BK08] contains a comprehensive survey of the known decidability results for various fragments of XPath, most of which cannot access data values. In the presence of data values, the notable new results since the publication of [BK08] are the downward [Fig09] and the forward [Fig10] fragments, as well as the fragment containing only the successor axis [BMSS09] (the latter is closely related to first-order logic with two variables), or containing reflexive-transitive relations (such as descendant, or the reflexive-transitive closure of the next/previous sibling relation) [Fig11, Fig13]. As already mentioned, this paper solves one of the remaining open problems of [BK08].

Organization. In Section 3 we introduce the BUDA model and we show that it captures vertical XPath in Section 5. The associated well-structured transition system and the proof to show the decidability of its reachability is in Section 4.

This paper is a journal version of [FS11]. Compared to the conference paper, we have modified and simplified significantly the automata model and the associated WSTS.

2. PRELIMINARIES

Basic notation. Let $\wp(S)$ denote the set of subsets of S , and $\wp_{<\infty}(S)$ be the set of *finite* subsets of S . Let $\mathbb{N} = \{0, 1, 2, \dots\}$, $\mathbb{N}_+ = \{1, 2, 3, \dots\}$, and let $[n] := \{1, \dots, n\}$ for any $n \in \mathbb{N}_+$. We fix once and for all \mathbb{D} to be any infinite domain of data values; for simplicity in our examples we will consider $\mathbb{D} = \mathbb{N}$. In general we use letters \mathbb{A}, \mathbb{B} for finite alphabets, the letter \mathbb{D} for an infinite alphabet and the letters \mathbb{E} and \mathbb{F} for any kind of alphabet. By \mathbb{E}^* we denote the set of finite sequences over \mathbb{E} , by \mathbb{E}^+ the set of finite sequences with at least one element over \mathbb{E} , and by \mathbb{E}^ω the set of infinite sequences over \mathbb{E} . We write ϵ for the empty sequence and ‘ \cdot ’ as the concatenation operator between sequences. We write $|S|$ to denote the length of S (if S is a finite sequence), or its cardinality (if S is a set).

Regular languages. We denote by $\text{REG}(\mathbb{A})$ the set of regular expressions over the finite alphabet \mathbb{A} . We make use of the many characterizations of regular languages over a finite alphabet \mathbb{A} . In particular, we use that a word language $\mathcal{L} \subseteq \mathbb{A}^*$ is regular iff it satisfies one of the following equivalent properties:

- there is a deterministic (or non-deterministic) finite automaton recognizing \mathcal{L} ,
- it is described by a regular expression,
- there is a finite monoid (M, \cdot) with a distinguished subset $T \subseteq M$, and a monoid homomorphism $h : \mathbb{A}^* \rightarrow M$ such that $w \in \mathcal{L}$ iff $h(w) \in T$,
- there is a finite semigroup (S, \cdot) with a distinguished subset $T \subseteq S$, and a semigroup homomorphism $h : \mathbb{A}^* \rightarrow S$ such that for all w with $|w| > 0$, $w \in \mathcal{L}$ iff $h(w) \in T$.

Depending on the section, in order to clarify the presentation, we will use the characterization that fits the best our needs.

Unranked finite trees. By $\text{Trees}(\mathbb{E})$ we denote the set of finite ordered and unranked trees over an alphabet \mathbb{E} . We view each *position* in a tree as an element of $(\mathbb{N}_+)^*$. Formally, we define $\text{POS} \subseteq \wp_{<\infty}((\mathbb{N}_+)^*)$ as the set of sets of finite tree positions, such that: $X \in \text{POS}$

iff (a) $X \subseteq (\mathbb{N}_+)^*$, $|X| < \infty$; (b) X is prefix-closed; and (c) if $n \cdot (i + 1) \in X$ for $i \in \mathbb{N}_+$, then $n \cdot i \in X$ for $n \in \mathbb{N}_+$. A tree is then a mapping from a set of positions to letters of the alphabet $Trees(\mathbb{E}) := \{\mathbf{t} : P \rightarrow \mathbb{E} \mid P \in \text{POS}\}$. By $\mathbf{t}|_x$ we denote the subtree of \mathbf{t} at position x : $\mathbf{t}|_x(y) = \mathbf{t}(x \cdot y)$. The root's position is the empty string and we denote it by ' ϵ '. The position of any other node in the tree is the concatenation of the position of its parent and the node's index in the ordered list of siblings. In this work we use v, w, x, y, z as variables for positions, and i, j, k, l, m, n as variables for numbers. Thus two positions x, y are sibling if they are of the form $x = z \cdot i$ and $y = z \cdot j$ for some z, j ; whereas x is the parent of y (resp. y is the child of x) if y is of the form $x \cdot i$ for some i . Note that from the notation $x \cdot i$ one knows that it is a position which is not the root, that has x as parent position, and that has $i - 1$ siblings to the left.

Given a tree $\mathbf{t} \in Trees(\mathbb{E})$, $\text{pos}(\mathbf{t})$ denotes the domain of \mathbf{t} , which consists of the set of positions of the tree, and $\text{alph}(\mathbf{t}) = \mathbb{E}$ denotes the alphabet of the tree. From now on, we informally refer by 'node' to a position x together with the value $\mathbf{t}(x)$.

Given two trees $\mathbf{t}_1 \in Trees(\mathbb{E})$, $\mathbf{t}_2 \in Trees(\mathbb{F})$ such that $\text{pos}(\mathbf{t}_1) = \text{pos}(\mathbf{t}_2) = P$, we define $\mathbf{t}_1 \otimes \mathbf{t}_2 : P \rightarrow (\mathbb{E} \times \mathbb{F})$ as $(\mathbf{t}_1 \otimes \mathbf{t}_2)(x) = (\mathbf{t}_1(x), \mathbf{t}_2(x))$.

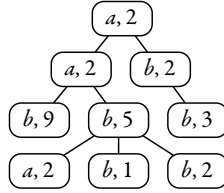


Figure 1: A data tree.

The set of *data trees* over a finite alphabet \mathbb{A} and an infinite domain \mathbb{D} is defined as $Trees(\mathbb{A} \times \mathbb{D})$. Note that every tree $\mathbf{t} \in Trees(\mathbb{A} \times \mathbb{D})$ can be decomposed into two trees $\mathbf{a} \in Trees(\mathbb{A})$ and $\mathbf{d} \in Trees(\mathbb{D})$ such that $\mathbf{t} = \mathbf{a} \otimes \mathbf{d}$. Figure 1 shows an example of a data tree. The notation for the set of data values used in a data tree is $data(\mathbf{a} \otimes \mathbf{d}) := \{\mathbf{d}(x) \mid x \in \text{pos}(\mathbf{d})\}$. With an abuse of notation we write $data(X)$ to denote all the elements of \mathbb{D} contained in X , for whatever object X may be.

Downward path. A *downward path* starting at a node x of a tree \mathbf{t} is the sequence of labels of a simple path whose initial node is x and going to a descendant of x . In other words, it is the word of the form $a_1 \cdots a_n$ where, for all $1 \leq i \leq n$, $a_i = \mathbf{t}(x_i)$ with $x_1 = x$ and x_{i+1} is a child of x_i .

XPath on data trees. Finally we define vertical XPath, the fragment of XPath where no horizontal navigation is allowed. We actually consider an extension of XPath allowing the Kleene star on *any* path expression and we denote it by regXPath . Although here we define XPath (a language conceived for XML documents) over *data trees* instead of over *XML documents*, the main decidability result can be easily transferred to XPath over XML documents through a standard translation (see for instance [BMSS09]).

Vertical regXPath is a two-sorted language, with *path* expressions (denoted by α, β, γ) and *node* expressions (denoted by φ, ψ, η). Path expressions are binary relations resulting from composing the child and parent relations (which are denoted respectively by \downarrow and \uparrow), and node expressions. Node expressions are boolean formulas that test a property of a node, like for example, that it has a certain label, or that it has a child labeled a with

$$\begin{aligned} \alpha, \beta &::= o \mid \alpha[\varphi] \mid [\varphi]\alpha \mid \alpha\beta \mid \alpha \cup \beta \mid \alpha^* & o \in \{\varepsilon, \downarrow, \uparrow\}, \\ \varphi, \psi &::= a \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle \alpha \rangle \mid \langle \alpha = \beta \rangle \mid \langle \alpha \neq \beta \rangle & a \in \mathbb{A}. \end{aligned}$$

The syntax of vertical XPath

$$\begin{aligned} \llbracket \downarrow \rrbracket^{\mathbf{t}} &= \{(x, x \cdot i) \mid x \cdot i \in \text{pos}(\mathbf{t})\} & \llbracket \uparrow \rrbracket^{\mathbf{t}} &= \{(x \cdot i, x) \mid x \cdot i \in \text{pos}(\mathbf{t})\} \\ \llbracket [\varphi] \rrbracket^{\mathbf{t}} &= \{(x, x) \mid x \in \text{pos}(\mathbf{t}), x \in \llbracket \varphi \rrbracket^{\mathbf{t}}\} & \llbracket \alpha^* \rrbracket^{\mathbf{t}} &= \text{the reflexive transitive closure of } \llbracket \alpha \rrbracket^{\mathbf{t}} \\ \llbracket \varepsilon \rrbracket^{\mathbf{t}} &= \{(x, x) \mid x \in \text{pos}(\mathbf{t})\} & \llbracket \alpha\beta \rrbracket^{\mathbf{t}} &= \{(x, z) \mid \exists y. (x, y) \in \llbracket \alpha \rrbracket^{\mathbf{t}}, \\ & & & (y, z) \in \llbracket \beta \rrbracket^{\mathbf{t}}\} \\ \llbracket \alpha \cup \beta \rrbracket^{\mathbf{t}} &= \llbracket \alpha \rrbracket^{\mathbf{t}} \cup \llbracket \beta \rrbracket^{\mathbf{t}} & \llbracket \langle \alpha \rangle \rrbracket^{\mathbf{t}} &= \{x \in \text{pos}(\mathbf{t}) \mid \exists y. (x, y) \in \llbracket \alpha \rrbracket^{\mathbf{t}}\} \\ \llbracket a \rrbracket^{\mathbf{t}} &= \{x \in \text{pos}(\mathbf{t}) \mid \mathbf{a}(x) = a\} & \llbracket \langle \alpha = \beta \rangle \rrbracket^{\mathbf{t}} &= \{x \in \text{pos}(\mathbf{t}) \mid \exists y, z. (x, y) \in \llbracket \alpha \rrbracket^{\mathbf{t}}, \\ & & & (x, z) \in \llbracket \beta \rrbracket^{\mathbf{t}}, \mathbf{d}(y) = \mathbf{d}(z)\} \\ \llbracket \neg\varphi \rrbracket^{\mathbf{t}} &= \text{pos}(\mathbf{t}) \setminus \llbracket \varphi \rrbracket^{\mathbf{t}} & \llbracket \langle \alpha \neq \beta \rangle \rrbracket^{\mathbf{t}} &= \{x \in \text{pos}(\mathbf{t}) \mid \exists y, z. (x, y) \in \llbracket \alpha \rrbracket^{\mathbf{t}}, \\ & & & (x, z) \in \llbracket \beta \rrbracket^{\mathbf{t}}, \mathbf{d}(y) \neq \mathbf{d}(z)\} \end{aligned}$$

The semantics of vertical XPath over a data tree $\mathbf{t} = \mathbf{a} \otimes \mathbf{d}$

Figure 2: The syntax and semantics of vertical XPath.

the same data value as an ancestor labeled b , which is expressed by $\langle \downarrow[a] = \uparrow^*[b] \rangle$. We write $\text{regXPath}(\mathfrak{A}, =)$ to denote this logic. A *formula* of $\text{regXPath}(\mathfrak{A}, =)$ is either a node expression or a path expression of the logic. Its syntax and semantics are defined in Figure 2.

As another example, we can select the nodes that have a descendant labeled b with two children also labeled by b with different data values by a formula $\varphi = \langle \downarrow^*[b \wedge \langle \downarrow[b] \neq \downarrow[b] \rangle] \rangle$. Given a tree \mathbf{t} as in Figure 1, we have $\llbracket \varphi \rrbracket^{\mathbf{t}} = \{\varepsilon, 1, 12\}$.

The satisfiability problem for $\text{regXPath}(\mathfrak{A}, =)$ is the problem of, given a formula φ , whether there exists a data tree \mathbf{t} such that $\llbracket \varphi \rrbracket^{\mathbf{t}} \neq \emptyset$.

Our main result on XPath is the following.

Theorem 2.1. *The satisfiability problem for $\text{regXPath}(\mathfrak{A}, =)$ is decidable.*

The proof of Theorem 2.1 goes as follows. We define a model of automata running over data trees. This model of automata is interesting on its own and the second main result of this paper shows that they have a decidable emptiness problem. Finally we show that formulas of $\text{regXPath}(\mathfrak{A}, =)$ can be translated into a BUDA.

3. THE AUTOMATA MODEL

In this section we introduce our automata model. It is essentially a bottom-up tree automaton with one register to store a data value and an alternating control. We will see in Section 5 that these automata are expressive enough to capture vertical regXPath . In Section 4 we will show that their emptiness problem is decidable. Theorem 2.1 then follows immediately.

A Bottom-Up Data Automaton (BUDA) \mathcal{A} runs over data trees of $\text{Trees}(\mathbb{A} \times \mathbb{D})$ and it is defined as a tuple $\mathcal{A} = (\mathbb{A}, \mathbb{B}, Q, q_0, \delta)$ where \mathbb{A} is the finite alphabet of the tree, \mathbb{B} is an internal finite alphabet of the automaton (whose purpose will be clear later), Q is a finite

set of states, q_0 is the initial state, and δ is the transition function which is a finite set of pairs of the form $(test, action)$ that will be described below.

Before we present the precise syntax and semantics of our automaton model, we first give the intuition. The automaton has one register, where it can store and read a data value from \mathbb{D} , and it has alternating control. Hence, at any moment several independent *threads* of the automaton may be running in parallel. Each thread has one register and consists of a state from Q and a data value from \mathbb{D} stored in the register. The automaton first guesses a finite internal label from \mathbb{B} for every node of the tree and all threads share access to this finite information. This internal information can be viewed as a synchronization feature between threads and will be necessary later for capturing the expressive power of vertical `regXPath`. The automaton is bottom-up, and it starts with one thread with state q_0 at every leaf of the tree with an arbitrary data value in its register. From there, each thread evolves independently according to the transition function δ : If the test part of a pair in δ is true then the thread can perform the corresponding action, which may trigger the creation of new threads. We first describe the set of possible tests the automata may perform and then the set of their possible actions.

The tests may consist of any conjunction of the basic tests described below or their negation. The automata can test the current state, the label (from \mathbb{A}) and internal label (from \mathbb{B}) of the current node and also whether the current node is the root, a leaf or an internal node. The automata can test equality of the current data value with the one stored in the register (denoted by `eq`). Finally the automata can test the existence of some downward path, starting from the current node and leading to a node whose data value is (or is not) equal to the one currently stored in the register, such that the path matches some regular expression on the labels. For example, for a regular expression exp over the alphabet $\mathbb{A} \times \mathbb{B}$, the test $\langle exp \rangle^=$ checks the existence of a downward path that matches exp , which starts at the current node and leads to a node whose data value matches the one currently stored in the register. Similarly, $\langle exp \rangle^\neq$ tests that it leads to a data value different from the one currently in the register.

The precise set of possible basic tests is:

$$\text{BTESTS} = \{p, \text{eq}, \langle exp \rangle^=, \langle exp \rangle^\neq, \text{root}, \text{leaf}, a, b \mid exp \in \text{REG}(\mathbb{A} \times \mathbb{B}), p \in Q, a \in \mathbb{A}, b \in \mathbb{B}\}.$$

If x is a basic test, we will write \bar{x} to denote the test corresponding to the negation of x . For instance $\overline{\text{eq}}$ tests whether the current data value differs from the one stored in the register. The possible set of tests is then:

$$\text{TESTS} = \text{BTESTS} \cup \overline{\text{BTESTS}}.$$

Based on the result of a test the thread can perform an action. A basic action either accepts (`accept`) and the corresponding thread terminates, or specifies a new state p and a new content for the register for each thread it generates, each of them moving up in the tree to the parent node. The possible updates of the register are: keep the register's data value unchanged (denoted by `keep`), store the current data value in the register (denoted by `store`), store an arbitrary data value non-deterministically chosen (`guess`), or start a new thread for every data value of the subtree (`univ`) of the current node. Note that this last action creates unboundedly many new threads. Altogether the precise set of possible basic actions is:

$$\text{ACTIONS} = \{\text{accept}, \text{keep}(p), \text{store}(p), \text{guess}(p), \text{univ}(p) \mid p \in Q\}$$

and the set of actions is any conjunction of those. As usual, conjunction corresponds to universality. For example with an action of the form $a_1 \wedge a_2$ the automaton starts two new

threads, one specified by a_1 and one specified by a_2 . If a_2 would be $\text{univ}(p)$ then it actually starts one new thread in state p and data value d per data value d occurring in the subtree of the current node.

A transition is therefore a pair $(\text{test}, \text{action})$ where test is a conjunction of basic tests in TESTS and action is a conjunction of basic actions in ACTIONS. There might be several rules involving the same tests, corresponding to non-determinism.

Before we move on to the formal definition of the language accepted by a BUDA, we stress that the automaton model is not closed under complementation because its set of actions are not closed under complementation: guess is a form of existential quantification while univ is a form of universal quantification, but they are not dual. Actually, we will show in Proposition 3.2 that adding their dual would yield undecidability of the model.

We now turn to the formal definition of the semantics of a BUDA. A data tree $\mathbf{a} \otimes \mathbf{d} \in \text{Trees}(\mathbb{A} \times \mathbb{D})$ is accepted by \mathcal{A} iff there exists an internal labeling $\mathbf{b} \in \text{Trees}(\mathbb{B})$ with $\text{pos}(\mathbf{b}) = \text{pos}(\mathbf{a} \otimes \mathbf{d})$ such that there is an accepting run of \mathcal{A} on $\mathbf{t} = \mathbf{a} \otimes \mathbf{b} \otimes \mathbf{d}$. We focus now on the definition of a run.

We say that a thread (q, d) makes a basic test $t \in \text{BTESTS}$ true at a position x of \mathbf{t} , and write $\mathbf{t}, x, (q, d) \models t$, if:

- t is one of the tests $p, a, b, \text{root}, \text{leaf}, \text{eq}$ and we have respectively $q = p, \mathbf{a}(x) = a, \mathbf{b}(x) = b, x$ is the root of \mathbf{t}, x is a leaf of $\mathbf{t}, \mathbf{d}(x) = d$,
- t is $\langle \text{exp} \rangle^=$ and there is a downward path in \mathbf{t} matching exp , starting at x and ending at y where $\mathbf{d}(y) = d$. The case of $\langle \text{exp} \rangle^{\neq}$ is treated similarly replacing $\mathbf{d}(y) = d$ by $\mathbf{d}(y) \neq d$.

This definition and notation lifts to arbitrary Boolean combination of basic tests in the obvious way. Note that $\overline{\langle \text{exp} \rangle^=}$ is not $\langle \text{exp} \rangle^{\neq}$. The former is true if there no downward path matching exp and reaching the current data value, while the latter requires the existence of a downward path matching exp and reaching a data value different from the current one.

A *configuration* of a BUDA \mathcal{A} is a set \mathcal{C} of threads, viewed as a finite subset of $Q \times \mathbb{D}$. A configuration \mathcal{C} is said to be *initial* iff it is the singleton $\{(q_0, e)\}$ for some $e \in \mathbb{D}$.

A *run* ρ of \mathcal{A} on $\mathbf{t} = \mathbf{a} \otimes \mathbf{b} \otimes \mathbf{d}$ is a function associating a configuration to any node x of \mathbf{t} such that

- for any leaf x of \mathbf{t} , $\rho(x)$ is initial,
- for any inner position x of \mathbf{t} , whose parent is the position y , and for any $(q, d) \in \rho(x)$ there exists $(t, \text{Ac}) \in \delta$ with $\text{Ac} = \bigwedge_{j \in J} \text{Ac}_j$ such that $\mathbf{t}, x, (q, d) \models t$ and for any $j \in J$ we have:
 - if Ac_j is $\text{keep}(p)$ then $(p, d) \in \rho(y)$,
 - if Ac_j is $\text{store}(p)$ then $(p, \mathbf{d}(x)) \in \rho(y)$,
 - if Ac_j is $\text{guess}(p)$ then $(p, e) \in \rho(y)$ for some $e \in \mathbb{D}$,
 - if Ac_j is $\text{univ}(p)$ then for all $e \in \text{data}(\mathbf{t}|_x)$, $(p, e) \in \rho(y)$.

The run ρ on \mathbf{t} is *accepting* if moreover for the root y of \mathbf{t} and all $(q, d) \in \rho(y)$ there exists $(t, \text{accept}) \in \delta$ such that $\mathbf{t}, y, (q, d) \models t$.

3.1. Discussion.

3.1.1. *Semigroup notation.* For convenience of notation in the proofs, we shall use an equivalent definition of BUDA using *semigroup homomorphisms* instead of regular expressions. That is, we consider an automaton $\mathcal{A} \in \text{BUDA}$ as a tuple $\mathcal{A} = (\mathbb{A}, \mathbb{B}, Q, q_0, \delta, \mathcal{S}, h)$ where \mathcal{S} is a finite semigroup, h is a semigroup homomorphism from $(\mathbb{A} \times \mathbb{B})^+$ to \mathcal{S} , and tests of the

form $\langle \mu \rangle^=$ and $\langle \mu \rangle^\neq$ contain a semigroup element $\mu \in \mathcal{S}$. Hence, $\langle \mu \rangle^=$ is true at x in \mathbf{t} if there is a downward path in \mathbf{t} starting at x and ending at a descendant y , evaluating to μ via h and such that $\mathbf{d}(y) = d$. The case of $\langle \mu \rangle^\neq$ is treated similarly replacing $\mathbf{d}(y) = d$ by $\mathbf{d}(y) \neq d$. Note that since regular languages are exactly those recognized by finite semigroups (recall Section 2) this is an equivalent automata model.

3.1.2. Disjunction. As mentioned earlier the automata model does not allow for disjunctions of actions or tests. But in fact these can be added without changing the expressivity of the automaton, by modifying the transition relation δ :

- any automaton having transition with a disjunction of actions $(t, a_1 \vee a_2)$ is equivalent to the automaton resulting from replacing $(t, a_1 \vee a_2)$ with the transitions (t, a_1) and (t, a_2) ,
- any automaton having transition with a disjunction of tests $(t_1 \vee t_2, a)$ is equivalent to the automaton resulting from replacing $(t_1 \vee t_2, a)$ with the transitions (t_1, a) and (t_2, a) .

For this reason we will sometimes write disjunction of actions or of tests, as a shorthand for the equivalent automaton without disjunctions.

We will also make use of a test $\langle \mu \rangle$ denoting $\langle \mu \rangle^= \vee \langle \mu \rangle^\neq$; and $\overline{\langle \mu \rangle}$ denoting $\overline{\langle \mu \rangle^=} \wedge \overline{\langle \mu \rangle^\neq}$.

3.1.3. Closure properties. We say that a model is closed under effective operation O , if it is closed under O and further the result of the application of the operation O is computable.

Proposition 3.1. *The class BUDA is closed under effective intersection and effective union.*

Proof. This is straightforward from the fact that the model has alternation and non-determinism. \square

As already mentioned, the closure under complementation of BUDA yields an undecidable model.

Proposition 3.2. *Any extension of BUDA closed under effective complementation and effective intersection has an undecidable emptiness problem.*

Proof. The automaton model $\text{ARA}(\text{guess})$ introduced in [Fig12] is an alternating 1-register automaton over *data words*, which are essentially data trees whose every position has at most one child. The model is equivalent to a restricted version of BUDA without the tests $\langle \text{exp} \rangle^=$, $\langle \text{exp} \rangle^\neq$ and without the action univ running on data words. Indeed, the $\text{ARA}(\text{guess})$ model is an alternating automaton with one register and a guess operator just like the one of BUDA. Then, there is a simple reduction from the emptiness problem for the automata model $\text{ARA}(\text{guess})$ into the emptiness problem of BUDA. Indeed, there is a straightforward translation f from $\text{ARA}(\text{guess})$ to BUDA so that for every $\mathcal{A} \in \text{ARA}(\text{guess})$

- if a data word is accepted by \mathcal{A} , then it is accepted by $f(\mathcal{A})$, and
- if a data tree is accepted by $f(\mathcal{A})$, then any of its maximal branches (seen as data words, starting at a leaf and ending at the root) is accepted by \mathcal{A} .

By means of contradiction, suppose that there is a class of automata C that extends the class BUDA and is closed under effective complementation and effective intersection. We show undecidability by reduction from the emptiness problem for Minsky machines [Min67]. In [Fig12, Proof of Proposition 3.2], it is shown that for every Minsky machine M one can build two properties P and P' expressible by $\text{ARA}(\text{guess})$ so that $P \wedge \neg P'$ is satisfiable iff M is non-empty. By the reduction above, the properties $\dot{P} = \text{every branch satisfies } P$ and

$\hat{P}' = \text{every branch satisfies } P'$ are expressible with C . Since C is closed under intersection and complementation, then $\hat{P} \wedge \neg \hat{P}'$ is expressible with C . Note that a data tree satisfies $\hat{P} \wedge \neg \hat{P}'$ iff it has a branch satisfying $P \wedge \neg P'$, which happens iff M is non-empty. Hence, C has an undecidable emptiness problem. \square

Since we will show that the emptiness problem for BUDA is decidable and BUDA is closed under intersection, we then have that they are not closed under complementation.

Corollary 3.3. *The class BUDA is not closed under effective complementation.*

We do not know how to show that BUDA is not closed under complement. A possible concrete example would be the property that on every branch of the data tree the data values under a node of label a are the same as those under the nodes of label b . The complement of this property is expressible by a BUDA, but it seems that a BUDA cannot express it.

3.2. Automata normal form. We now present a normal form of BUDAs, removing redundancy in its definition. This normal form simplifies the technical details in the proof of decidability presented in the next section. We use the semigroup point of view.

(NF1) The semigroup \mathcal{S} and homomorphism h have the property that different values are used for paths of length one: For all $w \in (\mathbb{A} \times \mathbb{B})^+$ and $c \in \mathbb{A} \times \mathbb{B}$, $h(w) = h(c)$ iff $w = c$.

(NF2) All transitions (*test*, *action*) are such that *test* contains only conjuncts of the form p , $\langle \text{exp} \rangle^=$, $\langle \text{exp} \rangle^{\neq}$, *root* as well as their negated $\overline{}$ counterparts.

An automaton $\mathcal{A} \in \text{BUDA}$ is said to be in *normal form* if it satisfies (NF1), (NF2).

Proposition 3.4. *For any $\mathcal{A} \in \text{BUDA}$, there is an equivalent $\mathcal{A}' \in \text{BUDA}$ in normal form that can be effectively obtained.*

Proof. First, given a finite semigroup we can easily compute another one that satisfies (NF1), only by adding some extra elements to the domain in order to tell apart all the one letter words for each symbol of the finite alphabet.

To show that (NF2) can always be assumed, note that any test for label can be simulated using $\langle \mu \rangle$. Indeed, a test a with $a \in \mathbb{A}$ can be simulated with $\bigvee_{b \in \mathbb{B}} \langle h(a, b) \rangle$, \bar{a} with $\bigwedge_{b \in \mathbb{B}} \overline{\langle h(a, b) \rangle}$, and similar tests can simulate b and \bar{b} for $b \in \mathbb{B}$. Once this is done, a test $\langle \mu \rangle$ can be simulated using $\langle \mu \rangle^= \vee \langle \mu \rangle^{\neq}$. The test *eq* can be simulated using $\bigvee_{a \in \mathbb{A} \times \mathbb{B}} \langle h(a) \rangle^=$. Similarly, $\overline{\text{eq}}$ can be simulated using $\langle \mu \rangle^{\neq}$. Lastly, *leaf* and $\overline{\text{leaf}}$ can be tested with $\bigwedge_{\mu \notin \{h(a) \mid a \in \mathbb{A} \times \mathbb{B}\}} \overline{\langle \mu \rangle^=}$ and $\bigvee_{\mu \notin \{h(a) \mid a \in \mathbb{A} \times \mathbb{B}\}} \langle \mu \rangle^= \vee \langle \mu \rangle^{\neq}$ respectively. Thus, we can suppose that the automaton \mathcal{A} does not contain any transition that uses tests for labels, *eq*, $\overline{\text{eq}}$, *leaf* and $\overline{\text{leaf}}$ without any loss of generality. \square

3.3. Non-moving actions. Here we introduce an extension to the automata model with non-moving ε -transitions. The automaton can now perform actions while remaining at the same node. We show that this extension does not change the expressive power of the model. This extension will prove useful when translating vertical *regXPath* into BUDA in Section 5.

A BUDA automaton with ε -transitions, noted BUDA^ε , is defined as any BUDA automaton, with the only difference that its state of states Q is split into two disjoint sets Q_ε and Q_{move} . Moreover the set Q is partially ordered by $<$. Whenever an action $\text{act}(p)$ is taken where $p \in Q_\varepsilon$, then the automaton switches to state p , performs the action act , but does not move

to the parent node and stays instead at the same node. In order to avoid infinite sequences of ε -transitions we require that when switching to a state q from a state p where $q \in Q_\varepsilon$ then $p < q$. Hence a thread can make at most $|Q_\varepsilon|$ ε -steps before moving up in the tree.

Formally a BUDA^ε is a tuple $(\mathbb{A}, \mathbb{B}, Q_\varepsilon, Q_{\text{move}}, <, q_0, \delta, \mathcal{S}, h)$ where $Q = Q_\varepsilon \cup Q_{\text{move}}$ is a set partially ordered by $<$ and the set of transition δ contains only pairs (t, a) such that if a contains a basic action $\text{act}(p)$ where $p \in Q_\varepsilon$ then t contains a basic test of the form $q \in Q$ with $q < p$. The rest is defined as for BUDA .

A run ρ of a BUDA^ε $\mathcal{A} = (\mathbb{A}, \mathbb{B}, Q_\varepsilon, Q_{\text{move}}, <, q_0, \delta, \mathcal{S}, h)$ on $\mathbf{t} = \mathbf{a} \otimes \mathbf{b} \otimes \mathbf{d}$ is defined as for BUDA :

- for any leaf x of \mathbf{t} , $\rho(x)$ is initial,
- for any inner position x of \mathbf{t} , whose parent is the position y , and any $(p, d) \in \rho(x)$ there exists $(t, a) \in \delta$ with $a = \bigwedge_{j \in J} a_j$ such that $\mathbf{t}, x, (p, d) \models t$ and for any $j \in J$ we have:
 - if a_j is $\text{keep}(q)$ with $q \in Q_{\text{move}}$, then $(q, d) \in \rho(y)$,
 - if a_j is $\text{keep}(q)$ with $q \in Q_\varepsilon$, then $(q, d) \in \rho(x)$,
 - if a_j is $\text{store}(q)$ with $q \in Q_{\text{move}}$, then $(q, \mathbf{d}(x)) \in \rho(y)$,
 - if a_j is $\text{store}(q)$ with $q \in Q_\varepsilon$, then $(q, \mathbf{d}(x)) \in \rho(x)$,
 - if a_j is $\text{guess}(q)$ with $q \in Q_{\text{move}}$, then $(q, e) \in \rho(y)$ for some $e \in \mathbb{D}$,
 - if a_j is $\text{guess}(q)$ with $q \in Q_\varepsilon$, then $(q, e) \in \rho(x)$ for some $e \in \mathbb{D}$,
 - if a_j is $\text{univ}(q)$ with $q \in Q_{\text{move}}$, then for all $e \in \text{data}(\mathbf{t}|_x)$, $(q, e) \in \rho(y)$,
 - if a_j is $\text{univ}(q)$ with $q \in Q_\varepsilon$, then for all $e \in \text{data}(\mathbf{t}|_x)$, $(q, e) \in \rho(x)$.

As expected we show that adding ε -transitions does not increase the expressive power.

Proposition 3.5. *There is an effective language-preserving translation from BUDA^ε into BUDA .*

Proof. Let \mathcal{A} be a BUDA^ε . We say that a basic action is an ε -action if it is of the form $\text{act}(q)$ for some $q \in Q_\varepsilon$. The *rank* of an ε -action $\text{act}(q)$ is the number of $p \in Q$ such that $p < q$.

We first assume without loss of generality that \mathcal{A} contains no ε -action of the form $\text{store}(q)$, since those can be simulated by a guess ε -action followed by a eq test.

We also assume without loss of generality that all transitions (t, a) of \mathcal{A} are such that t contains exactly one conjunct of the form p and no conjunct of the form \bar{q} , for $p, q \in Q$. This can be enforced using non-determinism by always testing all the finitely many possibilities for the states. All tests t now contain exactly one test for a state q and we call q *the state associated to t* . We denote by $t \setminus q$ the new test constructed from t by removing the conjunct q .

We now show how to remove the ε -actions of the form $\text{keep}(q)$. We remove them one by one computing at each step an equivalent automaton \mathcal{A}' with the same set of states and the same partial order on it. \mathcal{A}' is essentially \mathcal{A} with one transition removed (the one containing $\text{keep}(q)$) and several transitions added. The idea is classical and consists in performing the actions executed by transitions associated to q instead of executing $\text{keep}(q)$. The added transitions may contain new ε -actions of the form $\text{keep}(p)$ but those satisfy $q < p$. Hence this process will eventually terminate. Assume that \mathcal{A} has a transition (t, a) containing an ε -action $\text{keep}(q)$ where q has a minimal possible rank satisfying this property. We let \hat{a} denote the action computed from a by removing the conjunct $\text{keep}(q)$. \mathcal{A}' is copied from \mathcal{A} with the following modifications:

We remove (t, a) from the list of transitions and for each transition (t', a') of \mathcal{A} such that q is associated to t' we add the new transition $(t \wedge (t' \setminus q), \hat{a} \wedge a')$. Notice that all the ε -actions of the form $\text{keep}(p)$ occurring in these new transitions come from a' or \hat{a} and

therefore are such that $q < p$ as desired. Notice that since the action `keep` do not change the data value stored in the register, the test $t' \setminus q$ can be equivalently performed before or after executing this action. Hence the new automaton is equivalent to the old one.

We can now assume that \mathcal{A} contains only ε -actions of the form `guess`(q) and `univ`(q).

We next show how to remove the ε -actions of the form `guess`(q). We again remove them one by one computing an equivalent automaton \mathcal{A}' . This automaton has new states but those do not modify the rank of the ε -actions of the form `guess`(p). As before it is essentially \mathcal{A} with one transition removed and several new transitions. The idea is to perform the guessing of the data value at one of the children of the current node, while moving up in the tree. The new transitions needed to do this may introduce new ε -actions of the form `guess`(p), the rank of p being strictly smaller than the rank of q . Hence this process will eventually terminate. Assume that \mathcal{A} has a transition (t, a) containing an ε -action `guess`(q) where q has a maximal possible rank satisfying this property. Let p be the state associated to t (hence $p < q$) and \hat{a} be the action computed from a by removing the conjunct `guess`(q). \mathcal{A}' is copied from \mathcal{A} with the following modifications:

We add two new states q' and q'' . If $p \in Q_{move}$ then so are q' and q'' . If $p \in Q_\varepsilon$ then so are q' and q'' . We modify $<$ by setting $r < q'$ and $r < q''$ whenever $r < p$ and $q' < r$ and $q'' < r$ whenever $p < r$. In other words q' and q'' play the same role as p in the partial order but are incomparable with p . Hence the rank of any state of Q is not modified by the addition of these two states. Moreover the ranks of q' and q'' match the rank of p and are therefore strictly less than the rank of q .

We now update the list of transitions by first removing (t, a) . Then, consider a transition (t', a') of \mathcal{A} of state p' and such that a' contains an action of the form `act`(p). We add a new transition (t', a'') where a'' is the action constructed from a' by removing `act`(p) and adding `act`(q') and `guess`(q'') (this is consistent with the partial order). Finally we add in \mathcal{A}' a transition $(q' \wedge (t \setminus p), \hat{a})$ and a transition $(q'' \wedge (t' \setminus q), b)$ for any transition (t'', b) such that p is the state associated to t'' , both being consistent with the partial order.

The reader can now verify that \mathcal{A}' is equivalent to \mathcal{A} . If right after switching to state p using the transition (t', a') , where `act`(p) is a basic action of a' , \mathcal{A} decides to use the transition (t, a) then \mathcal{A}' can simulate this as follows. First it uses the transition (t', a'') . This generates two new basic actions, `act`(q') and `guess`(q'') instead of `act`(p). The thread generated by `act`(q') will perform the tests and actions that \mathcal{A} did by using the transition (t, a) , except for `guess`(q). But this latter action is simulated by `guess`(q'') that has been launched earlier. Hence \mathcal{A}' does simulate \mathcal{A} also in this case. The other direction, showing that a run of \mathcal{A}' can be simulated by a run of \mathcal{A} is proved similarly.

We can now assume that \mathcal{A} contains only ε -actions of the form `univ`(q). Removing those requires more care. The idea is, as above, to perform the action at the children of the current node while moving up in the tree. As this is a universal move, all children are concerned and we therefore need some synchronization. The internal alphabet will be used for this. In a nutshell the new automaton will mark the nodes where the `univ`(q) should be executed and mark all their children. Simple threads are launched at the leaves making sure the marking is consistent. When encountering a node marked as a child, `univ`(q) can be executed while moving up in the tree, taking care of all the data values of the corresponding subtree. An extra thread need to be executed at the parent node to take care of its data value. Note that different tests can be launched in different threads by the use of actions like `keep`(q) \wedge `keep`(p) with $p, q \in Q_\varepsilon$. We now turn to the details.

Assume that \mathcal{A} has a transition (t, a) containing an ε -action $\text{univ}(q)$ where q has a maximal possible rank satisfying this property. Let p be the state associated to t and \hat{a} be the action computed from a by removing the conjunct $\text{univ}(q)$. \mathcal{A}' is essentially a copy of \mathcal{A} with the following modifications:

We use $\mathbb{B}' = \mathbb{B} \times \{0, 1, 2\}$ as internal alphabet for \mathcal{A}' . The nodes containing 1 in their label are expected to simulate an action $\text{univ}(q)$, and their children must contain 2 in their label. On top of simulating \mathcal{A} , \mathcal{A}' starts a new thread at all the leaves of the tree, using a fresh new state in Q'_{move} . These threads make sure that all nodes whose label contain 2 have a parent with 1 in their label and that all nodes having 1 in their label have no child without 2 in their label (this can be done with the appropriate $\langle \text{exp} \rangle$ -test). Moreover, we assume yet a new state q' in Q'_{move} and these new threads trigger an action $\text{univ}(q')$ each time the symbol 2 is found. In other words, we make sure that any node x with internal label 1 has a thread (q', d) for every data value occurring strictly below x . We replace the transition (t, a) with $(t^1, \text{store}(q) \wedge \hat{a})$ where t^1 performs the same tests as t but also checks that the internal label contains 1. Finally, to any transition (t', a') of \mathcal{A} whose associated test is q we add a new transition replacing q with q' (note that we don't remove any transition, hence (t', a') is still a transition of \mathcal{A}').

Note that this generates a new ε -action of the form $\text{store}(q)$. However we have seen that those could be simulated using **guess** ε -actions and we have seen above that such ε -actions could be removed. As this last step does not introduce any new univ ε -actions, the resulting automaton has strictly less ε -actions of that rank.

For correctness, assume that \mathcal{A} accepts a data tree \mathbf{t} with an accepting run ρ . \mathcal{A}' marks all positions where ρ uses transition (t, a) , with 1, and the children of those nodes with 2. The rest of the simulation of \mathcal{A} is trivial. The run will be accepting because all the new generated threads will have exactly the same effect as the action $\text{univ}(q)$ at the marked nodes: the action $\text{store}(q)$ generates a thread with the current data value, while the actions $\text{univ}(q')$ generate a thread for all the data values within the subtrees. Conversely, assume that \mathcal{A}' has an accepting run ρ' on a data tree \mathbf{t} . Let x be a node whose internal label has 1. If ρ' does use the transition $(t^1, \text{store}(q) \wedge \hat{a})$ at node x then \mathcal{A} can simulate \mathcal{A}' using the transition (t, a) as they produce the same threads. If ρ' does not use the transition $(t^1, \text{store}(q) \wedge \hat{a})$ at that node, then the initial internal labeling was incorrectly guessed. But this is not important as \mathcal{A} can still simulate \mathcal{A}' on a subset of the threads, hence will also accept \mathbf{t} . In other words, \mathcal{A}' will have executed an action $\text{univ}(p)$ at x that was not necessary for accepting the tree. \square

4. THE EMPTINESS PROBLEM FOR BUDA

This is the most technical section of the paper. Its goal is to show:

Theorem 4.1. *The emptiness problem for BUDA is decidable.*

This result is shown through methods from the theory of *well-structured transition systems*, or WSTS for short [FS01]. It is obtained by interpreting the execution of a BUDA using a transition system compatible with some well-quasi-ordering (wqo). We start with the notions of the theory of WSTS that we will need.

A quasi-order \leq (*i.e.*, a reflexive and transitive relation) over a set S is said to be a *well-quasi-order* (wqo) if there is no infinite decreasing sequence and no infinite incomparable

sequence of elements of S . In other words if for every infinite sequence $s_1 s_2 \dots \in S^\omega$ there are two indices $i < j$ such that $s_i \leq s_j$. Given a wqo (S, \leq) and $T \subseteq S$, we define the *downward closure* of T as $\downarrow T := \{s \in S \mid \exists t \in T, s \leq t\}$ and T is *downward closed* if $\downarrow T = T$.

Given a BUDA \mathcal{A} , a first goal could be to compute the set of reachable configurations, i.e. the configurations C such that there is a data tree \mathbf{t} and a run ρ of \mathcal{A} on \mathbf{t} such that $\rho(x) = C$ for the root x of \mathbf{t} .

A naïve algorithm would be to start with the set of initial configurations and then to enrich this set, step by step, by applying the transition function of \mathcal{A} to some configurations from the set, hence preserving reachability.

This idea immediately raises some issues. The first one is that the set of reachable configurations is infinite and therefore we need a way to guarantee that the run of the algorithm will eventually stop. Second, we need to make sure we can compute the configurations reachable in one step from a finite set of configurations. In particular, as our trees are unranked, we do not know in advance how many reachable configurations we need to combine in order to derive the next ones. Moreover, in order to apply the transition function of \mathcal{A} , we need to know which of the tests of the form $\langle \mu \rangle^=$ and $\langle \mu \rangle^\neq$ are true in the data trees that make a configuration reachable.

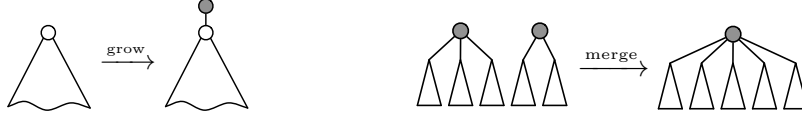
To overcome this last issue, we enrich the notion of configuration, initially a finite set of pairs $(q, d) \in Q \times \mathbb{D}$, by including all the information of the current subtree that is necessary to maintain in order to continue the simulation of the automaton from there. This information consists in a finite set of pairs of the form $(\mu, d) \in \mathcal{S} \times \mathbb{D}$ whose presence indicate that from the current node the data value d can be reached following a downward path evaluating to μ via the homomorphism h of \mathcal{A} . This enriched configuration will be henceforth called *extended configuration*.

To overcome the problem of termination we introduce a well-quasi-order on extended configurations and we show that, as far as emptiness is concerned, it is not necessary to consider extended configurations that are bigger than those already computed. We say that such a wqo is *compatible* with the transition system.¹ In other words it is enough to solve the *coverability* problem relative to that wqo instead of the *reachability problem*. Coverability can be decided by inserting a new extended configuration into the running set of reachable configurations only if the new extended configuration is incomparable to or smaller than those already present. Termination is then guaranteed by the well-quasi-order.

Finally, in order to overcome the problem coming from the tree unrankedness, we decompose a transition of \mathcal{A} into two basic steps of a transition system (see Figure 3). The first step simulates a transition of \mathcal{A} assuming the current node is the only child. The second step “merges” two configurations by simulating what \mathcal{A} would have done if the roots of the corresponding trees were identified. Repeating this last operation yields trees of arbitrary rank. Altogether the unrankedness problem is also transferred to the coverability problem.

To summarize, we associate to a given BUDA \mathcal{A} a transition system based on the configurations of \mathcal{A} and the transition relation of \mathcal{A} , we exhibit a compatible wqo, and can then decide emptiness of \mathcal{A} using the coverability algorithm sketched above.

¹In our case this will be assured by a property of the transition system (X, \rightarrow) and wqo (X, \leq) of the following form: for every $x, x', y \in X$ so that $x' \leq x$ and $x \rightarrow y$ there is $y' \in X$ so that $x' \rightarrow^* y'$ and $y' \leq y$.

Figure 3: The *grow* and *merge* operations.

4.1. Extended configurations. We define here the set of *extended configurations* of a given BUDA $\mathcal{A} = (\mathbb{A}, \mathbb{B}, Q, q_0, \delta, \mathcal{S}, h)$ in normal form.

In order to get a good intuition about the definition of extended configurations it is necessary to have a glimpse of what transitions will be performed on them. We will simulate each transition $\tau \in \delta$ by several steps of the transition system. Recall that each $\tau \in \delta$ consists of several tests and several actions. The first step of the simulation of τ will generate as many threads as there are tests and actions in τ . Each of these threads will then have the task of performing the corresponding test or action. We will also use symbols, \perp , \top and \top_g , to distinguish respectively: threads where no transition has yet been executed, threads on which a transition (other than *guess*) has been successfully applied, and threads on which a *guess* transition has been successfully applied.

Let $\text{TA} = \text{TESTS} \cup \text{ACTIONS} \cup \{\top, \top_g, \perp\}$. An *extended configuration* of \mathcal{A} is a tuple (Δ, Γ, r, m) where r and m are either true or false, Δ is a finite subset of $Q \times \text{TA} \times \mathbb{D}$ and Γ is a finite subset of $\mathcal{S} \times \mathbb{D}$ such that

$$\Gamma \text{ contains exactly one pair of the form } (h(c), d) \text{ with } c \in \mathbb{A} \times \mathbb{B}. \quad (\star)$$

This unique element of $\mathbb{A} \times \mathbb{B}$ is denoted as the *label* of the extended configuration and the unique associated data value is denoted as the *data value* of the extended configuration.

Intuitively, r says whether the current node should be treated as the root or not, m says whether the extended configuration is ready to be merged with another one or not, and Δ represents the set of ongoing threads at the current node. Its elements have the form (q, α, d) . If $\alpha \in \text{TESTS} \cup \text{ACTIONS}$ then this thread is expected to perform the corresponding test or action. If $\alpha \in \{\top, \top_g\}$ then this thread is ready to move up in the tree. If $\alpha = \perp$ then no transition has yet been applied on this thread. On the other hand, a pair $(\mu, d) \in \Gamma$ simulates the existence of a downward path evaluating to μ and whose last node carries the data value d . The condition (\star) is here for technical reasons. In particular it permits to recover the data value and label of the current tree root.

Consider $\theta = (\Delta, \Gamma, r, m)$. Remember that Δ is the set of threads, Γ is the set of pairs “(path evaluation, data value)” present in the abstracted subtree, r states whether the extended configuration is the root, and m whether it is in merge mode. In the sequel we will use the following notation for $\theta = (\Delta, \Gamma, r, m)$:

$$\begin{aligned} \Delta(d) &= \{(q, \alpha) \mid (q, \alpha, d) \in \Delta\}, \\ \Gamma(d) &= \{\mu \mid (\mu, d) \in \Gamma\}, \\ [\theta](d) &= (\Delta(d), \Gamma(d)), \\ \text{data}(\theta) &= \{d \mid (q, \alpha, d) \in \Delta \vee (\mu, d) \in \Gamma\}. \end{aligned}$$

We will also use the inverses of these, for $(R, \chi) \in \wp(Q \times \text{TA}) \times \wp(\mathcal{S})$:

$$\begin{aligned}\Delta^{-1}(R) &= \{d \mid \Delta(d) = R\}, \\ \Gamma^{-1}(\chi) &= \{d \mid \Gamma(d) = \chi\}, \\ [\theta]^{-1}(R, \chi) &= \{d \mid [\theta](d) = (R, \chi)\},\end{aligned}$$

Note that $\Delta(d)$ gives the information about the current threads carrying d in the register; $\Gamma(d)$ gives the information about downward paths that lead to the data value d ; $[\theta](d)$ is the aggregation of this information; $\text{data}(\theta)$ is the set of all data values present in θ ; $\Delta^{-1}(R)$ is the set of data values whose thread information is precisely R ; $\Gamma^{-1}(\chi)$ is the set of data values whose downward paths information is precisely χ ; and $[\theta]^{-1}(R, \chi)$ is the set of data values whose aggregated $[\theta]$ -information is precisely the pair (R, χ) .

We use the letter θ to denote an extended configuration and we write EC to denote the set of all extended configurations. Similarly, we use Θ to denote a finite set of extended configurations.

An extended configuration $\theta = (\Delta, \Gamma, r, m)$ is said to be *initial* if it corresponds to a leaf node, i.e., is such that $\Delta = \{(q_0, \perp, d)\}$ and $\Gamma = \{(h(a), d')\}$ for some $d, d' \in \mathbb{D}$ and $a \in \mathbb{A} \times \mathbb{B}$. Note that every initial extended configuration verifies condition (\star) . An extended configuration is said to be *accepting* if $\Delta = \emptyset$.

We will very often use this notation for components of extended configurations:

$$\begin{aligned}\theta &= (\Delta, \Gamma, r, m) \\ \theta' &= (\Delta', \Gamma', r', m')\end{aligned}\tag{\ddagger}$$

Two extended configurations θ_1 and θ_2 are said to be *equivalent* if they are identical modulo a bijection between data values, i.e. there is a bijection $f : \mathbb{D} \rightarrow \mathbb{D}$ such that $f(\theta_1) = \theta_2$ (with some abuse of notation). We denote this by $\theta_1 \sim \theta_2$.

Finally, we write Θ_I to denote the set of all initial extended configurations modulo \sim (i.e. a set containing at most one element for each \sim equivalence class). Note that Θ_I is *finite* and can be computed. A set of extended configurations is said to be *accepting* if it contains an accepting extended configuration.

4.2. Well-quasi-orders. We now equip EC with a well-quasi-order (EC, \preceq) .

The *profile of an extended configuration* $\theta = (\Delta, \Gamma, r, m)$, denoted by $\text{profile}(\theta)$, is the tuple (A_0, A_1, r, m) with

$$\begin{aligned}A_0 &= \{\chi \subseteq \mathcal{S} : |\Gamma^{-1}(\chi)| = 0\}, \\ A_1 &= \{\chi \subseteq \mathcal{S} : |\Gamma^{-1}(\chi)| = 1\}.\end{aligned}$$

The idea behind the definition of A_0 and A_1 is that the automata model can only test whether there exists either (i) none, (ii) exactly one, or (iii) more than one data values that are reachable by a downward path μ . If it is zero, we store this information in A_0 , if it is one we store it into A_1 . However an automaton cannot count more than this. And this is a key for decidability.

Given two extended configurations $\theta_1 = (\Delta_1, \Gamma_1, r_1, m_1)$ and $\theta_2 = (\Delta_2, \Gamma_2, r_2, m_2)$, we denote by $\theta_1 \preceq \theta_2$ the fact that

- $\text{profile}(\theta_1) = \text{profile}(\theta_2)$, and

- there is a function $f : \mathbb{D} \rightarrow \mathbb{D}$ such that $\text{data}(\theta_1) \subseteq f(\mathbb{D})$ (henceforth called the *surjectivity condition*) and for all $d \in \mathbb{D}$:
 - $\Delta_1(f(d)) \subseteq \Delta_2(d)$,
 - $\Gamma_1(f(d)) = \Gamma_2(d)$.

We write $\theta_1 \preceq_f \theta_2$ if we want to make explicit the function f witnessing the relation.

Remark 4.2. Assume that $\theta_1 \preceq_f \theta_2$ and consider a thread $(q, \alpha, d) \in \Delta_1$. Then by the surjectivity condition of f there is a data value e such that $f(e) = d$. From $\Delta_1(f(e)) \subseteq \Delta_2(e)$ it follows that (q, α, e) is a thread in Δ_2 . Similarly if θ_1 contains k threads $(q, \alpha, d_1), \dots, (q, \alpha, d_k)$ then so would θ_2 . The converse is however not true as f need not be injective. In fact, the equality between the profiles is here to guarantee a minimal version of the converse: if for some $\chi \subseteq \mathcal{S}$, θ_2 contains no data value in $\Gamma_2(\chi)$, then neither does θ_1 , and if $\Gamma_2(\chi)$ contains only one data value, so does θ_1 . We will use these remarks implicitly in the sequel.

Remark 4.3. Notice that from the previous remark and condition (\star) , it follows that if $\theta_1 \preceq_f \theta_2$ then θ_1 and θ_2 have the same label. Moreover if d is the data value of θ_2 then $f(d)$ is the data value of θ_1 .

Remark 4.4. Notice that if θ_1 and θ_2 are equivalent ($\theta_1 \sim \theta_2$), then $\theta_1 \preceq_f \theta_2$ where f is the function witnessing the equivalence.

The following lemmas are key observations:

Lemma 4.5. (EC, \preceq) is a wqo.

This is a simple consequence of Dickson's Lemma, that we state next. Let \leq_k be the componentwise order of vectors of natural numbers of dimension k . That is, $(x_1, \dots, x_k) \leq_k (y_1, \dots, y_k)$ if $x_i \leq y_i$ for all $i \in [k]$.

Lemma 4.6 (Dickson's Lemma [Dic13]). For every k , (\mathbb{N}^k, \leq_k) is a wqo.

Proof of Lemma 4.5. Note that $\theta \preceq_\iota \theta$ for all $\theta \in \text{EC}$ through the identity function $\iota : \mathbb{D} \rightarrow \mathbb{D}$, and that $\theta_1 \preceq_{f \circ g} \theta_3$ assuming $\theta_1 \preceq_f \theta_2$ and $\theta_2 \preceq_g \theta_3$. Thus, \preceq is a quasi-ordering. We next show that in fact it is a wqo.

Given an extended configuration $\theta = (\Delta, \Gamma, r, m)$, let us define $\bar{x}(\theta)$ as a function from $\wp(Q \times \text{TA}) \times \wp(\mathcal{S})$ to \mathbb{N} . We define $\bar{x}(\theta)(R, \chi) = |[\theta]^{-1}(R, \chi)|$. Notice that $\bar{x}(\theta)$ can be seen as a vector of natural numbers of dimension $k = |\wp(Q \times \text{TA}) \times \wp(\mathcal{S})|$. It then follows that for every θ_1, θ_2 , if

$$\{(R, \chi) \in \wp(Q \times \text{TA}) \times \wp(\mathcal{S}) : |[\theta_1]^{-1}(R, \chi)| = i\}$$

is equal to

$$\{(R, \chi) \in \wp(Q \times \text{TA}) \times \wp(\mathcal{S}) : |[\theta_2]^{-1}(R, \chi)| = i\}$$

for every $i \in \{0, 1\}$, and $\bar{x}(\theta_1) \leq_k \bar{x}(\theta_2)$, then $\text{profile}(\theta_1) = \text{profile}(\theta_2)$ and further $\theta_1 \preceq \theta_2$.

Consider an infinite sequence $(\theta_i)_{i \in \mathbb{N}}$. As there are only finitely many possible values for profile , there must be an infinite subsequence $(\theta'_i)_{i \in \mathbb{N}}$ of $(\theta_i)_{i \in \mathbb{N}}$ so that $\text{profile}(\theta'_i) = \text{profile}(\theta'_j)$ for all $i \neq j$. By Dickson's Lemma (Lemma 4.6), there are indices $i < j$ so that $\bar{x}(\theta'_i) \leq_k \bar{x}(\theta'_j)$. Hence, there are indices $i < j$ so that $\text{profile}(\theta_i) = \text{profile}(\theta_j)$ and $\bar{x}(\theta_i) \leq_k \bar{x}(\theta_j)$. This means that $|[\theta_i]^{-1}(R, \chi)| \leq |[\theta_j]^{-1}(R, \chi)|$ for every R, χ . Let $f : \text{data}(\theta_i) \rightarrow \text{data}(\theta_j)$ be so that $f([\theta_j]^{-1}(R, \chi)) = [\theta_i]^{-1}(R, \chi)$ for every R, χ . Note that $\Delta_i(f(d)) = \Delta_j(d)$ and $\Gamma_i(f(d)) = \Gamma_j(d)$ for every d . Thus, $\theta_i \preceq_f \theta_j$, and hence (EC, \preceq) is a wqo. \square

Finally we will need the fact that the set of accepting extended configurations is downward closed.

Lemma 4.7. *If $\theta \in \text{EC}$ is accepting and $\theta' \in \text{EC}$ is such that $\theta' \preceq \theta$ then θ' is accepting.*

Proof. If $\theta' \preceq_f \theta$ then, by the surjectivity condition of f , we have that for all $d \in \text{data}(\theta')$, $\Delta'(d) \subseteq \Delta(d')$ for some d' such that $f(d') = d$. As θ is accepting this implies that $\Delta(d') = \emptyset$ for all $d' \in \mathbb{D}$. Hence θ' is also accepting. \square

4.3. Transition system. We now equip EC with a transition relation \rightarrow that reflects the transition function δ of \mathcal{A} . As mentioned earlier, we decompose each transition of \mathcal{A} into several basic steps. The first kind can be viewed as ϵ -transitions, each of them concerning a single thread, which perform a basic test or action but without moving up in the tree. When all threads of an extended configuration have performed their ϵ -transition, the extended configuration switches to a merging state and is ready for the next step. The second kind merges several extended configurations with the same label and data value that are in a merge state (*i.e.*, $m = \text{true}$) in order to combine their immediate subtrees, this operation is merely a union. A third kind concludes the simulation of the transition by “adding a root” to an extended configuration over a data tree.

We will denote by $\xrightarrow{\text{grow}}$ the third kind of transition, by $\xrightarrow{\text{merge}}$, the second kind, and by \rightarrow_ϵ the union of all the ϵ -transitions. In order to obtain our compatibility result with respect to the partial order, we will need one extra transition $\xrightarrow{\text{inc}}$ that makes our trees in some sense “fatter”, by duplicating immediate subtrees of the root, and whose purpose will be essential for showing compatibility for the $\xrightarrow{\text{merge}}$ transition (Lemma 4.15). The union of all these transitions will form the transition relation \rightarrow of EC.

We start with the description for ϵ -transitions. Those are defined in a straightforward way in order to simulate the tests and actions of the initial automaton.

Given two extended configurations $\theta_1 = (\Delta_1, \Gamma_1, r_1, m_1)$ and $\theta_2 = (\Delta_2, \Gamma_2, r_2, m_2)$, we say that $\theta_1 \rightarrow_\epsilon \theta_2$ if $m_1 = m_2 = \text{false}$ (the merge information is used for simulating an *up*-transition as will be explained later), $r_2 = r_1$ (whether the current node is the root or not should not change), θ_1 and θ_2 have the same label and data value, $\Gamma_2 = \Gamma_1$ and, furthermore, one of the following holds:

- (1) $\theta_1 \xrightarrow{\delta} \theta_2$. This transition can happen if there is $(q, \perp, d) \in \Delta_1$ for some state $q \in Q$ and data $d \in \mathbb{D}$. In this case, for some transition $\tau \in \delta$, θ_2 is such that: $\Delta_2 = (\Delta_1 \setminus \{(q, \perp, d)\}) \cup \{(q, \alpha_i, d) : i \leq |\tau|\}$, where $(\alpha_i)_{i \leq |\tau|}$ are all the tests and actions occurring in τ .
- (2) $\theta_1 \xrightarrow{\text{univ}} \theta_2$. This transition can happen if there is $(q, \text{univ}(p), d) \in \Delta_1$ for some states $p, q \in Q$ and data $d \in \mathbb{D}$. In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, \text{univ}(p), d)\}) \cup \{(p, \top, e) : \exists \mu . (\mu, e) \in \Gamma_1\}$.
- (3) $\theta_1 \xrightarrow{\text{guess}} \theta_2$. This transition can happen if there is $(q, \text{guess}(p), d) \in \Delta_1$ for some states $p, q \in Q$ and data $d \in \mathbb{D}$. In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, \text{guess}(p), d)\}) \cup \{(p, \top_g, d')\}$ for some $d' \in \mathbb{D}$.
- (4) $\theta_1 \xrightarrow{\text{store}} \theta_2$. This transition can happen if there is $(q, \text{store}(p), d) \in \Delta_1$ for some states $p, q \in Q$ and data $d \in \mathbb{D}$. In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, \text{store}(p), d)\}) \cup \{(p, \top, d')\}$ where d' is the data value of both θ_1 and θ_2 given by (\star) .

- (5) $\theta_1 \xrightarrow{\text{keep}} \theta_2$. This transition can happen if there is $(q, \text{keep}(p), d) \in \Delta_1$ for some states $p, q \in Q$ and data $d \in \mathbb{D}$. In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, \text{keep}(p), d)\}) \cup \{(p, \top, d)\}$.
- (6) $\theta_1 \xrightarrow{\text{accept}} \theta_2$. This transition can happen if there is $(q, \text{accept}, d) \in \Delta_1$ for some state $q \in Q$ and data $d \in \mathbb{D}$. In this case θ_2 is such that $\Delta_2 = \Delta_1 \setminus \{(q, \text{accept}, d)\}$.
- (7) $\theta_1 \xrightarrow{\langle \mu \rangle^-} \theta_2$. This transition can happen if there is $(q, \langle \mu \rangle^-, d) \in \Delta_1$ for some state $q \in Q$ and data $d \in \mathbb{D}$ with $(\mu, d) \in \Gamma_1$. In this case θ_2 is such that $\Delta_2 = \Delta_1 \setminus \{(q, \langle \mu \rangle^-, d)\}$.
- (8) $\theta_1 \xrightarrow{\langle \mu \rangle^\neq} \theta_2$. This transition can happen if there is $(q, \langle \mu \rangle^\neq, d) \in \Delta_1$ for some state $q \in Q$ and data $d \in \mathbb{D}$ and there exists a data $e \in \mathbb{D}$, $e \neq d$ such that $(\mu, e) \in \Gamma_1$. In this case θ_2 is such that $\Delta_2 = \Delta_1 \setminus \{(q, \langle \mu \rangle^\neq, d)\}$.
- (9) $\theta_1 \xrightarrow{\text{root}} \theta_2$. This transition can happen if $r_1 = \text{true}$ and there is $(q, \text{root}, d) \in \Delta_1$ for some state $q \in Q$ and data $d \in \mathbb{D}$. In this case θ_2 is such that $\Delta_2 = \Delta_1 \setminus \{(q, \text{root}, d)\}$.
- (10) $\theta_1 \xrightarrow{q} \theta_2$. This transition can happen there is $(q, q, d) \in \Delta_1$ for some data $d \in \mathbb{D}$. In this case θ_2 is such that $\Delta_2 = \Delta_1 \setminus \{(q, q, d)\}$.
- (11) The negation of these tests: \bar{q} , $\overline{\langle \mu \rangle^-}$, $\overline{\langle \mu \rangle^\neq}$ and $\overline{\text{root}}$, are defined in a similar way. For instance $\theta_1 \xrightarrow{\overline{\langle \mu \rangle^\neq}} \theta_2$ can happen if there is $(q, \overline{\langle \mu \rangle^\neq}, d) \in \Delta_1$ for some state $q \in Q$ and data $d \in \mathbb{D}$ but no data $e \in \mathbb{D}$, $e \neq d$ such that $(\mu, e) \in \Gamma_1$. In this case θ_2 is such that $\Delta_2 = \Delta_1 \setminus \{(q, \overline{\langle \mu \rangle^\neq}, d)\}$.

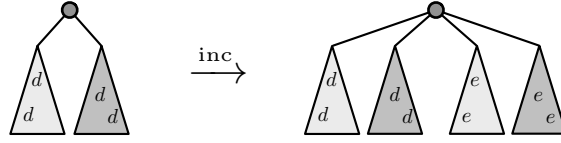
Notice that we do not include transitions for the tests **eq**, **leaf**, *a* and *b* as we work with abstractions of BUDA in normal form (cf. Proposition 3.4).

Remark 4.8. *Notice that all the transitions above do not modify the profile of the extended configuration as their Γ part remains untouched.*

As mentioned earlier, for technical reasons we will need a transition that makes our trees fatter. The idea is that this transition corresponds to duplicating an immediate subtree of the root, using a fresh new name for one of its data value of a given type, where the type of a data value d with an extended configuration θ is $[\theta](d)$. This transition assumes the same constraints as for \rightarrow_ϵ except that we no longer have $\Gamma_2 = \Gamma_1$.

- (12) $\theta_1 \xrightarrow{\text{inc}(S, \chi)} \theta_2$. This transition can happen if $|\theta_1^{-1}(S, \chi)| \geq 1$ and $|(\Gamma_1)^{-1}(\chi)| \geq 2$. Then θ_2 is such that $\text{data}(\theta_2) = \text{data}(\theta_1) \cup \{e\}$ for some data $e \notin \text{data}(\theta_1)$, $[\theta_2](e) = (\hat{S}, \chi)$ where $\hat{S} = S \setminus \{(q, \top_g) : (q, \top_g) \in S\}$, and for all $d \neq e$, $[\theta_2](d) = [\theta_1](d)$.

Observe that the conditions required for using $\xrightarrow{\text{inc}(S, \chi)}$ enforce that the truth value of any test is not changed. Indeed, condition $|\theta_1^{-1}(S, \chi)| \geq 1$ says that there are at least one data value of type (S, χ) . On the other hand, condition $|(\Gamma_1)^{-1}(\chi)| \geq 2$ ensures in fact this is not a ‘special’ data value in the sense that it is the only data value accessible through χ . This is essential to make sure that there is at least one data value that can be ‘copied’ multiple times since the automaton can express properties like “there is exactly one data value accessible through $\mu \in \chi$ ”. This operation is, intuitively, a way of duplicating the derivation of θ_1 in the transition system. The idea is that it would correspond to a run on an expansion of the tree that would yield the extended configuration θ_1 , by duplicating the immediate subtrees of the root, renaming the data value d with e (see Figure 4). However, note that the \top_g -flagged threads are not duplicated in this operation. This corresponds to


 Figure 4: The *inc* operation.

assuming that when applying this operation, the new threads do not guess a ‘new’ value e but they stick to guessing the ‘old’ one d . This ensures that θ_2 reaches all accepting configurations that θ_1 reaches.

We use $\xrightarrow{\text{inc}}$ as the union of all $\xrightarrow{\text{inc}(S, \chi)}$ for all (S, χ) .

Remark 4.9. Notice that the preconditions of $\xrightarrow{\text{inc}(S, \chi)}$ implies that the profile of the extended configuration is not changed.

We now turn to the simulation of an extended configuration moving up in the tree. We split this part into two phases: adding a new root symbol and merging the roots (see Figure 3).

- (13) $\theta_1 \xrightarrow{\text{grow}} \theta_2$. Given two extended configurations θ_1 and θ_2 as above, we define $\theta_1 \xrightarrow{\text{grow}} \theta_2$ if $r_1 = m_1 = \text{false}$, and for all $(q, \alpha, d) \in \Delta_1$, we have $\alpha \in \{\top, \top_g\}$. In this case θ_2 is such that

$$\begin{aligned} \Delta_2 &= \{(q, \perp, d) : (q, \alpha, d) \in \Delta_1\}, \text{ and} \\ \Gamma_2 &= \{(\mu', e) : (\mu, e) \in \Gamma_1, \mu' = h(c) \cdot \mu\} \cup \{(h(c), d')\}, \end{aligned}$$

for some $c \in \mathbb{A} \times \mathbb{B}$ and $d' \in \mathbb{D}$. Notice that c and d' are then the label and data value of θ_2 and that r_2 and m_2 could be *true* or *false*. As a consequence of the normal form (NF1) of the semigroup, this operation preserves property (\star) .

- (14) $\theta_1, \theta_2 \xrightarrow{\text{merge}} \theta_0$. Given 3 extended configurations $\theta_1 = (\Delta_1, \Gamma_1, r_1, m_1)$, $\theta_2 = (\Delta_2, \Gamma_2, r_2, m_2)$, $\theta_0 = (\Delta_0, \Gamma_0, r_0, m_0)$ we say that $\theta_1, \theta_2 \xrightarrow{\text{merge}} \theta_0$ if they all have the same label and data value, $m_1 = m_2 = \text{true}$, $r_1 = r_2 = r_0$, $\Delta_0 = \Delta_1 \cup \Delta_2$, and $\Gamma_0 = \Gamma_1 \cup \Gamma_2$. Notice that this operation preserves property (\star) and that m_0 could be either *true* or *false*.

Given a finite set Θ of extended configurations and $\theta \in \text{EC}$ we write $\Theta \Rightarrow \theta$ if there are extended configurations in Θ generating θ according to the transition rules 1 to 14. Based on this, we define by induction $\Theta \Rightarrow^+ \theta$ if:

- (1) $\Theta \Rightarrow \theta$, or
- (2) there is an extended configuration $\theta' \in \text{EC}$ such that $\Theta \Rightarrow \theta'$ and $\Theta \cup \{\theta'\} \Rightarrow^+ \theta$.

In the definition of the transition system, the m flag is simply used to constrain the transition system to have all its $\xrightarrow{\text{merge}}$ operations right after $\xrightarrow{\text{grow}}$ and before any \rightarrow_ϵ . Thus any transition \Rightarrow^+ complies to the following regular expression:

$$((\rightarrow_\epsilon \mid \xrightarrow{\text{inc}})^* \xrightarrow{\text{grow}} (\xrightarrow{\text{merge}})^*)^* (\rightarrow_\epsilon \mid \xrightarrow{\text{inc}})^* . \quad (\dagger)$$

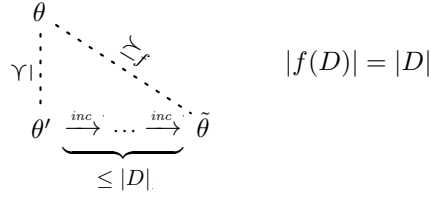


Figure 5: Lemma 4.11.

4.4. Compatibility. Given two finite sets of extended configurations Θ and Θ' we write $\Theta \leq_{\min} \Theta'$ if for all $\theta' \in \Theta'$ there is $\theta \in \Theta$ such that $\theta \preceq \theta'$. That is, every element from Θ' is *minorized* by an element of Θ .

Given a finite set Θ of extended configurations, we denote by Θ^\sim the set of all extended configurations equivalent to some configuration in Θ . In other words, Θ^\sim is the closure of Θ under bijections on data values.

The following proposition essentially shows that bigger extended configurations can be safely ignored in order to test for emptiness. This will be the main technical contribution of this section.

Proposition 4.10. *If Θ, Θ' are finite sets of extended configurations such that $\Theta \leq_{\min} \Theta'$ and $\theta' \in \text{EC}$ is such that $\Theta' \Rightarrow \theta'$ then there exists $\theta \preceq \theta'$ such that $\Theta^\sim \Rightarrow^n \theta$, for some n computable from Θ .*

The key ingredient to prove this, is to show that transitions on EC are compatible with the order of EC. We treat the case of $\xrightarrow{\text{merge}}$ in a separate lemma later.

We first show useful lemmas illustrating the power of $\xrightarrow{\text{inc}}$.

Lemma 4.11 (Figure 5). *Let $\theta, \theta' \in \text{EC}$ such that $\theta' \preceq \theta$. Let $D \subseteq \mathbb{D}$ be a finite set of data values. Then there exists $\tilde{\theta}$ such that:*

- (1) $\theta' \xrightarrow{(\text{inc})^n} \tilde{\theta}$, for some $n \leq |D|$,
- (2) $\tilde{\theta} \preceq_f \theta$,
- (3) f is injective on D .

Proof. Fix $\theta = (\Delta, \Gamma, r, m)$ and $\theta' = (\Delta', \Gamma', r', m')$. Assume that $\theta' \preceq_f \theta$. We will modify θ' and f until f becomes injective.

If f is not injective then we have $d, d' \in D$, $d \neq d'$ but $f(d) = f(d')$.

Let $(S, \chi) = [\theta'](f(d))$. Then we have $\chi = \Gamma(d) = \Gamma'(f(d)) = \Gamma'(f(d')) = \Gamma(d')$. From $\theta' \preceq \theta$ it follows that $\text{profile}(\theta) = \text{profile}(\theta')$. Therefore θ' must contain another data value $e \neq f(d)$ such that $\Gamma'(e) = \chi$. Hence we can apply $\theta' \xrightarrow{\text{inc}(S, \chi)} \theta''$ adding a copy \tilde{d} of $f(d)$, so that $[\theta''](\tilde{d}) = (\hat{S}, \chi)$ where $\hat{S} = S \setminus \{(q, \top_g) : (q, \top_g) \in S\}$. The reader can verify that the resulting extended configuration θ'' is such that $\theta'' \preceq_g \theta$ where g is the mapping identical to f except that $g(d) = \tilde{d} \neq f(d') = g(d')$. Repeating this operation at most $|D|$ times we obtain the desired $\tilde{\theta}$. \square

Lemma 4.12. *Let $\theta, \theta' \in \text{EC}$ as in (\ddagger) such that $\theta' \preceq \theta$. Then there exists $\tilde{\theta}$ such that:*

- (1) $\theta' \xrightarrow{(\text{inc})^n} \tilde{\theta}$, for some $n \leq 3 \cdot 2^{|S|}$,
- (2) $\tilde{\theta} \preceq \theta$,
- (3) for all $\chi \subseteq S$, if θ contains strictly more than 2 data values d with $\Gamma(d) = \chi$, then so does $\tilde{\theta}$.

Proof. We apply Lemma 4.11 to the set D defined by selecting for each $\chi \subseteq \mathcal{S}$ three data values e such that $\Gamma(e) = \chi$, if this is possible. This gives an upper bound of $3 \cdot 2^{|\mathcal{S}|}$ applications of $\xrightarrow{\text{inc}}$. \square

Lemma 4.13. *Let $\theta, \theta' \in \text{EC}$ as in (\ddagger) such that $\theta' \preceq \theta$ and let $d \in \mathbb{D}$. Then there exists $\tilde{\theta}$ such that:*

- (1) $\theta' \xrightarrow{(\text{inc})^n} \tilde{\theta}$, for some $n \leq 2^{|\mathcal{S}|} + 1$,
- (2) $\tilde{\theta} \preceq_f \theta$,
- (3) $f^{-1}(f(d)) = \{d\}$.

Proof. For every $S \subseteq Q \times \text{TA}$, let $E_S \subseteq \mathbb{D}$ be a subset of $f^{-1}(f(d))$ containing one data value $e \neq d$ so that $\Delta(e) = S$, or no data values otherwise.

Then, we can apply Lemma 4.11 with $D = \{d\} \cup \bigcup_{S \subseteq Q \times \text{TA}} E_S$ and we obtain an extended configuration $\tilde{\theta}$ so that $\theta' \xrightarrow{(\text{inc})^{\leq |D|}} \tilde{\theta}$ and $\tilde{\theta} \preceq_{f'} \theta$ for some f' that is injective on D . From f' we can create the desired f , by sending each $e \neq d$, $e \in f^{-1}(f(d))$ to $f'(e')$ where $\{e'\} = E_{\Delta(e)}$. \square

We now show the first monotonicity lemma regarding all transitions except $\xrightarrow{\text{merge}}$. In this case n does not depend on the extended configurations involved, only on size of the monoid \mathcal{S} .

Lemma 4.14. *Let θ_1, θ_2 and θ'_1 be extended configurations such that $\theta_1 \rightarrow \theta_2$ and $\theta'_1 \preceq \theta_1$, where \rightarrow is either $\xrightarrow{\text{grow}}$ or in \rightarrow_ϵ . Then there exists an extended configuration θ'_2 such that $\theta'_2 \preceq \theta_2$ and $\theta'_1 \xrightarrow{n} \theta'_2$ for some $n \leq 4 \cdot 2^{|\mathcal{S}|} + 1$.*

Proof. This is done by a case analysis depending on where \rightarrow comes from. Throughout the proof we use the following notation for $i \in \{1, 2\}$: $\theta_i = (\Delta_i, \Gamma_i, r_i, m_i)$, similarly for θ'_i . Moreover a_i and d_i denote the label and data value associated to θ_i (similarly for θ'_i). We also denote by f a function witnessing $\theta'_1 \preceq \theta_1$. In particular (recall Remark 4.3) we have $a_1 = a'_1$ and $f(d_1) = d'_1$.

By Lemma 4.12, applying at most $3 \cdot 2^{|\mathcal{S}|}$ transition steps on θ'_1 we can now further assume that the conclusion of Lemma 4.12 holds for θ_1 and θ'_1 . We now turn to the case analysis. Each case will transform further θ'_1 in order to get the desired property. This last transformation will require at most $\max(3, 2^{|\mathcal{S}|} + 1)$ steps.

- (a) Suppose $\theta'_1 \preceq_f \theta_1 \xrightarrow{\text{grow}} \theta_2$.

We first show that $\xrightarrow{\text{grow}}$ can also be applied to θ'_1 . Assume towards a contradiction that this is not the case. Then θ'_1 contains a thread (q, α, e') such that $\alpha \notin \{\top, \top_g\}$. Then by Remark 4.2 a thread of the form (q, α, e) , with $f(e) = e'$ is in θ_1 , a contradiction from the fact that $\xrightarrow{\text{grow}}$ was applied to θ_1 .

Let θ'_2 be the extended configuration such that $\theta'_1 \xrightarrow{\text{grow}} \theta'_2$, $r'_2 = r_2$, $a'_2 = a_2$ and $d'_2 = f(d_2)$. We show that $\theta'_2 \preceq_f \theta_2$ concluding this case. Note that f still satisfies the surjectivity condition. It remains to show that $\text{profile}(\theta'_2) = \text{profile}(\theta_2)$, and for every d , $\Delta'_2(f(d)) \subseteq \Delta_2(d)$ and $\Gamma'_2(f(d)) = \Gamma_2(d)$.

- (i) $\Delta'_2(f(d)) \subseteq \Delta_2(d)$ and $\Gamma'_2(f(d)) = \Gamma_2(d)$. By definition of $\xrightarrow{\text{grow}}$, we have that $\Gamma_2(d)$ is completely determined from $\Gamma_1(d)$, a_2 and the fact that d is equal to d_2 or not. In a similar way, whether a thread (q, \perp) is in $\Delta_2(d)$ is determined by whether (q, \top) or (q, \top_g) is in $\Delta_1(d)$. But by construction, modulo f , all these facts are

identical for θ'_2 and θ_2 . Hence, since $\Delta'_1(f(d)) \subseteq \Delta_1(d)$ and $\Gamma'_1(f(d)) = \Gamma_1(d)$, we have that $\Delta'_2(f(d)) \subseteq \Delta_2(d)$ and $\Gamma'_2(f(d)) = \Gamma_2(d)$.

- (ii) $profile(\theta'_2) = profile(\theta_2)$. We already have by construction $r'_2 = r_2$ and $m'_2 = m_2$. From the previous item we have for all χ , $|\Gamma'_2{}^{-1}(\chi)| > 0$ implies $|\Gamma_2{}^{-1}(\chi)| > 0$ and that $|\Gamma'_2{}^{-1}(\chi)| > 1$ implies $|\Gamma_2{}^{-1}(\chi)| > 1$. It remains to show the converse of these implications. We only show the second one as the first one is similar and simpler. We thus only show that if $|\Gamma_2{}^{-1}(\chi)| > 1$ then $|\Gamma'_2{}^{-1}(\chi)| > 1$.

Assume we have two distinct data values $e_1 \neq e_2$ such that $\Gamma_2(e_1) = \Gamma_2(e_2) = \chi$. Let $\chi_1 = \Gamma_1(e_1)$ and $\chi_2 = \Gamma_1(e_2)$. From $profile(\theta'_1) = profile(\theta_1)$ we know that there exist data values e'_1 and e'_2 such that $\chi_1 = \Gamma'_1(e'_1)$ and $\chi_2 = \Gamma'_1(e'_2)$. Even in the case where $\chi_1 = \chi_2$ we get from $profile(\theta'_1) = profile(\theta_1)$ that e'_1 and e'_2 can be chosen distinct. Moreover, as $\Gamma'_1(f(d_2)) = \Gamma_1(d_2)$, if $e_1 = d_2$ (resp. $e_2 = d_2$) then we can pick $e'_1 = f(d_2)$ (resp. $e'_2 = f(d_2)$). In the case where both e_1 and e_2 are different from d_2 we pick e'_1 and e'_2 different from $f(d_2)$. The latter is always possible because of Item (3) of Lemma 4.12. Altogether we have $e_i = d_2$ iff $e'_i = f(d_2)$ for $i = 1, 2$.

Hence, since $a_2 = a'_2$ and $f(d_2) = d'_2$, by definition of $\xrightarrow{\text{grow}}$ we have $\Gamma'_2(e'_1) = \Gamma'_2(e'_2) = \chi$.

- (b) Suppose $\theta'_1 \preceq_f \theta_1 \xrightarrow{\text{univ}} \theta_2$.

By definition of $\xrightarrow{\text{univ}}$ there is $(q, \text{univ}(p), d) \in \Delta_1$ and $a_2 = a_1$, $d_2 = d_1$, $\Gamma_2 = \Gamma_1$, and $\Delta_2 = (\Delta_1 \setminus \{(q, \text{univ}(p), d)\}) \cup \{(p, \top, e) : \exists \mu . (\mu, e) \in \Gamma_1\}$. Applying Lemma 4.13 to d , we can assume that $f^{-1}(f(d)) = \{d\}$.

- If $(q, \text{univ}(p), f(d)) \notin \Delta'_1$, then we show that $\theta'_1 \preceq_f \theta_2$. First, it is immediate that $profile(\theta'_1) = profile(\theta_2)$ since $profile(\theta_2) = profile(\theta_1)$ by definition of $\xrightarrow{\text{univ}}$ (recall Remark 4.8) and $profile(\theta_1) = profile(\theta'_1)$ since $\theta'_1 \preceq \theta_1$. Furthermore:
 - for all $\hat{e} \neq d$, we have that $\Gamma'_1(f(\hat{e})) = \Gamma_1(\hat{e}) = \Gamma_2(\hat{e})$ and $\Delta'_1(f(\hat{e})) \subseteq \Delta_1(\hat{e}) \subseteq \Delta_2(\hat{e})$,
 - for d we have $\Gamma'_1(f(d)) = \Gamma_1(d) = \Gamma_2(d)$ and $\Delta'_1(f(d)) \subseteq \Delta_1(d) \setminus \{(q, \text{univ}(p))\} \subseteq \Delta_2(d)$, since $(q, \text{univ}(p)) \notin \Delta'_1(f(d))$.
- If $(q, \text{univ}(p), f(d)) \in \Delta'_1$, we perform a $\xrightarrow{\text{univ}}$ transition on θ'_1 and obtain θ'_2 so that $\theta'_1 \xrightarrow{\text{univ}} \theta'_2$ and $\theta'_2 \preceq \theta_2$. Let θ'_2 be so that $r'_2 = r'_1$, $m'_2 = m'_1$, $\Gamma'_2 = \Gamma'_1$, and $\Delta'_2 = (\Delta'_1 \setminus \{(q, \text{univ}(p), f(d))\}) \cup \{(p, \top, e) : \exists \mu . (\mu, e) \in \Gamma'_1\}$. Hence, $\theta'_1 \xrightarrow{\text{univ}} \theta'_2$. We now show that $\theta'_2 \preceq_f \theta_2$. We have that $profile(\theta'_2) = profile(\theta'_1)$ and $profile(\theta_2) = profile(\theta_1)$ by definition of $\xrightarrow{\text{univ}}$, and that $profile(\theta_1) = profile(\theta'_1)$ by $\theta'_1 \preceq \theta_1$. Hence, $profile(\theta'_2) = profile(\theta_2)$.
 - For all \hat{e} so that $f(\hat{e}) \neq f(d)$ (and hence $\hat{e} \neq d$), we have that $\Gamma'_2(f(\hat{e})) = \Gamma'_1(f(\hat{e})) = \Gamma_1(\hat{e}) = \Gamma_2(\hat{e})$ and $\Delta'_2(f(\hat{e})) = \Delta'_1(f(\hat{e})) \cup \{(p, \top) : \Gamma'_1(f(\hat{e})) \neq \emptyset\} \subseteq \Delta_1(\hat{e}) \cup \{(p, \top) : \Gamma_1(\hat{e}) \neq \emptyset\} = \Delta_2(\hat{e})$.
 - For d , we have $\Gamma'_2(f(d)) = \Gamma'_1(f(d)) = \Gamma_1(d) = \Gamma_2(d)$ and, on the other hand, $\Delta'_2(f(d)) = \Delta'_1(f(d)) \setminus \{(q, \text{univ}(p))\} \cup \{(p, \top) : \Gamma'_1(f(d)) \neq \emptyset\} \subseteq \Delta_1(d) \setminus \{(q, \text{univ}(p))\} \cup \{(p, \top) : \Gamma_1(d) \neq \emptyset\} = \Delta_2(d)$.
 - Finally, for any \hat{e} such that $f(\hat{e}) = f(d)$ we have that $\hat{e} = d$ since $f^{-1}(f(d)) = \{d\}$, and we apply the previous item.

Hence, $\theta'_2 \preceq_f \theta_2$.

- (c) Suppose $\theta'_1 \preceq_f \theta_1 \xrightarrow{\text{guess}} \theta_2$.

By definition of $\xrightarrow{\text{guess}}$ there is $(q, \text{guess}(p), d) \in \Delta_1$ and $a_2 = a_1$, $d_2 = d_1$, $r_2 = r_1$, $m_2 = m_1$, $\Gamma_2 = \Gamma_1$ and $\Delta_2 = (\Delta_1 \setminus \{(q, \text{guess}(p), d)\}) \cup \{(p, \top_g, e)\}$ for some data value e . We show that either $\theta'_1 \preceq \theta_2$ or there is some θ'_2 so that $\theta'_1 \xrightarrow{\text{guess}} \theta'_2 \preceq \theta_2$. By applying Lemma 4.11 for $D = \{d, e\}$, we can assume without any loss of generality that if $d \neq e$ then $f(d) \neq f(e)$.

- If $(q, \text{guess}(p), f(d)) \notin \Delta'_1$, then we show that $\theta'_1 \preceq_f \theta_2$. First, it is immediate that $\text{profile}(\theta'_1) = \text{profile}(\theta_2)$ since $\text{profile}(\theta_2) = \text{profile}(\theta_1)$ by definition of $\xrightarrow{\text{guess}}$ and $\text{profile}(\theta_1) = \text{profile}(\theta'_1)$ since $\theta'_1 \preceq \theta_1$. For all data values \hat{e} we have $\Gamma'_1(f(\hat{e})) = \Gamma_1(\hat{e}) = \Gamma_2(\hat{e})$. Furthermore:
 - for all $\hat{e} \notin \{d, e\}$, we have $\Delta'_1(f(\hat{e})) \subseteq \Delta_1(\hat{e}) = \Delta_2(\hat{e})$,
 - for d we have $\Delta'_1(f(d)) \subseteq \Delta_1(d) \setminus \{(q, \text{guess}(p))\} \subseteq \Delta_2(d)$ (since $(q, \text{guess}(p)) \notin \Delta'_1(f(d))$),
 - for e , assuming that $e \neq d$ (otherwise the previous item applies), we have $\Delta'_1(f(e)) \subseteq \Delta_1(e) \subseteq \Delta_2(e)$.

Hence, $\theta'_1 \preceq_f \theta_2$.

- If $(q, \text{guess}(p), f(d)) \in \Delta'_1$, we show that can perform the same action and we obtain θ'_2 so that $\theta'_1 \xrightarrow{\text{guess}} \theta'_2$ and $\theta'_2 \preceq \theta_2$. Let θ'_2 be so that $r'_2 = r'_1$, $m'_2 = m'_1$, $\Gamma'_2 = \Gamma'_1$, and $\Delta'_2 = (\Delta'_1 \setminus \{(q, \text{guess}(p), f(d))\}) \cup \{(p, \top_g, f(e))\}$. Hence, $\theta'_1 \xrightarrow{\text{guess}} \theta'_2$.

We now show that $\theta'_2 \preceq_f \theta_2$. We have that $\text{profile}(\theta'_2) = \text{profile}(\theta'_1)$ and $\text{profile}(\theta_2) = \text{profile}(\theta_1)$ by definition of $\xrightarrow{\text{guess}}$, and that $\text{profile}(\theta_1) = \text{profile}(\theta'_1)$ by $\theta'_1 \preceq \theta_1$. Hence, $\text{profile}(\theta'_2) = \text{profile}(\theta_2)$. By construction we also have for all \hat{e} , $\Gamma'_2(f(\hat{e})) = \Gamma'_1(f(\hat{e})) = \Gamma_1(\hat{e}) = \Gamma_2(\hat{e})$. Furthermore:

- for all $\hat{e} \notin \{d, e\}$, we have that $\Delta'_2(f(\hat{e})) = \Delta'_1(f(\hat{e})) \subseteq \Delta_1(\hat{e}) = \Delta_2(\hat{e})$,
- for d , we have
 - * if $d \neq e$, then (since we assumed we have applied Lemma 4.11 with $D = \{d, e\}$) we have $f(d) \neq f(e)$, and hence $\Delta'_2(f(d)) = \Delta'_1(f(d)) \setminus \{(q, \text{guess}(p))\} \subseteq \Delta_1(d) \setminus \{(q, \text{guess}(p))\} = \Delta_2(d)$,
 - * if $d = e$, $\Delta'_2(f(d)) = (\Delta'_1(f(d)) \setminus \{(q, \text{guess}(p))\}) \cup \{(p, \top_g)\} \subseteq (\Delta_1(d) \setminus \{(q, \text{guess}(p))\}) \cup \{(p, \top_g)\} = \Delta_2(d)$,
- for e , assuming that $e \neq d$ (otherwise the previous item applies), we have $\Delta'_2(f(e)) = \Delta'_1(f(e)) \cup \{(p, \top_g)\} \subseteq \Delta_1(e) \cup \{(p, a)\} = \Delta_2(e)$.

Hence, $\theta'_2 \preceq_f \theta_2$.

- (d) Suppose we have $\theta'_1 \preceq_f \theta_1 \xrightarrow{\text{inc}(S, \chi)} \theta_2$.

By definition of $\xrightarrow{\text{inc}(S, \chi)}$, this implies that $|\theta_1^{-1}(S, \chi)| \geq 1$ and $|\Gamma_1^{-1}(\chi)| \geq 2$.

Let d be the only data value in $\text{data}(\theta_2) \setminus \text{data}(\theta_1)$ given by the definition of $\xrightarrow{\text{inc}(S, \chi)}$. Let $\hat{d} \in [\theta_1]^{-1}(S, \chi)$. We further have that $[\theta_2](e) = [\theta_1](e)$ for all $e \neq d$, and $[\theta_2](d) = (\hat{S}, \chi)$ where $\hat{S} = S \setminus \{(q, \top_g) : (q, \top_g) \in S\}$.

From $\theta'_1 \preceq_f \theta_1$ it follows that $[\theta'_1](f(\hat{d})) = (S', \chi)$ for some $S' \subseteq S$. Further, by $\text{profile}(\theta'_1) = \text{profile}(\theta_1)$ we have that $|\Gamma_1^{-1}(\chi)| \geq 2$. We can then apply $\xrightarrow{\text{inc}(S', \chi)}$ to θ'_1 . Let θ'_2 be so that $\theta'_1 \xrightarrow{\text{inc}(S', \chi)} \theta'_2$ and $f(d)$ is chosen as the new data value duplicating $f(\hat{d})$. Note that $\Delta'_1(f(d)) \subseteq \Delta_1(d)$ and $\Gamma'_1(f(d)) = \Gamma_1(d)$ and $\Delta_1(d) = \emptyset = \Gamma_1(d)$, and hence $f(d)$ has all the desired properties. We show that $\theta'_2 \preceq_f \theta_2$ concluding this case. Note that f satisfies the surjectivity condition. It remains to show that $\text{profile}(\theta'_2) = \text{profile}(\theta_2)$ and for all $e \in \mathbb{D}$ we have $\Delta'_2(f(e)) \subseteq \Delta_2(e)$ and $\Gamma'_2(f(e)) = \Gamma_2(e)$.

- (i) $profile(\theta'_2) = profile(\theta_2)$. This is immediate from the fact that $profile(\theta'_2) = profile(\theta'_1)$, $profile(\theta_2) = profile(\theta_1)$ by definition of $\xrightarrow{\text{inc}}$ (recall Remark 4.9), and $profile(\theta'_1) = profile(\theta_1)$ by definition of $\theta'_1 \preceq \theta_1$.
- (ii) $\Delta'_2(f(e)) \subseteq \Delta_2(e)$ and $\Gamma'_2(f(e)) = \Gamma_2(e)$. For every $e \neq d$, it is immediate as $[\theta](e)$ is not affected by $\xrightarrow{\text{inc}}$.

For the data value d , note that from $\theta'_1 \preceq_f \theta_1$ and the definition of $\xrightarrow{\text{inc}}$ it follows that $\Gamma'_2(f(\hat{d})) = \Gamma'_1(f(\hat{d})) = \Gamma_1(\hat{d}) = \Gamma_1(d) = \Gamma_2(d)$, as desired. Similarly we obtain that $\Delta'_2(f(d)) = \hat{S}'$ and $\Delta_2(d) = \hat{S}$, where $\hat{S}' = S' \setminus \{(q, \top_g) : (q, \top_g) \in S'\}$. Since $S' \subseteq S$, we have $\hat{S}' \subseteq \hat{S}$, and hence $\Delta'_2(f(d)) \subseteq \Delta_2(d)$.

- (e) Suppose we have $\theta'_1 \preceq \theta_1 \xrightarrow{\text{store}} \theta_2$. This is treated like for $\xrightarrow{\text{guess}}$.
- (f) Suppose we have $\theta'_1 \preceq \theta_1 \xrightarrow{\text{keep}} \theta_2$. This is treated like for $\xrightarrow{\text{guess}}$.
- (g) Suppose we have $\theta'_1 \preceq \theta_1 \xrightarrow{\text{accept}} \theta_2$. This is treated like for $\xrightarrow{\text{guess}}$.
- (h) Suppose we have $\theta'_1 \preceq_f \theta_1 \xrightarrow{\delta} \theta_2$. Then, there is $(q, \perp, d) \in \Delta_1$ and some $\tau \in \delta$ so that $\Gamma_2 = \Gamma_1$, $r_2 = r_1$, $m_2 = m_1$, and $\Delta_2 = (\Delta_1 \setminus \{(q, \perp, d)\}) \cup \{(q, \alpha_i, d) : i \leq |\tau|\}$, where $(\alpha_i)_{i \leq |\tau|}$ are all the tests and actions occurring in τ . Applying Lemma 4.13 to d , we can assume that $f^{-1}(f(d)) = \{d\}$.

- If $(q, \perp, f(d)) \notin \Delta'_1$, it is immediate that $\theta'_1 \preceq_f \theta_2$.
- If $(q, \perp, f(d)) \in \Delta'_1$, let θ'_2 the result of a similar transition $\xrightarrow{\delta}$ triggered by τ on $(q, \perp, f(d))$. That is, $\Gamma'_2 = \Gamma'_1$, $r'_2 = r'_1$, $m'_2 = m'_1$, and $\Delta'_2 = (\Delta'_1 \setminus \{(q, \perp, f(d))\}) \cup \{(q, \alpha_i, f(d)) : i \leq |\tau|\}$. Hence, $\theta'_1 \xrightarrow{\delta} \theta'_2$. We show that $\theta'_2 \preceq_f \theta_2$. First note that $profile(\theta'_2) = profile(\theta'_1)$ and $profile(\theta_2) = profile(\theta_1)$ by Remark 4.8, and that $profile(\theta'_1) = profile(\theta_1)$ by $\theta'_1 \preceq \theta_1$; thus, $profile(\theta'_2) = profile(\theta_2)$. On the other hand, since $\Gamma'_2 = \Gamma'_1$ and $\Gamma_2 = \Gamma_1$ and $\Gamma_1 = \Gamma'_1$, we are only left with checking that for every e , $\Delta'_2(f(e)) \subseteq \Delta'_1(e)$.
 - For every e so that $f(e) \neq f(d)$ this is true since $\Delta'_2(f(e)) = \Delta'_1(f(e)) \subseteq \Delta_1(e) = \Delta_2(e)$.
 - For d we have $\Delta'_2(f(d)) = (\Delta'_1(f(d)) \setminus \{(q, \perp)\}) \cup \{(q, \alpha_i) : i \leq |\tau|\} \subseteq (\Delta_1(d) \setminus \{(q, \perp)\}) \cup \{(q, \alpha_i) : i \leq |\tau|\} = \Delta_2(d)$.
 - Finally, for any e so that $f(e) = f(d)$ we have that $e = d$ since $f^{-1}(f(d)) = \{d\}$, and the previous item applies.

- (i) Suppose we have $\theta'_1 \preceq_f \theta_1 \xrightarrow{\langle \mu \rangle^=} \theta_2$.

By definition of $\xrightarrow{\langle \mu \rangle^=}$ there is $(q, \langle \mu \rangle^=, d) \in \Delta_1$ and θ_2 is defined as $a_2 = a_1$, $d_2 = d_1$, $\Gamma_2 = \Gamma_1$ and $\Delta_2 = \Delta_1 \setminus \{(q, \langle \mu \rangle^=, d)\}$.

- Assume first that $(q, \langle \mu \rangle^=, f(d)) \notin \Delta'_1$. We show that $\theta'_1 \preceq_f \theta_2$. By construction f satisfies the surjectivity condition. Moreover the profiles remain untouched during the transition hence $profile(\theta'_1) = profile(\theta_2)$ and for all $e \in \mathbb{D}$, $\Gamma'_1(f(e)) = \Gamma_2(e)$. It remains to show that for all $e \in \mathbb{D}$, $\Delta'_1(f(e)) \subseteq \Delta_2(e)$. For $e \neq d$ this is immediate as $\Delta_1(f(e)) = \Delta_2(f(e))$. For $e = d$ we know by hypothesis that $\Delta'_1(f(d))$ does not contain $(q, \langle \mu \rangle^=)$ which is the only one affected by the transition. The result follows.
- Assume now that $(q, \langle \mu \rangle^=, f(d)) \in \Delta'_1$. As $\Gamma'_1(f(d)) = \Gamma_1(d)$, $\mu \in \Gamma'_1(f(d))$ and we can apply a transition $\xrightarrow{\langle \mu \rangle^=}$ to θ'_1 using this thread. Let θ'_2 be the resulting extended configuration. Applying Lemma 4.13 to d , we can assume that $f^{-1}(f(d)) = \{d\}$.

We show that $\theta'_2 \preceq_f \theta_2$. Notice that f does satisfies the surjectivity condition. It remains to show that $profile(\theta'_2) = profile(\theta_2)$ and for all $e \in \mathbb{D}$ we have $\Delta'_2(f(e)) \subseteq \Delta_2(e)$ and $\Gamma'_2(f(e)) = \Gamma_2(e)$.

(i) $\Delta'_2(f(e)) \subseteq \Delta_2(e)$ and $\Gamma'_2(f(e)) = \Gamma_2(e)$. Let $(S, \chi) = [\theta_2](e)$.

If $e \neq d$ then it follows from the definition of $\xrightarrow{\langle \mu \rangle^-}$ that $[\theta_2](e) = [\theta_1](e)$ and $[\theta'_2](f(e)) = [\theta'_1](f(e))$ (since $f(e) \neq f(d)$ by $f^{-1}(f(d)) = \{d\}$). From $\theta'_1 \preceq_f \theta_1$ it follows that $[\theta'_1](f(e)) = (S', \chi)$ with $S' \subseteq S$. The result follows.

In the case where $e = d$, $[\theta_2](e) = [\theta'_2](f(e)) = (S, \chi)$.

(ii) $profile(\theta_2) = profile(\theta'_2)$. This is obvious as by Remark 4.8, the transition does not affect profiles.

(j) The cases $\xrightarrow{\langle \mu \rangle^\neq}$, \xrightarrow{root} , \xrightarrow{q} and their negative counterparts are treated similarly.

We only need to verify that whenever $\theta'_1 \preceq \theta_1$, if θ_1 satisfies the condition for any transition in $\{\xrightarrow{\langle \mu \rangle^\neq}, \xrightarrow{root}, \xrightarrow{q}\}$ or their negative counterparts, then θ'_1 also satisfies this condition. For all these cases this is a simple consequence of θ_1 and θ'_1 having the same profile.

In the case $\overline{\langle \mu \rangle^-}$, if $\mu \notin \Gamma_1(d)$, then the same hold in θ'_1 for $f(d)$.

For the case $\langle \mu \rangle^\neq$, if $\mu \in \Gamma_1(d')$ for some $d' \neq d$, then by equality of profiles there is a data value e such that $[\theta'_1](e) = [\theta_1](d')$. Notice that even in the case when $[\theta_1](e) = [\theta_1](d)$, the definition of $profile$ and the fact that $d \neq d'$ guarantees that we can always choose $e \neq d$. Hence $\mu \in \Gamma_1(e)$ and $e \neq d$ as required.

For the case $\overline{\langle \mu \rangle^\neq}$, if $\mu \notin \Gamma_1(d')$ for all $d' \neq d$, $\Gamma_1(d)$ is the unique one that may contain μ . By equality of the profile, this must also be the case for $\Gamma'_1(d)$ as desired.

The cases \xrightarrow{root} and $\xrightarrow{\overline{root}}$ are straightforward as the value of r is preserved by \preceq .

The cases \xrightarrow{q} and $\xrightarrow{\overline{q}}$ are also immediate. \square

We now do the same for \xrightarrow{merge} . In this case n depends on the size of the extended configurations and we may need to perform a permutation on the data values. In this case, the use of \xrightarrow{inc} becomes essential. Indeed, while it is not true that a transition $\theta_1, \theta_2 \xrightarrow{merge} \theta_0$ can be downward-simulated with a transition \xrightarrow{merge} , it can be simulated by a sequence of transitions $(\xrightarrow{inc})^* \xrightarrow{merge}$ as we show next, which is the reason why we needed to introduce \xrightarrow{inc} in our transition system. More precisely, while for $\theta'_1 \preceq \theta_1$, $\theta'_2 \preceq \theta_2$ we have that $\theta_1, \theta_2 \xrightarrow{merge} \theta_0$ may not be simulated from applying a merge transition to θ'_1, θ'_2 , it still *could* be simulated from some “fatter” versions of θ'_1, θ'_2 (i.e., from the result of applying a number of \xrightarrow{inc} transitions).

Lemma 4.15. *Let $\theta_1, \theta_2, \theta_0$ be extended configurations such that $\theta_1, \theta_2 \xrightarrow{merge} \theta_0$. If $\theta'_1 \preceq \theta_1$ and $\theta'_2 \preceq \theta_2$, then there are $\hat{\theta}_1 \sim \theta'_1$, $\hat{\theta}'_2 \sim \theta'_2$ and θ'_0 such that $\theta'_0 \preceq \theta_0$ and $\{\hat{\theta}'_1, \hat{\theta}'_2\} \Rightarrow^n \theta'_0$ for some n computable from θ'_1 and θ'_2 .*

Proof. Throughout the proof we use the following notation for $i \in \{0, 1, 2\}$: $\theta_i = (\Delta_i, \Gamma_i, r_i, m_i)$, similarly for θ'_i . Moreover a_i and d_i denote the label and data value associated to θ_i (similarly for θ'_i). By definition of \xrightarrow{merge} we have $a_1 = a_2$ and $d_1 = d_2$. Suppose $\theta'_1 \preceq_{f_1} \theta_1$ and $\theta'_2 \preceq_{f_2} \theta_2$. In particular (recall Remark 4.3) we have $a_1 = a'_1 = a_2 = a'_2$, $f_1(d_1) = d'_1$ and $f_2(d_2) = d'_2$.

By definition of \xrightarrow{merge} , θ_0 is constructed by taking the union of θ_1 and θ_2 . Hence, for all d we have $[\theta_0](d) = [\theta_1](d) \cup [\theta_2](d)$.

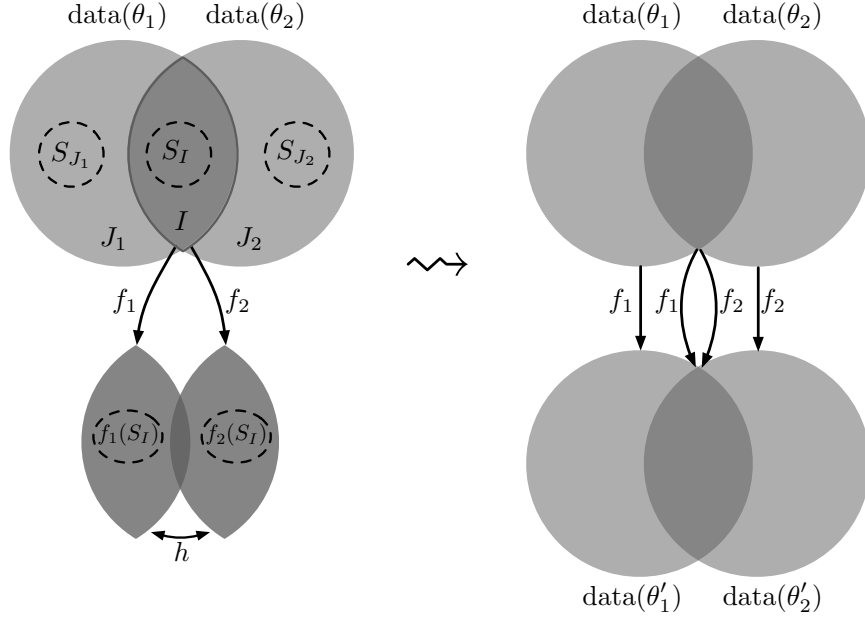


Figure 6: Argument of Lemma 4.15.

We need to merge θ'_1 and θ'_2 in order to reflect the way the union of θ_1 and θ_2 is done. For instance if d is a data value that occurs both in $data(\theta_1)$ and $data(\theta_2)$ then we would like to identify $f_1(d)$ with $f_2(d)$. We also need to keep track of the number of data values having the same type, up to 2. This is essentially what we do below.

Let $I = data(\theta_1) \cap data(\theta_2)$, $J_1 = data(\theta_1) \setminus I$, $J_2 = data(\theta_2) \setminus I$. Figure 6 contains a depiction of the main argument that will be described in the next paragraphs.

We first define sets $S_1 = S_I \cup S_{J_1}$ and $S_2 = S_I \cup S_{J_2}$ of *special* data values. S_I is constructed as follows: For each pair $\chi_1, \chi_2 \subseteq \mathcal{S}$ we put in S_I two (if possible) elements $d \in I$ such that $\Gamma_1(d) = \chi_1$ and $\Gamma_2(d) = \chi_2$. If only one such data value exists we put in S_I only this one. Note that S_I contains d_1 and hence d_2 . Similarly, S_{J_1} is constructed as follows: For each $\chi_1 \subseteq \mathcal{S}$ we add to S_{J_1} two (if possible) elements $d \in J_1$ such that $\Gamma_1(d) = \chi_1$. If only one such data value exists we add only this one. S_{J_2} is defined analogously. Note that the size of S_1 and S_2 are bounded by $2 \cdot 2^{2 \cdot |S|}$.

We then apply Lemma 4.11 on $\theta'_1 \preceq \theta_1$ with S_1 and on $\theta'_2 \preceq \theta_2$ with S_2 and further assume that f_1 and f_2 are injective on S_1 and S_2 .

Next we establish a bijection between $f_1(I)$ and $f_2(I)$. Let h be the relation $\{(f_1(d), f_2(d)) \mid d \in I\}$. Note that if we have the property $f_1(d) = f_1(d')$ iff $f_2(d) = f_2(d')$ for all $d \in I$, then h is a bijection: it is an injective function due to the property, and it is surjective since we define it from $f_1(I)$ to $f_2(I)$. Although this property may not necessarily hold, we show next that in this case we can apply a number of $\xrightarrow{\text{inc}}$ transitions to θ'_1 and θ'_2 in order to obtain extended configurations in which the property holds, and thus that we can safely assume that the property holds for the remaining of the proof. Indeed, suppose the property does not hold: for some d, d' we have $f_1(d) = f_1(d')$ but $f_2(d) \neq f_2(d')$ (the other case of the failing of the property is, of course, symmetrical), and let $(S, \chi) = [\theta'_1](f_1(d))$. Note that

we have

$$\chi = \Gamma_1(d) = \Gamma'_1(f_1(d)) = \Gamma'_1(f_1(d')) = \Gamma_1(d').$$

Since $\text{profile}(\theta_1) = \text{profile}(\theta'_1)$, θ'_1 must contain another data value $e \neq f_1(d)$ such that $\Gamma'_1(e) = \chi$. Hence we are enabled to apply $\xrightarrow{\text{inc}(S, \chi)}$ to θ'_1 and add a fresh copy \hat{d} of $f_1(d)$. The resulting extended configuration $\hat{\theta}'_1$ is still smaller than θ_1 by means of a function \hat{f}_1 identical to f_1 except that $\hat{f}_1(d) = \hat{d} \neq f_1(d') = \hat{f}_1(d')$. Note that now there is one less pair $d, d' \in I$ contradicting $\hat{f}_1(d) = \hat{f}_1(d')$ iff $f_2(d) = f_2(d')$, for $\hat{\theta}'_1 \preceq_{\hat{f}_1} \theta_1$ and $\theta'_2 \preceq_{f_2} \theta_2$. Thus, we can apply $\xrightarrow{\text{inc}}$ to θ'_1 and θ'_2 a number of times —bounded by $|\text{data}(\theta'_1)| + |\text{data}(\theta'_2)|$ — in order to obtain extended configurations $\hat{\theta}'_1 \preceq_{\hat{f}_1} \theta_1$ and $\hat{\theta}'_2 \preceq_{f_2} \theta_2$ so that $\{(\hat{f}_1(d), \hat{f}_2(d)) \mid d \in I\}$ is a bijection. For the rest of the proof, and in order to avoid introducing more symbols, we are simply going to assume that h is a bijection from $f_1(I)$ to $f_2(I)$ without loss of generality.

We extend h to a bijection on \mathbb{D} making sure that $h(d) \notin S_{J_2}$ for all $d \in S_{J_1}$, and $h(d) \notin S_{J_1}$ for all $d \in S_{J_2}$. The fact that we can extend h follows easily from \mathbb{D} being infinite, as we show next. For $i = 1, 2$, let $D_i \subseteq \mathbb{D} \setminus (S_{J_1} \cup S_{J_2} \cup f_1(I) \cup f_2(I))$ be so that $|D_i| = |S_{J_i}|$ so that $D_1 \cap D_2 = \emptyset$, and let h_i be a bijection between S_{J_i} and D_i . Now take any bijection h' between $\mathbb{D} \setminus (f_1(I) \cup S_{J_1} \cup S_{J_2})$ and $\mathbb{D} \setminus (f_2(I) \cup D_1 \cup D_2)$ —it exists since these sets have the same cardinality \aleph_0 — and define the extension as $h \cup h_1 \cup h_2 \cup h'$.

Finally, consider the extended configuration θ''_1 resulting from replacing every data value $d \in \text{data}(\theta'_1)$ with $h(d)$ in θ'_1 . The intuition is that in this way we make equal the data values of $f_1(I)$ with those of $f_2(I)$ without making equal any other values between θ'_1 and θ'_2 . Let $f'_1 = f_1 \circ h$; we now have that $\theta''_1 \preceq_{f'_1} \theta_1$.

θ''_1 and f'_1 satisfy the same hypothesis as the previous θ'_1 and f_1 but now $f'_1(d) = f_2(d)$ for all $d \in I$ (in particular $d''_1 = d'_2$) and furthermore $f'_1(I) = \text{data}(\theta''_1) \cap \text{data}(\theta'_2) = f_2(I)$. Note that we are implicitly using that transitions (in particular $\xrightarrow{\text{inc}}$) are closed under data bijections; that is, $\theta \xrightarrow{\text{inc}} \theta'$ iff $f(\theta) \xrightarrow{\text{inc}} f(\theta')$ for any data bijection f .

Let θ'_0 be the extended configuration obtained by applying $\xrightarrow{\text{merge}}$ to θ''_1 and θ'_2 . We claim that $\theta'_0 \preceq \theta_0$. This is witnessed by the function $f(d) = f'_1(d)$ if $d \in \text{data}(\theta_1)$ and $f(d) = f_2(d)$ otherwise. Note that f satisfies the surjectivity condition. It remains to show that $\text{profile}(\theta'_0) = \text{profile}(\theta_0)$ and for all $e \in \mathbb{D}$ we have $\Delta'_0(f(e)) \subseteq \Delta_0(e)$ and $\Gamma'_0(f(e)) = \Gamma_0(e)$.

- (1) Let us first show $\Delta'_0(f(e)) \subseteq \Delta_0(e)$ and $\Gamma'_0(f(e)) = \Gamma_0(e)$.
 - (a) Assume $e \in I$. Then $\Delta'_0(f(e)) = \Delta''_1(f(e)) \cup \Delta'_2(f(e)) = \Delta''_1(f'_1(e)) \cup \Delta'_2(f_2(e)) \subseteq \Delta_1(e) \cup \Delta_2(e) \subseteq \Delta_0(e)$. Similarly, $\Gamma'_0(f(e)) = \Gamma''_1(f(e)) \cup \Gamma'_2(f(e)) = \Gamma''_1(f'_1(e)) \cup \Gamma'_2(f_2(e)) = \Gamma_1(e) \cup \Gamma_2(e) = \Gamma_0(e)$.
 - (b) If $e \in \text{data}(\theta_1) \setminus I$ we have: $\Delta'_0(f(e)) = \Delta''_1(f(e)) \cup \Delta'_2(f(e)) = \Delta''_1(f'_1(e)) \subseteq \Delta_1(e) \subseteq \Delta_0(e)$. Similarly, $\Gamma'_0(f(e)) = \Gamma''_1(f(e)) \cup \Gamma'_2(f(e)) = \Gamma''_1(f'_1(e)) = \Gamma_1(e) = \Gamma_0(e)$.
- (2) Let us now show $\text{profile}(\theta'_0) = \text{profile}(\theta_0)$. This is an immediate consequence of the injectivity of f'_1 and f_2 on the special values. \square

Proof of Proposition 4.10. It follows directly from Lemma 4.14 and Lemma 4.15. \square

Corollary 4.16. *Given a BUDA \mathcal{A} , it is decidable whether an accepting extended configuration is derivable from Θ_I^\sim in the transition system associated with \mathcal{A} .*

Proof. Let f be the function computing the maximum value of n as specified in Proposition 4.10. Consider the following procedure.

- (1) Initialize Θ to Θ_I .
- (2) While there is some θ so that $\Theta \not\leq_{\min} \{\theta\}$ and $\Theta^\sim \Rightarrow^n \theta$ for $n \leq f(\Theta)$: add θ to Θ .
- (3) Accept if there is an accepting extended configuration in Θ , otherwise reject.

Because \leq is a wqo, the procedure always terminates. It should also be noted that the second step is computable. Indeed there are only finitely many elements within Θ^\sim to consider, those derived from Θ via a permutation of the data values $data(\Theta)$, as the other ones would derive an extended configuration equivalent to one already derived. This means that for example if $n = 1$, it suffices to consider all distinct θ modulo bijection of data values so that $\Theta \not\leq_{\min} \{\theta\}$ and

- $\theta_1 \rightarrow \theta$ for some $\theta_1 \in \Theta$ for any transition except $\xrightarrow{\text{merge}}$, or
- $\theta'_1, \theta'_2 \xrightarrow{\text{merge}} \theta$ for some $\theta'_1 \sim \theta_1$, $\theta'_2 \sim \theta_2$, and $\theta_1, \theta_2 \in \Theta$. However, in this case we only need to consider all pairs (θ'_1, θ'_2) that are images of (θ_1, θ_2) via a bijection of $data(\theta_1) \cup data(\theta_2)$, and there are only finitely many such bijections.

Note that due to transitions $\xrightarrow{\text{guess}}$ and $\xrightarrow{\text{grow}}$ there may be infinitely many such θ , but only finitely many modulo \sim .

Let Θ be the final set after the evaluation of the algorithm. First, note that all the extended configurations θ from Θ are so that $\Theta_I^\sim \Rightarrow^+ \theta$. Therefore, if the algorithm accepts, there is an accepting extended configuration derivable from Θ_I^\sim . We therefore show the converse.

Suppose that $\Theta_I^\sim \Rightarrow^t \theta_t$ and let $\theta_1, \dots, \theta_t$ be the extended configurations derived at each step. We show by induction that for every i we have $\Theta \leq_{\min} \Theta_I \cup \{\theta_1, \dots, \theta_i\}$. The base case when $i = 0$ is trivial since $\Theta_I \subseteq \Theta$. For the inductive case, suppose that $\Theta \leq_{\min} \Theta_I \cup \{\theta_1, \dots, \theta_i\}$. By Proposition 4.10, since $\Theta_I \cup \{\theta_1, \dots, \theta_i\} \Rightarrow \theta_{i+1}$ there is some $n \leq f(\Theta)$ so that $\Theta^\sim \Rightarrow^n \theta$ with $\theta \preceq \theta_{i+1}$. By the condition of the algorithm, it must be so that $\Theta \leq_{\min} \{\theta\}$ and hence $\Theta \leq_{\min} \{\theta_{i+1}\}$. Therefore, $\Theta \leq_{\min} \Theta_I \cup \{\theta_1, \dots, \theta_{i+1}\}$.

As a consequence of this property, if there is an accepting configuration θ_F so that $\Theta_I^\sim \Rightarrow^+ \theta_F$, then in particular we have $\Theta \leq_{\min} \{\theta_F\}$ and hence there must be some $\theta \in \Theta$ so that $\theta \preceq \theta_F$. Since θ_F is accepting, and the set of accepting extended configurations is downward closed (Lemma 4.7), it follows that θ is accepting. \square

As shown next, this implies the decidability for the emptiness problem for BUDA.

4.5. From BUDA to its extended configurations. The transition system $\mathcal{W}_{\mathcal{A}}$ associated to a BUDA \mathcal{A} is then defined as follows. Its elements are the extended configurations of \mathcal{A} as defined in Section 4.1. Its transition relation is as defined in Section 4.3. As shown in Section 4.4 the wqo defined in Section 4.2 is compatible with the transition system. Hence coverability of $\mathcal{W}_{\mathcal{A}}$ is decidable. It remains to show that $\mathcal{W}_{\mathcal{A}}$ has the desired behavior, i.e. that its coverability problem is equivalent to the emptiness problem of \mathcal{A} . This is what we do in this section.

One direction is easy as the transition system can easily simulate \mathcal{A} . The other direction requires more care. As evidenced in (\dagger) , $\mathcal{W}_{\mathcal{A}}$ may perform a $\xrightarrow{\text{inc}(S, \chi)}$ transition anytime. Remember that the effect of $\xrightarrow{\text{inc}(S, \chi)}$ can be seen as a result of the tree growing in width. We will see that this can be simulated by a BUDA only when it moves up in the tree. This issue is solved by showing that all transitions except $\xrightarrow{\text{grow}}$ commute with $\xrightarrow{\text{inc}(S, \chi)}$ hence all

transitions $\xrightarrow{\text{inc}(S, \chi)}$ can be grouped just before a $\xrightarrow{\text{grow}}$ and combined with it in order to form an up-transition of a BUDA. As a consequence, we obtain the following.

Proposition 4.17. *Let \mathcal{A} be a BUDA. Let $\mathcal{W}_{\mathcal{A}}$ be the transition system associated with \mathcal{A} . Then \mathcal{A} has an accepting run if, and only if, $\mathcal{W}_{\mathcal{A}}$ can reach an accepting extended configuration from the set of initial extended configurations.*

In the sequel, we say that $\Gamma \subseteq (\mathcal{S} \times \mathbb{D})$ is *consistent* with a data tree $\mathbf{t} = \mathbf{a} \otimes \mathbf{b} \otimes \mathbf{d} \in \text{Trees}(\mathbb{A} \times \mathbb{B} \times \mathbb{D})$ when for every possible (μ, d) , Γ contains (μ, d) iff there is a downward path in \mathbf{t} that starts at the root and ends at some position x such that $\mathbf{d}(x) = d$ and evaluates to μ via h . In particular this implies that the label and data value of Γ are the label and data value of the root of \mathbf{t} .

We first show that the transition system associated with a BUDA at least simulates its behavior.

Lemma 4.18. *Consider an $\mathcal{A} \in \text{BUDA}$ and its associated transition system $\mathcal{W}_{\mathcal{A}}$. If \mathcal{A} has an accepting run then $\mathcal{W}_{\mathcal{A}}$ can reach an accepting extended configuration from its initial extended configuration.*

Proof. We show that from every accepting run ρ of \mathcal{A} on $\mathbf{t} = \mathbf{a} \otimes \mathbf{b} \otimes \mathbf{d}$ there exists a finite sequence of transitions in $\mathcal{W}_{\mathcal{A}}$ starting in Θ_{Γ}^{\sim} and ending in an accepting configuration.

Given a position $x \in \text{pos}(\mathbf{t})$, we define the extended configuration of $\mathbf{t}|_x$ as $\theta_x = (\Delta_x, \Gamma_x, r_x, m_x)$, where $r_x = \text{false}$ (unless x is the root position), $m_x = \text{false}$, $(\mathbf{a} \otimes \mathbf{b})(x)$ is the label of θ_x , $\mathbf{d}(x)$ is the data value of θ_x , Γ_x is consistent with $\mathbf{t}|_x$, and $\Delta_x = \{(q, \perp, d) \mid (q, d) \in \rho(x)\}$.

We show by bottom-up induction on x that θ_x is reachable from Θ_{Γ}^{\sim} . By definition of Θ_{Γ}^{\sim} , for any leaf node x of \mathbf{t} we do have $\theta_x \in \Theta_{\Gamma}^{\sim}$ and hence our inductive process can start.

Let now x be a node of \mathbf{t} and let $x \cdot 1, \dots, x \cdot n$ be its children. By induction $\theta_{x \cdot 1}, \dots, \theta_{x \cdot n}$ are reachable from Θ_{Γ}^{\sim} . For each i , and thread (q, \perp, d) in $\theta_{x \cdot i}$ there exists a transition $\tau_{i, q, d} \in \delta$ witnessing the fact that ρ is a valid run. We derive θ_x from there as follows:

- (1) for each i , and each thread (q, \perp, d) let θ_i be the extended configuration obtained from $\theta_{x \cdot i}$ using a transition $\xrightarrow{\delta}$ based on $\tau_{i, q, d}$,
- (2) for each i , we derive the extended configuration θ'_i from θ_i using the appropriate transitions for each thread newly introduced at the previous step. Note that because ρ is accepting, each operation is successful and each new thread has \top or \top_g as second component. Hence,
- (3) for each i we can apply $\xrightarrow{\text{grow}}$ to θ'_i and derive the extended configuration θ''_i using $\mathbf{d}(x)$ and $(\mathbf{a} \otimes \mathbf{b})(x)$ as the data value and label of the new extended configuration,
- (4) we then make a sequence of $(n - 1)$ applications of $\xrightarrow{\text{merge}}$ adding one by one θ''_i to the previously derived extended configuration,

Repeating this simulation we finally derive the configuration θ_{ε} of the root of \mathbf{t} . Since ρ is accepting, for every $(q, d) \in \rho(\varepsilon)$ there is $\tau = (t, \text{accept}) \in \delta$ so that $\mathbf{t}, \varepsilon, (q, d) \models t$. This means that for every $(q, \perp, d) \in \Delta_{\varepsilon}$ one can apply a $\xrightarrow{\delta}$ transition based on τ arriving to an extended configuration $\theta'_{\varepsilon} = (\Delta, \Gamma, r, m)$ with $r = \text{true}$ and $\Delta = \emptyset$ which is hence accepting. \square

The other direction requires more care. First, we need to prove the following lemma.

Lemma 4.19. *If we have $\theta_1 \rightarrow_{\varepsilon} \theta_2 \xrightarrow{\text{inc}(S, \chi)} \theta_3$ then either*

- $\hat{\theta}_1 \xrightarrow{\text{inc}(S', \chi)} \theta' \rightarrow_\epsilon \theta_3$, or
- $\hat{\theta}_1 \xrightarrow{\text{inc}(S', \chi)} \theta' \rightarrow_\epsilon \theta'' \rightarrow_\epsilon \theta_3$

for some extended configuration $\hat{\theta}_1, \theta', \theta''$ and set $S' \subseteq Q$ such that $\hat{\theta}_1 \sim \theta_1$.

Proof. Suppose that $\theta_1 \rightarrow_\epsilon \theta_2 \xrightarrow{\text{inc}(S, \chi)} \theta_3$. Notice that $\Gamma_1 = \Gamma_2$ by definition of \rightarrow_ϵ , and that $|\lceil \theta_2 \rceil^{-1}(S, \chi)| \geq 1$ by definition of $\xrightarrow{\text{inc}(S, \chi)}$. Let e be a data value so that $\lceil \theta_2 \rceil(e) = (S, \chi)$, and let $e' \in \text{data}(\theta_3) \setminus \text{data}(\theta_2)$ be the new data value added as a result of $\theta_2 \xrightarrow{\text{inc}(S, \chi)} \theta_3$. We thus have $\lceil \theta_3 \rceil(e') = (\hat{S}, \chi)$ with $\hat{S} = S \setminus \{(q, \top_g) : (q, \top_g) \in S\}$. Modulo replacing θ_1 by an equivalent extended configuration we can further assume without any loss of generality that $e' \notin \text{data}(\theta_1)$.

Let (q, α, d) be the thread of Δ_1 that triggers \rightarrow_ϵ . We will treat all cases of \rightarrow_ϵ at once, independently of which particular transition it is. Let $H \subseteq \Delta_2$ be the new threads generated from (q, α, d) by $\theta_1 \rightarrow_\epsilon \theta_2$ (note that H may be empty). We then have that:

$$\Delta_2 = (\Delta_1 \setminus \{(q, \alpha, d)\}) \cup H \quad (4.1)$$

$$\Delta_3 = \Delta_2 \cup (\hat{S} \times \{e'\}) \quad (4.2)$$

$$\Gamma_2 = \Gamma_1 \quad (4.3)$$

$$\Gamma_3 = \Gamma_2 \cup (\chi \times \{e'\}) \quad (4.4)$$

Let $(S', \chi) = \lceil \theta_1 \rceil(e)$ (recall that $\Gamma_1(e) = \Gamma_2(e) = \chi$). We show that $\xrightarrow{\text{inc}(S', \chi)}$ can be applied to θ_1 . In other words we show that $|\Gamma_1^{-1}(\chi)| \geq 2$. As $\xrightarrow{\text{inc}(S, \chi)}$ was applied to θ_2 we have that $|\Gamma_2^{-1}(\chi)| \geq 2$. Since $\Gamma_1 = \Gamma_2$, this implies $|\Gamma_1^{-1}(\chi)| \geq 2$ and we are done.

We then define $\theta' = (\Delta', \Gamma', r_1, m_1)$ such that $\theta_1 \xrightarrow{\text{inc}(S', \chi)} \theta'$ with

$$\Delta' = \Delta_1 \cup (\hat{S}' \times \{e'\}) \quad (4.5)$$

$$\Gamma' = \Gamma_1 \cup (\chi \times \{e'\}) \quad (4.6)$$

where $\hat{S}' = S' \setminus \{(q, \top_g) : (q, \top_g) \in S'\}$.

Notice that by (4.3), (4.4) and (4.6) we get $\Gamma' = \Gamma_3$.

We show that $\theta' \rightarrow_\epsilon \theta_3$ or $\theta' \rightarrow_\epsilon \theta'' \rightarrow_\epsilon \theta_3$ for some θ'' . We distinguish between two possibilities: either $\Delta_1(e) = \Delta_2(e)$ or not.

- The easiest case is when $\Delta_1(e) = \Delta_2(e)$ (in particular $S = S'$). This means that the two transitions of $\theta_1 \rightarrow_\epsilon \theta_2 \xrightarrow{\text{inc}(S, \chi)} \theta_3$ do not interact with one another. Since a transition $\xrightarrow{\text{inc}}$ preserves the truth of tests, the same transition as the one between θ_1 and θ_2 can be applied to θ' . We show that this yields θ_3 . As already mentioned, $\Gamma_3 = \Gamma'$.

Let H' be the new threads generated from (q, α, d) by applying this transition (i.e. the set of threads in the resulting extended configuration is $(\Delta' \setminus \{(q, \alpha, d)\}) \cup H'$). If the transition was a $\xrightarrow{\text{guess}}$ then we make sure that the guessed data value is the one that was guessed in θ_2 . It is therefore immediate to verify that $H = H'$ unless the transition was a $\xrightarrow{\text{univ}}$. In this latter case, assuming $\alpha = \text{univ}(p)$, we have $H' = H \cup \{(p, \top, e')\}$. But we also have $(p, \top, e) \in \Delta_2$ because $e \in \text{data}(\theta_1)$ and $\theta_1 \xrightarrow{\text{univ}} \theta_2$. As $\Delta_1(e) = \Delta_2(e)$ we have $(p, \top, e) \in \Delta_1$ and therefore $(p, \top, e') \in \Delta'$. In all cases we get $(\Delta' \setminus \{(q, \alpha, d)\}) \cup H' =$

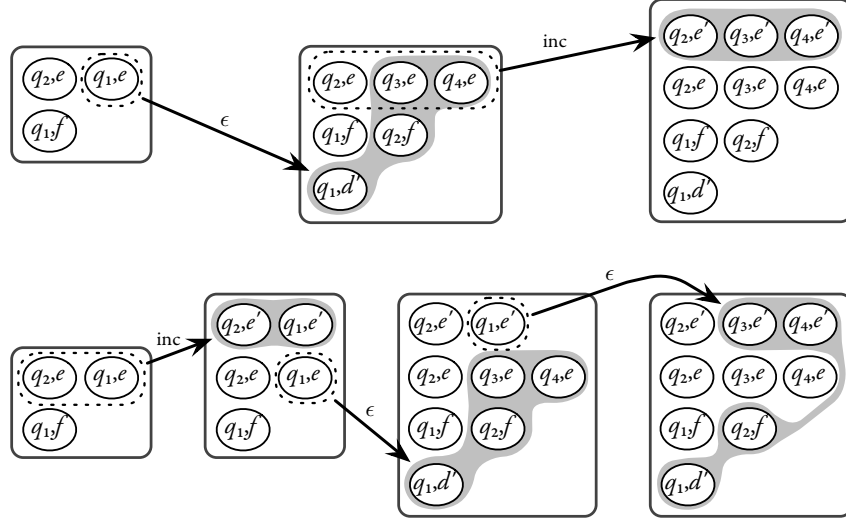


Figure 7: General idea of how to commute $\xrightarrow{\text{inc}(S, \chi)}$ with \rightarrow_ϵ . In the figure we use a general \rightarrow_ϵ which creates new threads with the same data value and with fresh data value.

$(\Delta' \setminus \{(q, \alpha, d)\}) \cup H$. Further:

$$\begin{aligned}
 (\Delta' \setminus \{(q, \alpha, d)\}) \cup H' &= (\Delta' \setminus \{(q, \alpha, d)\}) \cup H && \text{(by the previous remark)} \\
 &= ((\Delta_1 \cup (\hat{S}' \times \{e'\})) \setminus \{(q, \alpha, d)\}) \cup H && \text{(by (4.5))} \\
 &= (\Delta_1 \setminus \{(q, \alpha, d)\}) \cup H \cup (\hat{S}' \times \{e'\}) && \text{(since } e' \neq d\text{)} \\
 &= \Delta_2 \cup (\hat{S}' \times \{e'\}) && \text{(by (4.1))} \\
 &= \Delta_2 \cup (\hat{S} \times \{e'\}) && \text{(since } S = S'\text{)} \\
 &= \Delta_3. && \text{(by (4.2))}
 \end{aligned}$$

Hence, we have that $\theta' \rightarrow_\epsilon \theta_3$.

- If $\Delta_1(e) \neq \Delta_2(e)$, we distinguish again between two possibilities depending on whether $d = e$ or not.
 - Assume first that $d = e$. We are in a situation as the one depicted in Figure 7. In this case we simply apply twice the same transition to θ' , the first time using the thread (q, α, e) and the second time using (q, α, e') (in the case of $\xrightarrow{\text{guess}}$ we guess twice the same data value). The resulting extended configuration is θ_3 .
 - The remaining situation is when $d \neq e$ and $\Delta_1(e) \neq \Delta_2(e)$. It could come from a transition $\xrightarrow{\text{univ}}$, $\xrightarrow{\text{guess}}$ or $\xrightarrow{\text{store}}$ (the other transitions only affect the data value d and therefore $\Delta_1(e) = \Delta_2(e)$). Suppose first that $\alpha = \text{guess}(p)$, and that it produces the thread (p, \top_g, e) as a result. We show that $\theta' \xrightarrow{\text{guess}} \theta_3$.

Notice that since e is with a thread (p, \top_g) , we have that $\hat{S}' = \hat{S}$, and we can apply the previous reasoning. Indeed, by definition of $\xrightarrow{\text{guess}}$ we have that the resulting Δ is

$$\begin{aligned}
& (\Delta' \setminus \{(q, \text{guess}(p), d)\}) \cup \{(p, \top_g, e)\} \\
&= ((\Delta_1 \cup (\hat{S}' \times \{e'\})) \setminus \{(q, \text{guess}(p), d)\}) \cup \{(p, \top_g, e)\} && \text{(by (4.5))} \\
&= (\Delta_1 \setminus \{(q, \text{guess}(p), d)\}) \cup \{(p, \top_g, e)\} \cup (\hat{S}' \times \{e'\}) && \text{(since } e' \neq d) \\
&= \Delta_2 \cup (\hat{S}' \times \{e'\}) && \text{(by (4.1))} \\
&= \Delta_2 \cup (\hat{S} \times \{e'\}) && \text{(since } \hat{S}' = \hat{S}) \\
&= \Delta_3. && \text{(by (4.2))}
\end{aligned}$$

The fact that the resulting Γ is Γ_3 is immediate. Hence, $\theta' \xrightarrow{\text{guess}} \theta_3$.

Suppose now that $\alpha = \text{univ}(p)$. We show that $\theta' \xrightarrow{\text{univ}} \theta_3$.

By definition of $\xrightarrow{\text{univ}}$ we have that $S = S' \cup \{(p, \top)\}$. Furthermore we have:

$$\begin{aligned}
& (\Delta' \setminus \{(q, \text{univ}(p), d)\}) \cup \{(p, \top, d') : d' \in \text{data}(\Gamma')\} \\
&= ((\Delta_1 \cup (\hat{S}' \times \{e'\})) \setminus \{(q, \text{univ}(p), d)\}) \cup \{(p, \top, d') : d' \in \text{data}(\Gamma')\} && \text{(by (4.5))} \\
&= (\Delta_1 \setminus \{(q, \text{univ}(p), d)\}) \cup \{(p, \top, d') : d' \in \text{data}(\Gamma')\} \cup (\hat{S}' \times \{e'\}) && \text{(since } e' \neq d) \\
&= (\Delta_1 \setminus \{(q, \text{univ}(p), d)\}) \cup \{(p, \top, d') : d' \in \text{data}(\Gamma_1)\} \cup \{(p, \top, e')\} \cup (\hat{S}' \times \{e'\}) && \text{(since } e' \neq d) \\
&= \Delta_2 \cup \{(p, \top, e')\} \cup (\hat{S}' \times \{e'\}) && \text{(by definition of } \xrightarrow{\text{univ}}) \\
&= \Delta_2 \cup (\hat{S} \times \{e'\}) && \text{(since } S = S' \cup \{(p, \top)\}) \\
&= \Delta_3. && \text{(by (4.2))}
\end{aligned}$$

Hence, $\theta' \xrightarrow{\text{univ}} \theta_3$.

The case of $\xrightarrow{\text{store}}$ is treated similarly. \square

Finally we show:

Lemma 4.20. *Consider a BUDA \mathcal{A} and its associated transition system $\mathcal{W}_{\mathcal{A}}$. If $\mathcal{W}_{\mathcal{A}}$ can reach an accepting extended configuration from the set of initial extended configurations then \mathcal{A} has an accepting run.*

Proof. An extended configuration $\theta = (\Delta, \Gamma, r, m)$ is called *starting* if for all (q, α, d) in Δ we have $\alpha = \perp$. Note that by definition of the transition relations, a starting extended configuration can only be obtained from a non starting extended configuration via a $\xrightarrow{\text{grow}}$ transition. Moreover starting extended configurations are preserved only by $\xrightarrow{\text{merge}}$ and $\xrightarrow{\text{inc}}$ transitions.

We show by induction on the length of the derivation that for every starting reachable extended configuration θ (by reachable we mean such that $\Theta_{\Gamma}^{\sim} \Rightarrow^+ \theta$), there exists a tree \mathbf{t}_{θ} and a run ρ_{θ} of \mathcal{A} on \mathbf{t}_{θ} , such that Γ is consistent with \mathbf{t}_{θ} and $\rho(x)$ is Δ for the root x of \mathbf{t}_{θ} .

This is clearly the case for all extended configurations in Θ_{Γ}^{\sim} .

For the inductive argument it is useful to notice that if a starting extended configuration θ can be associated with a tree \mathbf{t}_{θ} and a run ρ_{θ} of \mathcal{A} on \mathbf{t}_{θ} satisfying the inductive hypothesis

then, for any bijection h on the data values, $h(\theta')$, $h(\mathbf{t}_\theta)$ and $h(\rho_\theta)$ satisfy also the inductive hypothesis.

Assume θ_1 and θ_2 are both starting reachable extended configurations and that $\theta_1, \theta_2 \xrightarrow{\text{merge}} \theta_0$. By induction we have trees \mathbf{t}_{θ_1} and \mathbf{t}_{θ_2} and runs ρ_{θ_1} and ρ_{θ_2} satisfying the induction hypothesis. By definition of $\xrightarrow{\text{merge}}$, θ_1 and θ_2 have the same label and data value. Hence by consistency of Γ_1 and Γ_2 , the roots of \mathbf{t}_{θ_1} and \mathbf{t}_{θ_2} have the same label and data value. Let \mathbf{t} be the tree constructed from the union of \mathbf{t}_{θ_1} and \mathbf{t}_{θ_2} by identifying their roots. We show that \mathbf{t} is the desired \mathbf{t}_{θ_0} . The reader can easily verify that ρ , constructed by taking the union of ρ_{θ_1} and ρ_{θ_2} , is a run of \mathcal{A} on \mathbf{t} and that Γ_0 is consistent with \mathbf{t} .

Assume now that θ is a starting reachable extended configuration and that $\theta \xrightarrow{\text{inc}(S, \chi)} \theta'$. Let e be the data value such that $(S, \chi) = [\theta](e)$ and let e' be the new data value added in θ' . By induction we have a tree \mathbf{t}_θ and a run ρ_θ satisfying the induction hypothesis. Notice that by definition of $\xrightarrow{\text{inc}(S, \chi)}$ the data value e duplicated cannot be the one of the root of \mathbf{t}_θ (because the associated χ' occurs only once). Let \mathbf{t}'_θ be the tree obtained from \mathbf{t}_θ by replacing e with e' . Let \mathbf{t} be the tree constructed from the union of \mathbf{t}_θ and \mathbf{t}'_θ by identifying their roots. We show that \mathbf{t} is the desired tree. The reader can easily verify that ρ , constructed by taking the union of ρ_θ and ρ'_θ , is a run of \mathcal{A} on \mathbf{t} , where ρ'_θ is the copy of ρ_θ on \mathbf{t}'_θ and that Γ' is consistent with \mathbf{t} .

It remains to show that a starting extended configuration obtained via $\xrightarrow{\text{grow}}$ corresponds to a real configuration of \mathcal{A} . Assume θ' is a reachable extended configuration and that θ is such that $\theta' \xrightarrow{\text{grow}} \theta$. By definition, all threads $(q, \alpha, d) \in \Delta'$ are such that $\alpha \in \{\top, \top_g\}$. Consider a derivation witnessing the fact that θ' is reachable. Let θ_1 be a starting extended configuration in this derivation such that all other extended configurations between θ_1 and θ' are not starting. Hence we have $\theta_1 \xrightarrow{(\rightarrow_\epsilon \mid \xrightarrow{\text{inc}})^+} \theta'$ (no $\xrightarrow{\text{grow}}$ nor $\xrightarrow{\text{merge}}$ can occur in this derivation as a $\xrightarrow{\text{merge}}$ can only appear right after a $\xrightarrow{\text{grow}}$, because of the m flag, and a $\xrightarrow{\text{grow}}$ would derive a starting extended configuration). By Lemma 4.19, θ' can equivalently be derived from $\hat{\theta}_1 \sim \theta_1$ using a derivation of the form $\xrightarrow{\text{inc}}^* \rightarrow_\epsilon^+$ (a sequence of $\xrightarrow{\text{inc}}$ followed by a sequence of \rightarrow_ϵ). As $\xrightarrow{\text{inc}}$ preserves startingness, this shows that there is a reachable starting extended transition θ_2 such that $\theta_2 \xrightarrow{\rightarrow_\epsilon^+} \theta' \xrightarrow{\text{grow}} \theta$. By induction hypothesis we have a tree \mathbf{t}_{θ_2} and a run ρ_{θ_2} satisfying the induction hypothesis. Let \mathbf{t} be the tree constructed from \mathbf{t}_{θ_2} by adding a new node, having for label and data value those of θ , and a unique immediate subtree \mathbf{t}_{θ_2} . Let ρ be constructed from ρ_{θ_2} as follows: If x is a node of \mathbf{t} occurring in \mathbf{t}_{θ_2} then $\rho(x) = \rho_{\theta_2}(x)$. If y is the new root of \mathbf{t} then, for each thread $(p, \perp, d) \in \Delta_2$ there must be a transition $\xrightarrow{\delta}$ in the derivation from θ_2 to θ' using that thread (otherwise that thread would never disappear and a $\xrightarrow{\text{grow}}$ transition would not be applicable on θ'). Let $\tau = (t, a) \in \delta$ be the corresponding transition of \mathcal{A} . Because of the consistency condition of Γ_2 on \mathbf{t}_{θ_2} we have that $\mathbf{t}_{\theta_2}, x, (p, d) \models t$, where x is the root of \mathbf{t}_{θ_2} and we add to $\rho(y)$ the effect of the action a . The reader can now easily verify that this gives the desired tree and run. \square

By combining the previous Lemmas we immediately obtain the proof of Proposition 4.17. Hence, combining Proposition 4.17 and Corollary 4.16 Theorem 4.1 is proven.

5. SATISFIABILITY OF VERTICAL XPATH.

In order to conclude the proof of Theorem 2.1 it remains to show that BUDA can capture emptiness of $\text{regXPath}(\mathfrak{A}, =)$ expressions. Given a formula η of $\text{regXPath}(\mathfrak{A}, =)$, we say that a BUDA \mathcal{A} is *equivalent* to η if for every data tree \mathbf{t} , \mathbf{t} is accepted by \mathcal{A} iff $\llbracket \eta \rrbracket^{\mathbf{t}} \neq \emptyset$. The main contribution of this section is the following.

Proposition 5.1. *For every $\eta \in \text{regXPath}(\mathfrak{A}, =)$ there exists an equivalent $\mathcal{A} \in \text{BUDA}$ computable from η .*

We first give the general idea of the construction. From $\eta \in \text{regXPath}(\mathfrak{A}, =)$ we actually compute an equivalent BUDA^ε . By Proposition 3.5 this is enough to prove the result. Note that BUDA^ε can easily simulate any positive test $\langle \alpha = \beta \rangle$ or $\langle \alpha \neq \beta \rangle$ of $\text{regXPath}(\mathfrak{A}, =)$ using a **guess** action and tests of the form $\langle \text{exp} \rangle^=$ and $\langle \text{exp} \rangle^\neq$. In the following examples, we disregard the internal alphabet \mathbb{B} in the expressions exp for clarity. Consider, for example, the property $\langle \downarrow_*[a] \neq \uparrow \downarrow[b] \rangle$, which states that there is a descendant labeled a with a different data value than a sibling labeled b . A BUDA^ε automaton can test this property as follows.

- (1) It guesses a data value d and stores it in the register.
- (2) It tests that d can be reached by $\downarrow_*[a]$ with a test $\langle \mathbb{A}^*a \rangle^=$.
- (3) It moves up to its parent.
- (4) It tests that a data value different from d can be reached in one of its children labeled with b , using the test $\langle \mathbb{A}b \rangle^\neq$.

On the other hand, the simulation of negative tests ($\neg \langle \alpha = \beta \rangle$ or $\neg \langle \alpha \neq \beta \rangle$) is more complex as BUDA^ε is not closed under complementation. Nevertheless, the automaton has enough universal behavior (in the operations univ , $\overline{\langle \text{exp} \rangle^=}$ and $\overline{\langle \text{exp} \rangle^\neq}$) in order to do the job. Consider for example the formula $\neg \langle \uparrow^*[b] \downarrow[a] = \downarrow_*[c] \rangle$, that states that no data value is shared between a descendant labeled c and any a -child of a b -ancestor. To test this property, the automaton behaves as follows.

- (1) It creates one thread in state q for every data value in the subtree, using $\text{univ}(q)$.
- (2) A thread in state q tests whether the data value of the register is reachable by $\downarrow_*[c]$, using a test $\langle \mathbb{A}^*c \rangle^=$. If the test is successful, it changes to state p , otherwise it stops and accepts.
- (3) A thread in state p moves up towards the root, and each time it finds a b , it tests that the currently stored data value cannot be reached by $\downarrow[a]$. This is done with a test of the kind $\overline{\langle ba \rangle^=}$.

This is essentially what we do. As usual the details are slightly more complicated. In particular we will have to deal with more complicated regular expressions involving possibly complex node expressions. As our automaton is bottom-up, we will need to compute all loops within a subtree. We will use the internal alphabet \mathbb{B} for this purpose.

Proof of Proposition 5.1. Let η be a node expression in $\text{regXPath}(\mathfrak{A}, =)$. We construct a $\text{BUDA}^\varepsilon \mathcal{A}_\eta$ that tests whether η holds at all the leaves of the tree. Note that this is without any loss of generality, since it is then easy to test any formula η at the root with $\langle \uparrow^*[\neg \langle \uparrow \rangle \wedge \eta] \rangle$.

We denote by $\text{nsub}(\eta)$ the node subformulas of η , and by $\text{psub}(\eta)$ the path subformulas of η . For any $\varphi \in \text{nsub}(\eta)$ we denote by $\overline{\varphi}$ its *simple negation*, that is, $\overline{\varphi} = \psi$ if φ is of the form $\neg \psi$, and $\overline{\varphi} = \neg \varphi$ otherwise. By $\text{nsub}^\neg(\eta)$ we denote the closure of $\text{nsub}(\eta)$ under simple negations.

For technical reasons, we distinguish between the *nesting levels* of the formulas in $\text{nsub}(\eta)$. Node expressions of nesting level 0 are those testing node labels and any boolean combination of those. Node expressions of nesting level $i + 1$ are those of nesting level i plus those the form $\langle \alpha = \beta \rangle$ or $\langle \alpha \neq \beta \rangle$, and any boolean combination of them, where α and β are path expressions using only node subexpressions of nesting level i . We denote by $\text{nsub}_i(\eta)$ the subset of $\text{nsub}^\top(\eta)$ containing node expressions of nesting level i . Similarly we denote by $\text{psub}_i(\eta)$ the path expressions of $\text{psub}(\eta)$ using only formulas in $\text{nsub}_i(\eta)$ as node subexpressions. Note that the maximal nesting level n is bounded by $|\eta|$.

For each $i \leq n$, consider now the finite alphabet

$$\mathbb{A}_{\eta,i} = \{\uparrow, \downarrow, [\varphi] \mid \varphi \in \text{nsub}_i(\eta)\}.$$

Every path expression $\alpha \in \text{psub}_i(\eta)$ can then be interpreted as a regular expression over $\mathbb{A}_{\eta,i}$, and every word $w \in \mathbb{A}_{\eta,i}^*$ can be interpreted as a path expression.

A *path* π of \mathbf{t} is a non-empty string of node positions of \mathbf{t} (*i.e.*, $\pi \in \text{pos}(\mathbf{t})^+$) so that every two consecutive elements of π are in a parent/child relation (*i.e.*, one is the parent of the other). A path π is *looping* if the first and last elements are the same, and it is *non-ascending* if each of its elements is either a descendant of the first element or equal to it. We say that a path π of \mathbf{t} *verifies* $w \in \mathbb{A}_{\eta,i}^*$ if π behaves according to the sequence of letters of w . More formally this means:

- $w = \epsilon$ and $|\pi| = 1$;
- $w = \uparrow w'$, $\pi = uv\pi'$ and v is the parent of u in \mathbf{t} and $v\pi'$ verifies w' ;
- $w = \downarrow w'$, $\pi = uv\pi'$ and v is a child of u in \mathbf{t} and $v\pi'$ verifies w' ; or
- $w = [\varphi]w'$, $\pi = u\pi'$ and $u \in \llbracket \varphi \rrbracket^{\mathbf{t}}$ and π verifies w' .

By the characterizations of regular languages, for each $i \leq n$, there exists a finite monoid² \mathcal{M}_i and a homomorphism $g_i : \mathbb{A}_{\eta,i}^* \rightarrow \mathcal{M}_i$ such that for every $\alpha \in \text{psub}_i(\eta)$ there is a set $S_\alpha \subseteq \mathcal{M}_i$ so that $w \in \mathbb{A}_{\eta,i}^*$ is recognized by α iff $g_i(w) \in S_\alpha$. Let us denote by $1_{\mathcal{M}_i}$ the neutral element of \mathcal{M}_i and by ν_i, ν'_i the elements of \mathcal{M}_i .

The internal alphabet of \mathcal{A}_η is $\mathbb{B} = \wp(\mathcal{M}_0) \times \cdots \times \wp(\mathcal{M}_n)$. Intuitively, \mathcal{A}_η accepts trees $\mathbf{t} \otimes \mathbf{b}$ so that for each $i \leq n$ and any node x , the i^{th} component of $\mathbf{b}(x)$, denoted $\mathbf{b}_i(x)$ in the sequel, contains the set of all $\nu_i \in \mathcal{M}_i$ so that there is $w \in \mathbb{A}_{\eta,i}^*$ where

- (1) $g_i(w) = \nu_i$, and
- (2) there is a non-ascending looping path π of \mathbf{t} so that π starts and ends in x and verifies w .

In other words, $\mathbf{b}(x)$ contains all the information about the non-ascending looping paths at x and one of the chief tasks of \mathcal{A}_η is to ensure that the $\mathbf{b}(x)$ are properly set. For this, we have in the set of states Q of \mathcal{A}_η a state $\llbracket \varphi \rrbracket$ for any $\varphi \in \text{nsub}^\top(\eta)$. We will design \mathcal{A}_η such that in an accepting run, if a thread in state $\llbracket \varphi \rrbracket$ is started at a node $x \in \mathbf{t}$ then $x \in \llbracket \varphi \rrbracket^{\mathbf{t}}$. When this is the case, properties 1 and 2 are enforced by starting a thread at each leaf of \mathbf{t} with a state $q_{\mathbb{B}}^i$. A thread in state $q_{\mathbb{B}}^i$ moves up in the tree while performing the following tests and actions at any node x , where $\Lambda_i(x)$ is the set of all $\varphi \in \text{nsub}_i(\eta)$ such that $x \in \llbracket \varphi \rrbracket^{\mathbf{t}}$:

- If x is a leaf, then $\mathbf{b}_i(x)$ is the submonoid of \mathcal{M}_i generated by $g_i(\Lambda_i(x))$. This property can be enforced by guessing a maximally consistent set $L_i \subseteq \text{nsub}_i(\eta)$ of formulas that hold at the node, testing whether $\mathbf{b}_i(x)$ has the desired form (*i.e.*, that it is the submonoid

²Unlike in Section 3.1 we work here with an automata model with ϵ transitions. Therefore it is more convenient to use monoids instead of semigroups.

generated by $g_i(L_i)$, and verifying that $L_i = \Lambda_i(x)$ by starting a thread with state $\langle\langle\varphi\rangle\rangle$ for every $\varphi \in L_i$.

- If x is not a leaf then $\mathbf{b}_i(x)$ is the submonoid of M_i generated by $g_i(\Lambda_i(x)) \cup S_i$ where $S_i = \bigcup_{y \text{ child of } x} g_i(\downarrow)\mathbf{b}_i(y)g_i(\uparrow)$. This is done by guessing L_i and S_i , performing the same tests and actions as above concerning L_i and testing that S_i is correct by testing the internal labels of the children of the current nodes using tests of the form $\langle\langle(\mathbb{A} \times \mathbb{B}) \cdot (\mathbb{A} \times \{b\})\rangle\rangle$ and $\overline{\langle\langle(\mathbb{A} \times \mathbb{B}) \cdot (\mathbb{A} \times \{b\})\rangle\rangle}$ for appropriate $b \in \mathbb{B}$.

Thus, the initial state q_0 of \mathcal{A}_η will simply create $n + 2$ threads, with states $q_{\mathbb{B}}^i$ for every i , and one with state $\langle\langle\eta\rangle\rangle$.

Intuitively, the automaton that verifies φ starts with one thread in state $\langle\langle\varphi\rangle\rangle$ at every leaf. The boolean connectives \wedge, \vee are treated by the alternation/nondeterminism. If φ is $\langle\alpha = \beta\rangle$, the automaton guesses the witness data value and verifies that we can reach that value through α (resp. β) by going to a state $\langle\langle\alpha\rangle\rangle_{1_{\mathcal{M}}}^-$. To verify this, the state $\langle\langle\alpha\rangle\rangle_{1_{\mathcal{M}}}^-$ checks that either the data in the register is reachable through α in the subtree and through a test $\langle e_\alpha \rangle^-$ for a suitable regular expression e_α , or it is elsewhere as a result of α starting with an upward axis, and the automaton recursively moves up to some $\langle\langle\alpha\rangle\rangle_\nu^-$ remembering the ‘history’ of labels when going up in ν . Tests of the form $\langle\alpha = \beta\rangle$ are treated similarly. Finally, for a test $\neg\langle\alpha = \beta\rangle$ (resp. $\neg\langle\alpha \neq \beta\rangle$), the automaton uses the power of unbounded alternation: it creates a new thread for each data value in the subtree reachable through α and passes the control to the states $\langle\langle\beta\rangle\rangle_\nu^{-=}$ (resp. $\langle\langle\beta\rangle\rangle_\nu^{-\neq}$), which are in charge of testing that no node can be reached through α with $=$ (resp. \neq) data value. Again, this has to be repeated for every ancestor as well, taking into account the type of the path in ν . Symmetrical conditions are also requested on β .

More concretely, the automaton \mathcal{A}_η uses states of the form $\langle\langle\psi\rangle\rangle$ or $\langle\langle\alpha\rangle\rangle_\nu^\odot$, where ψ is a subformula of φ , α is a path expression of φ , and $\odot \in \{=, \neq, \neg=, \neg\neq\}$ and $\nu \in \bigcup_i \mathcal{M}_i$. It now remains to explain how \mathcal{A}_η can test whether $x \in \llbracket\varphi\rrbracket^t$ by starting a thread with state $\langle\langle\varphi\rangle\rangle$ at node x . We explain this by induction on the structure of φ . Using alternation of the automata model we can easily simulate disjunctions and conjunctions. Testing node labels is also a simple task. Altogether it remains to explain how formulas of the form $\langle\alpha = \beta\rangle$ and $\langle\alpha \neq \beta\rangle$, or their negations can be tested by \mathcal{A}_η . In order to do so, for any $i \leq n$, $\nu_i \in \mathcal{M}_i$, $\alpha, \beta \in \mathbf{psub}_i(\eta)$, $\otimes \in \{=, \neq, \neg=, \neg\neq\}$, $\odot \in \{\neg=, \neg\neq\}$, we have in Q the states $\langle\langle\alpha\rangle\rangle_{\nu_i}^\otimes$ and $\langle\langle\alpha, \beta\rangle\rangle_{\nu_i}^\odot$, where $\alpha, \beta \in \mathbf{psub}_i(\eta)$.

We say that a thread $(q, d) \in Q \times \mathbb{D}$ has an *accepting run from y* if there is an accepting run ρ where instead of requiring that for every leaf x of \mathbf{t} , $\rho(x)$ is initial, we require that

- for every leaf x of \mathbf{t} , $\rho(x)$ contains a thread in state $q_{\mathbb{B}}^i$, where i is the maximum nesting level of the formulas in the state q and
- $\rho(y)$ contains the thread (q, d) .

The transitions of \mathcal{A}_η will be built so that the following conditions are met, for every $d \in \mathbb{D}$.

- (a) A thread $(\langle\langle\varphi\rangle\rangle, d)$ has an accepting run from x implies $x \in \llbracket\varphi\rrbracket^t$.
- (b) A thread $(\langle\langle\alpha\rangle\rangle_{\nu_i}^=, d)$ (resp. $(\langle\langle\alpha\rangle\rangle_{\nu_i}^{\neq}, d)$) has an accepting run from x implies there is a path π and a word $w \in \mathbb{A}_{\eta, i}^*$ so that
 - π starts at x and ends at a node with the same (resp. different) data value as d , and
 - π verifies w and $\nu_i \cdot g_i(w) \in S_\alpha$.
- (c) A thread $(\langle\langle\alpha\rangle\rangle_{\nu_i}^{-=}, d)$ (resp. $(\langle\langle\alpha\rangle\rangle_{\nu_i}^{-\neq}, d)$) has an accepting run from x implies there is no path π and no word $w \in \mathbb{A}_{\eta, i}^*$ so that

- π starts at x and ends at a node with the same (resp. different) data value as d , and
 - π verifies w and $\nu_i \cdot g_i(w) \in S_\alpha$.
- (d) A thread $(\langle\langle\alpha, \beta\rangle\rangle_{\nu_i}^{\neg=}, d)$ (resp. $(\langle\langle\alpha, \beta\rangle\rangle_{\nu_i}^{\neg\neq}, d)$) has an accepting run from x implies there are no paths π, π' and no words $w, w' \in \mathbb{A}_{\eta, i}^*$ so that
- both π and π' start at x and end at nodes with the same (resp. different) data value, and
 - π verifies w where $\nu_i \cdot g_i(w) \in S_\alpha$, and π' verifies w' where $\nu_i \cdot g_i(w') \in S_\beta$.

Note that condition (a) is enough to conclude the correctness of the construction. The other conditions are here to help enforcing this condition.

It now remains to set the transition function of \mathcal{A}_η in order to ensure all the properties stated above. First note the following observation. For every $i \leq n$ and $\nu_i \in \mathcal{M}_i$ and $\alpha \in \mathbf{psub}_i(\eta)$ the set of all words $(a_1, b_1), \dots, (a_m, b_m)$ of $(\mathbb{A} \times \mathbb{B})^+$ such that there are $\nu_{i,1} \in b_1, \dots, \nu_{i,m} \in b_m$ with

$$\nu_i \cdot (\nu_{i,1} \cdot g_i(\downarrow) \cdot \nu_{i,2} \cdot g_i(\downarrow) \cdot \dots \cdot \nu_{i,m-1} \cdot g_i(\downarrow) \cdot \nu_{i,m}) \in S_\alpha$$

is a regular language and we denote by $\exp(\alpha, \nu_i)$ the corresponding regular expression.

We now describe the behavior of each thread at a node x depending on its state. In this description we assume that α and β are in $\mathbf{psub}_i(\eta)$.

- $\langle\langle\alpha = \beta\rangle\rangle$: In this case \mathcal{A}_η guesses a data value, stores it in its register using `guess` and continues the execution with both states $\langle\langle\alpha\rangle\rangle_{1_{\mathcal{M}_i}}^{\overline{=}}$ and $\langle\langle\beta\rangle\rangle_{1_{\mathcal{M}_i}}^{\overline{=}}$, that will test if there exist two nodes accessible by α and β respectively such that both carry the data value of the register.
- $\langle\langle\alpha \neq \beta\rangle\rangle$: Similarly as above, \mathcal{A}_η guesses a data value, stores it in its register and continues the execution with both states $\langle\langle\alpha\rangle\rangle_{1_{\mathcal{M}_i}}^{\overline{=}}$ and $\langle\langle\beta\rangle\rangle_{1_{\mathcal{M}_i}}^{\neq}$, which are responsible of testing that there is a α path leading to a node with the same data value as the one in the register and a β path leading to a node with a different data value.
- $\langle\langle\alpha\rangle\rangle_{\nu_i}^{\overline{=}}$ or $\langle\langle\beta\rangle\rangle_{\nu_i}^{\neq}$: We denote by \odot the symbol $=$ or \neq occurring in superscript. In this case \mathcal{A}_η chooses non-deterministically between one of the following actions.
 - It checks that the required data value is already in the subtree, making the test $\langle\exp(\alpha, \nu_i)\rangle^{\odot}$.
 - It moves up and switches state to $\langle\langle\alpha\rangle\rangle_{\nu_i \cdot \nu'_i \cdot g_i(\uparrow)}^{\odot}$ for some $\nu'_i \in \mathbf{b}_i(x)$.
- $\langle\langle\neg(\alpha = \beta)\rangle\rangle$ or $\langle\langle\neg(\alpha \neq \beta)\rangle\rangle$: We denote by \odot the symbol $=$ or \neq occurring in the middle. In this case \mathcal{A}_η continues the execution with state $\langle\langle\alpha, \beta\rangle\rangle_{1_{\mathcal{M}_i}}^{\neg\odot}$.
- $\langle\langle\alpha, \beta\rangle\rangle_{\nu_i}^{\neg=}$ or $\langle\langle\alpha, \beta\rangle\rangle_{\nu_i}^{\neg\neq}$: We denote by \odot the symbol $=$ or \neq occurring in superscript. In this case \mathcal{A}_η performs all the following actions using alternation:
 - If the test `root` succeeds, it moves up, and creates a thread in state $\langle\langle\alpha, \beta\rangle\rangle_{\nu_i \cdot \nu'_i \cdot g_i(\uparrow)}^{\neg\odot}$ for each $\nu'_i \in \mathbf{b}_i(x)$.
 - For every data value d in the subtree that can be reached via $\exp(\alpha, \nu_i)$ it moves to state $\langle\langle\beta\rangle\rangle_{\nu_i}^{\neg\odot}$ with data value d . This can be achieved by performing a `univ` operation and then by choosing non-deterministically to perform one of the following transitions
 - test $\langle\exp(\alpha, \nu_i)\rangle^{\overline{=}}$ and move to state $\langle\langle\beta\rangle\rangle_{\nu_i}^{\neg\odot}$, or
 - test $\overline{\langle\exp(\alpha, \nu_i)\rangle^{\overline{=}}}$ and accept.
 - Analogously, for every data value d in the subtree that can be reached via $\exp(\beta, \nu_i)$, it moves to state $\langle\langle\alpha\rangle\rangle_{\nu_i}^{\neg\odot}$ with data value d .

- $\langle\langle\alpha\rangle\rangle_{\nu_i}^{\neg=}$ or $\langle\langle\alpha\rangle\rangle_{\nu_i}^{\neg\neq}$: We denote by \odot the symbol $=$ or \neq occurring in superscript. In this case \mathcal{A}_η performs all the following actions using alternation:
 - if the test $\overline{\text{root}}$ succeeds, for all $\nu'_i \in \mathbf{b}_i(x)$, it starts a new thread in state to $\langle\langle\alpha\rangle\rangle_{\nu_i \cdot \nu'_i \cdot g_i(\uparrow)}^{\neg\odot}$ at the parent of the current node.
 - tests that $\langle\text{exp}(\alpha, \nu_i)\rangle^{\odot}$ holds.

Correctness. We show, for every position x and state that conditions (a)–(d) hold. We proceed by induction on the nesting level of the expressions involved. Recall that once the node expressions of nesting level i are correctly enforced by a thread of the form $\langle\langle\varphi\rangle\rangle$ then the behavior of \mathcal{A}_η enforces that $\mathbf{b}_i(x)$ contains the correct information for all x (i.e. verify 1 and 2). Therefore $\text{exp}(\alpha, \nu_i)$ does find a downward path evaluating to ν'_i such that $\nu_i \cdot \nu'_i \in S_\alpha$.

The base case is the nesting level 0. At this level node expressions only test the labels of the current node and this is exactly what the automaton does. Hence $\mathbf{b}_0(x)$ verifies 1 and 2.

We now assume a correct behavior for nesting level i and show the same for level $i + 1$. For this we need to show that (a) holds for $\varphi \in \text{nsb}_{i+1}(\eta)$ and that (b)–(d) holds for $\alpha, \beta \in \text{psb}_i(\eta)$ and $\nu_i \in \mathcal{M}_i$.

We prove (b) by induction on the depth of x starting from the root.

Suppose that there is an accepting run of $(\langle\langle\alpha\rangle\rangle_{\nu_i}^{\neg=}, d)$ from x on $\mathbf{a} \otimes \mathbf{b} \otimes \mathbf{d}$. By definition of the transition set, one of the following must hold.

- The test $\langle\text{exp}(\alpha, \nu_i)\rangle^{\neg=}$ succeeded. Then, by inductive hypothesis on i , there is a downward path π starting at x and ending at some position with data value d so that the path verifies some $w \in \mathbb{A}_{\eta, i}^*$ with $\nu_i \cdot g_i(w) \in S_\alpha$ and we are done. Note that this is the only possible case at the root of \mathbf{t} , hence proving the base case.
- $(\langle\langle\alpha\rangle\rangle_{\nu_i \cdot \nu'_i \cdot g_i(\uparrow)}^{\neg=}, d)$ has an accepting run from the parent y of x . By inductive hypothesis on i there is a non-ascending looping path π'' that starts and ends at x , and verifying a word $w'' \in \mathbb{A}_{\eta, i}$ so that $g_i(w'') = \nu'_i$. By inductive hypothesis on the depth of x , this means that there is a path π that starts at y and ends at a node with data value d , so that π verifies some $w \in \mathbb{A}_{\eta, i}^*$ where $\nu_i \cdot \nu'_i \cdot g_i(\uparrow) \cdot g_i(w) \in S_\alpha$. Therefore the path $\pi' = \pi'' \cdot \pi$ starts at x and ends at a node with data value d , and π' verifies $w' = w'' \uparrow w$, where $\nu_i \cdot g_i(w') \in S_\alpha$.

Suppose now there is a path π starting in x and ending at a node with a data value d , verifying some $w \in \mathbb{A}_{\eta, i}^*$ so that $\nu_i \cdot g_i(w) \in S_\alpha$. We want to show that the thread $(\langle\langle\alpha\rangle\rangle_{\nu_i}^{\neg=}, d)$ has an accepting run starting at x . Then either

- π is non-ascending, then by induction on i , $w \in \text{exp}(\alpha, \nu_i)$ and the test $\langle\text{exp}(\alpha, \nu_i)\rangle^{\neg=}$ succeeds.
- $\pi = \pi' \cdot \pi''$ where π' is the maximal prefix of π that is looping and non-ascending and π'' starts at the parent y of x . Let w' be such that $w = w'w''$, π' verifies w' and π'' verifies w'' . Let $\nu'_i = g_i(w')$. By induction on the depth of x , we know that the thread $(\langle\langle\alpha\rangle\rangle_{\nu_i \cdot \nu'_i \cdot g_i(\uparrow)}^{\neg=}, d)$ has an accepting run starting at y .

The case of $\langle\langle\alpha\rangle\rangle_{\nu_i}^{\neq}$ is treated similarly.

The cases (c) and (d) are treated similarly.

Consider for example (c). The proof is again by induction on the depth of x starting from the root. We do only the case $\langle\langle\alpha\rangle\rangle_{\nu_i}^{\neg=}$ as the other one is proven identically.

Suppose that there is an accepting run of $(\langle\langle\alpha\rangle\rangle_{\nu_i}^{\neg=}, d)$ from x on $\mathbf{a} \otimes \mathbf{b} \otimes \mathbf{d}$. By definition of the transition set, all of the following must hold.

- The test $\overline{\langle\exp(\alpha, \nu_i)\rangle}^=$ succeeded. Then, by inductive hypothesis on i , there is no downward path π starting at x and ending at some position with data value d so that the path verifies some $w \in \mathbb{A}_{\eta, i}^*$ with $\nu_i \cdot g_i(w) \in S_\alpha$. Note that this is the only possible case at the root of \mathbf{t} , hence proving the base case.
- $(\langle\langle\alpha\rangle\rangle_{\nu_i \cdot \nu'_i \cdot g_i(\uparrow)}^{\neg=}, d)$ has an accepting run from the parent y of x for any $\nu'_i \in \mathbf{b}_i(x)$. By inductive hypothesis on the depth of x , this means that there is no path π that starts at y and ends at a node with data value d , so that π verifies some $w \in \mathbb{A}_{\eta, i}^*$ where $\nu_i \cdot \nu'_i \cdot g_i(\uparrow) \cdot g_i(w) \in S_\alpha$. Assume now that there was a path π' starting from x and ending at some position with data value d so that the path verifies some $w \in \mathbb{A}_{\eta, i}^*$ with $\nu_i \cdot g_i(w) \in S_\alpha$. If this path is not ascending then it is a contradiction with the previous case. If this path is ascending then it starts with a non-ascending loop at a and the continue from y . By induction on i the non-ascending loop part evaluates to $\nu \in \mathbf{b}(x)$ and therefore we also get a contradiction with the the fact that $(\langle\langle\alpha\rangle\rangle_{\nu_i \cdot \nu'_i \cdot g_i(\uparrow)}^{\neg=}, d)$ has an accepting run from y .

The converse direction is treated similarly.

A similar reasoning shows the case (d) and is omitted here.

Property (a) is now a simple consequence of (b)–(d). □

6. CONCLUDING REMARKS

We have exhibited a decidable class of automata over data trees. This automaton model is powerful enough to code node expressions of $\text{regXPath}(\mathfrak{A}, =)$. Therefore, since these expressions are closed under negation, we have shown decidability of the satisfiability, containment and equivalence problems for node expressions of $\text{regXPath}(\mathfrak{A}, =)$.

Consider the containment problem for *path* expressions as the problem of, given two path expressions α and β of our logic, whether $\llbracket\alpha\rrbracket^{\mathbf{t}} \subseteq \llbracket\beta\rrbracket^{\mathbf{t}}$ for all data trees \mathbf{t} . It is straightforward that the technique used in [tCL09, Proposition 4] to reduce containment of path expressions into the satisfiability problem for node expressions works also in our context. Indeed, this technique is independent of having data equality tests and only requires that the logic be closed under boolean connectives. We therefore obtain a decision procedure for this problem as a corollary of Theorem 2.1.

Proposition 6.1. *The containment problem for path expressions of $\text{regXPath}(\mathfrak{A}, =)$ is decidable.*

Our decision algorithm relies heavily on the fact that we work with **unranked** data trees. As already shown in [FS09] without this assumption $\text{XPath}(\mathfrak{A}, =)$ would be undecidable. In particular if we further impose the presence of a DTD, $\text{XPath}(\mathfrak{A}, =)$ becomes undecidable, unless the DTD is simple enough for being expressible as a BUDA.

REFERENCES

- [BFG08] Michael Benedikt, Wenfei Fan, and Floris Geerts. XPath satisfiability in the presence of DTDs. *Journal of the ACM*, 55(2):1–79, 2008.
- [BK08] Michael Benedikt and Christoph Koch. XPath leashed. *ACM Computing Surveys*, 41(1), 2008.
- [BMSS09] Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *Journal of the ACM*, 56(3):1–48, 2009.
- [Dic13] Leonard E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *The American Journal of Mathematics*, 35(4):413–422, 1913.
- [Fig09] Diego Figueira. Satisfiability of downward XPath with data equality tests. In *ACM Symposium on Principles of Database Systems (PODS'09)*, 2009.
- [Fig10] Diego Figueira. Forward-XPath and extended register automata on data-trees. In *International Conference on Database Theory (ICDT'10)*, 2010.
- [Fig11] Diego Figueira. A decidable two-way logic on data words. In *Annual IEEE Symposium on Logic in Computer Science (LICS'11)*, Toronto, Canada, 2011.
- [Fig12] Diego Figueira. Alternating register automata on finite data words and trees. *Logical Methods in Computer Science*, 8(1:22), 2012.
- [Fig13] Diego Figueira. On XPath with transitive axes and data tests. In Wenfei Fan, editor, *ACM Symposium on Principles of Database Systems (PODS'13)*, 2013.
- [FS01] Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.
- [FS09] Diego Figueira and Luc Segoufin. Future-looking logics on data words and trees. In *Intl. Symp. on Mathematical Foundations of Computer Science (MFCS'09)*, 2009.
- [FS11] Diego Figueira and Luc Segoufin. Bottom-up automata on data trees and vertical XPath. In *International Symposium on Theoretical Aspects of Computer Science*, pages 93–104, 2011.
- [GF05] Floris Geerts and Wenfei Fan. Satisfiability of XPath queries with sibling axes. In *Intl. Symp. on Database Programming Languages (DBPL'05)*, 2005.
- [GKP05] Georg Gottlob, Christoph Koch, and Reinhard Pichler. Efficient algorithms for processing XPath queries. *ACM Transactions on Database Systems*, 30(2):444–491, 2005.
- [JL11] Marcin Jurdziński and Ranko Lazić. Alternating automata on data trees and XPath satisfiability. *ACM Transactions on Computational Logic*, 2(3), 2011.
- [Min67] Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.
- [tCL09] Balder ten Cate and Carsten Lutz. The complexity of query containment in expressive fragments of XPath 2.0. *Journal of the ACM*, 56(6), 2009.