Alternating register automata on finite words and trees

Diego Figueira

HAL Id: hal-01799710
https://hal.archives-ouvertes.fr/hal-01799710

Submitted on 25 May 2018

Alternating register automata
on finite data words and trees[*]

DIEGO FIGUEIRA

INRIA & ENS Cachan, LSV, France

University of Edinburgh, UK

ABSTRACT. We study alternating register automata on data words and data trees in relation to logics. A data word (resp. data tree) is a word (resp. tree) whose every position carries a label from a finite alphabet and a data value from an infinite domain. We investigate one-way automata with alternating control over data words or trees, with one register for storing data and comparing them for equality. This is a continuation of the study started by Demri, Lazić and Jurdziński.

From the standpoint of register automata models, this work aims at two objectives: (1) simplifying the existent decidability proofs for the emptiness problem for alternating register automata; and (2) exhibiting decidable extensions for these models.

From the logical perspective, we show that (a) in the case of data words, satisfiability of LTL with one register and quantification over data values is decidable; and (b) the satisfiability problem for the so-called *forward* fragment of XPath on XML documents is decidable, even in the presence of DTDs and even of key constraints. The decidability is obtained through a reduction to the automata model introduced. This fragment contains the child, descendant, next-sibling and following-sibling axes, as well as data equality and inequality tests.

1. INTRODUCTION

In static analysis of databases as in software verification, we frequently find the need of reasoning with *infinite alphabets*. In program verification one may need to decide statically whether a program satisfies some given specification; and the necessity of dealing with infinite alphabets can arise from different angles. For example, in the presence of concurrency,

suppose that an unbounded number of processes run, each one with its process identification, and we must deal with properties expressing the interplay between these processes. Further, procedures may have parameters and data from some infinite domain could be exchanged as arguments. On the other hand, in the databases context, static analysis on XML and its query languages recurrently needs to take into account not only the labels of the nodes, but also the actual data contained in the attributes. It is hence important to study formalisms to reason with words or trees that can carry elements from some infinite domain.

This work is about decidable alternating register automata models on data words and data trees in relation to logics manipulating data. This is a continuation of the investigation carried out by Demri, Lazić and Jurdziński [DL09, JL11]. A non trivial consequence of our contribution on alternating automata is that the satisfiability problem for the forward fragment of XPath on XML documents is decidable.

We consider two kinds of data models: data words and data trees. A data word (data tree) is a finite word (unranked finite tree) whose every position carries a pair of elements: a symbol from a finite alphabet and and an element (a *datum*) from an infinite set (the *data domain*). We work on finite structures, and all the results we present are relative to finite words and trees.

Over these two models we consider two formalisms: alternating register automata on the one hand, and logics on the other. Each automata model is related to a logic, in the sense that the satisfiability of the logic can be reduced to the emptiness of the automata model. Both automata models we present (one for data words, the other for data trees) have in essence the same behavior. Let us give a more detailed description of these formalisms.

Automata. The automata model we define is based on the ARA model (for Alternating Register Automata) of [DL09] in the case of data words, or the ATRA model (for Alternating Tree Register Automata) of [JL11] in the case of data trees. ARA are one-way automata with alternating control and one register to store data values for later comparison. ATRA correspond to a natural extension of ARA over data trees. The ATRA model can move in two directions: to the leftmost child, and/or to the next sibling to the right. Both models were shown to have a decidable emptiness problem. The proofs of decidability are based on non trivial reductions to a class of decidable counter automata with faulty increments.

In the present work, decidability of these models is shown by interpreting the semantics of the automaton in the theory of well-quasi-orderings in terms of a well-structured transition system [see FS01]. The object of this alternative proof is twofold. On the one hand, we propose a direct, unified and self-contained proof of the main decidability results of [DL09, JL11]. Whereas in [DL09, JL11] decidability results are shown by reduction to a class of faulty counter automata, here we avoid such translation, and show decidability directly interpreting the configurations of the automata in the theory of well-structured transition systems. We stress, however, that the underlying techniques used here are similar to those of [DL09, JL11]. On the other hand, we further generalize these results. Our proof can be easily extended to show the decidability of the nonemptiness problem for two powerful extensions. These extensions consist in the following abilities: (a) the automaton can nondeterministically *guess* any data value of the domain and store it in the register; and (b) it can make a certain kind of universal quantification over the data values seen along the run of the automaton, in particular over the data values seen so far. We name these extensions guess and spread respectively. These extensions can be both added to the

ARA model or to the ATRA model, since our proofs for ARA and ATRA emptiness problems share the same core. We call the class of alternating register automata with these extensions as ARA(guess, spread) in the case of data words, or ATRA(guess, spread) in the case of data trees. We demonstrate that these extensions are also decidable if the data domain is equipped with a linear order and the automata model is extended accordingly. Further, these models are powerful enough to decide a large fragment of XPath, and of a temporal logic with registers.

XPath. This work is originally motivated by the increasing importance of reasoning tasks about XML documents. An XML document can be seen as an unranked ordered tree where each node carries a *label* from a finite alphabet and a set of *attributes*, each with an associated datum from some infinite domain. A data tree is a simplification of an XML document that happens to be equivalent for the problems treated here.

XPath is arguably the most widely used XML node selecting language, part of XQuery and XSLT; it is an open standard and a W3C Recommendation [CD99]. Static analysis on XML languages is crucial for query optimization tasks, consistency checking of XML specifications, type checking transformations, or many applications on security. Among the most important problems are those of query equivalence and query containment. By answering these questions we can decide at compile time whether the query contains a contradiction, and thus whether the computation of the query on the document can be avoided, or if one query can be safely replaced by a simpler one. For logics closed under boolean combination, these problems reduce to satisfiability checking, and hence we focus on this problem. Unfortunately, the satisfiability problem for XPath with *data tests* is undecidable [GF05] even when the data domain has no structure (*i.e.*, where the only data relation available is the test for equality or inequality). It is then natural to identify and study decidable expressive fragments.

In this work we prove that the *forward* fragment of XPath has a decidable satisfiability problem by a reduction to the nonemptiness problem of ATRA(guess, spread). Let us describe this logic. Core-XPath [GKP05] is the fragment of XPath that captures all the navigational behavior of XPath. It has been well studied and its satisfiability problem is known to be decidable in EXPTIME in the presence of DTDs [Mar04]. We consider an extension of this language with the possibility to make *equality* and *inequality* tests between attributes of XML elements. This logic is named Core-Data-XPath in [BMSS09], and its satisfiability problem is undecidable [GF05]. Here we address a large fragment of Core-Data-XPath named 'forward XPath', that contains the 'child', 'descendant', 'self-or-descendant', 'next-sibling', 'following-sibling', and 'self-or-following-sibling' axes. For economy of space we refer to these axes as $\downarrow, \downarrow^+, \downarrow^*, \rightarrow, \rightarrow^+, \rightarrow^*$ respectively. Note that $\rightarrow^+$ and $\rightarrow^*$ are interdefinable in the presence of $\rightarrow$, and similarly with $\downarrow^+$ and $\downarrow^*$. We then refer to this fragment as XPath($\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, =$), where '=' is to indicate that the logic can express equality or inequality tests between data values.

Although our automata model does not capture forward-XPath in terms of expressiveness, we show that there is a reduction to the nonemptiness problem of ATRA(guess, spread). These automata can recognize any regular language, in particular a DTD, a Relax NG document type, or the core of XML Schema (stripped of functional dependencies). Since we show that XPath($\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, =$) can express unary key constraints, it then follows that satisfiability of forward-XPath in the presence of DTDs and key constraints is decidable. This

settles a natural question left open in [JL11], where decidability for a restriction of forward-XPath was shown. The fragment treated in the cited work is restricted to data tests of the form $\langle \varepsilon = \alpha \rangle$ (or $\langle \varepsilon \neq \alpha \rangle$), that is, formulæ that test whether there exists an element accessible via the α-relation with the same (resp. different) data value as the *current* node of evaluation. However, the forward fragment allows unrestricted tests $\langle \alpha = \beta \rangle$ and makes the coding into a register automata model highly non trivial. As a consequence, we also answer positively the open question raised in [BFG08] on whether the downward fragment of XPath in the presence of DTDs is decidable.[1]

Temporal logics. ARA(guess, spread) over data words also yield new decidability results on the satisfiability for some extensions of the temporal logic with one register denoted by $\mathsf{LTL}^{\downarrow}(\mathsf{U}, \mathsf{X})$ in [DL09]. This logic contains a 'freeze' operator to store the current datum and a 'test' operator to test the current datum against the stored one. Our automata model captures an extension of this logic with quantification over data values, where we can express "*for all data values in the past, φ holds*", or "*there exists a data value in the future where φ holds*". Indeed, none of these two types of properties can be expressed in the previous formalisms of [DL09] and [JL11]. These quantifiers may be added to $\mathsf{LTL}^{\downarrow}(\mathsf{U}, \mathsf{X})$ over data words without losing decidability. What is more, decidability is preserved if the data domain is equipped with a linear order that is accessible by the logic. Also, by a translation into ATRA(guess, spread), these operators can be added to a CTL version of this logic over data trees, or to the μ-calculus treated in [JL07][2]. However, adding the *dual* of either of these operators results in an undecidable logic.

Contribution. From the standpoint of register automata models, our contributions can be summarized as follows.
- We exhibit a unified framework to show decidability for the emptiness problem for alternating register automata. This proof has the advantage of working both for data words and data trees practically unchanged. It is also a simplification of the existing decidability proofs.
- We exhibit decidable extensions for these models of automata. These extensions work for automata either running over words or trees. For each of these models there are consequences on the satisfiability of some expressive logics.

From the standpoint of logics, we show the following results.
- In the case of data trees, we show that the satisfiability problem for the 'forward' fragment of XPath with data test equalities and inequalities is decidable, even in the presence of DTDs (or any regular language) and unary key constraints. Decidability is shown through a reduction to the emptiness problem for ATRA(guess, spread) on data trees.
- We show that the temporal logic $\mathsf{LTL}^{\downarrow}(\mathsf{U}, \mathsf{X})$ for data words extended with some quantification over data values is decidable. This result is established thanks to a translation from formulæ to alternating register automata of ARA(guess, spread).

---

[1]The satisfiability problem on downward XPath but in the *absence* of DTDs is shown to be ExpTime-complete in [Fig09].

[2]This is the conference version of [JL11].

Related work. The main results presented here first appeared in the conference paper [Fig10a]. Here we include the full proofs and the analysis for alternating register automata on *data words* (something that was out of the scope of [Fig10a]) as well as its relation to temporal logics. Also, we show how to proceed in the presence of a linear order, maintaining decidability. The results contained here also appear in the thesis [Fig10b, Ch. 3 and 6].

By the lower bounds given in [DL09], the complexity of the problems we treat in the work is very high: non-primitive-recursive. This lower bound is also known to hold for the logics we treat here $\mathsf{LTL}^{\downarrow}(\mathsf{U}, \mathsf{X})$ and forward-XPath. In fact, even very simple fragments of $\mathsf{LTL}^{\downarrow}$ and XPath are known to have non-primitive-recursive lower bounds, including the fragment of [JL11] or even much simpler ones without the one-step '$\rightarrow$' axis, as shown in [FS09]. This is the reason why in this work we limit ourselves to decidability / undecidability results.

The work in [BFG08] investigates the satisfiability problem for many XPath logics, mostly fragments without negation or without data equality tests in the absence of sibling axes. Also, in [Fig09] there is a study of the satisfiability problem for *downward* XPath fragments with and without data equality tests. All these are sub-fragments of forward-XPath, and notably, none of these works considers *horizontal* axes to navigate between siblings. Hence, by exploiting the bisimulation invariance property enjoyed by these logics, the complexity of the satisfiability problem is kept relatively low (at most ExpTime) in the presence of data values. However, as already mentioned, when horizontal axes are present most of the problems have a non-primitive-recursive complexity. In [GF05], several fragments with horizontal axes are treated. The only fragment with data tests and negation studied there is incomparable with the forward fragment, and it is shown to be undecidable. In [FS11] it is shown that the *vertical* fragment of XPath is decidable in non-primitive-recursive time. This is the fragment with both *downward* and *upward* axes, but notably no horizontal axes (in fact, adding horizontal axes to the vertical fragment results in undecidability).

The satisfiability of first-order logic with two variables and data equality tests is explored in [BMSS09]. It is shown that FO^2 with local one-step relations to move around the data tree and a data equality test relation is decidable. [BMSS09] also shows the decidability of a fragment of $\mathsf{XPath}(\uparrow, \downarrow, \leftarrow, \rightarrow, =)$ with sibling and upward axes. However, the logic is restricted to one-step axes and to data formulæ of the kind $\langle \varepsilon = \alpha \rangle$ (or $\neq$), while the fragment we treat here cannot move upwards but has transitive axes and unrestricted data tests.

2. Preliminaries

Notation. We first fix some basic notation. Let $\wp(C)$ denote the set of subsets of C, and $\wp_{<\infty}(C)$ be the set of *finite* subsets of C. Let $\mathbb{N} = \{1, 2, \dots\}$ be the set of positive integers, and let $[n] := \{i \mid 1 \leq i \leq n\}$ for any $n \in \mathbb{N}$. We call $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. We fix once and for all $\mathbb{D}$ to be any infinite domain of data values; for simplicity in our examples we will consider $\mathbb{D} = \mathbb{N}_0$. 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 $\mathbb{E}$ and by $\mathbb{E}^{\omega}$ the set of infinite sequences over $\mathbb{E}$. We use '$\cdot$' as the concatenation operator between sequences. We write $|S|$ to denote the length of S (if S is a sequence), or the cardinality of S (if S is a set).

2.1. **Finite words.** We consider a finite word over $\mathbb{E}$ as a function $\mathbf{w} : [n] \to \mathbb{E}$ for some $n \in \mathbb{N}$, and we define the set of words as $Words(\mathbb{E}) := \{[n] \to \mathbb{E} \mid n \in \mathbb{N}\}$. We write $\mathsf{pos}(\mathbf{w}) = \{1, \ldots, n\}$ to denote the set of **positions** (that is, the domain of $\mathbf{w}$), and we define the size of $\mathbf{w}$ as $|\mathbf{w}| = n$. Given $\mathbf{w} \in Words(\mathbb{E})$ and $\mathbf{w}' \in Words(\mathbb{F})$ with $\mathsf{pos}(\mathbf{w}) = \mathsf{pos}(\mathbf{w}') = P$, we write $\mathbf{w} \otimes \mathbf{w}' \in Words(\mathbb{E} \times \mathbb{F})$ for the word such that $\mathsf{pos}(\mathbf{w} \otimes \mathbf{w}') = P$ and $(\mathbf{w} \otimes \mathbf{w}')(x) = (\mathbf{w}(x), \mathbf{w}'(x))$. A **data word** is a word of $Words(\mathbb{A} \times \mathbb{D})$, where $\mathbb{A}$ is a finite alphabet of letters and $\mathbb{D}$ is an infinite domain. Note that we define data words as having at least one position. This is done for simplicity of the definition of the formalisms we work with, and the results contained here also extend to possibly-empty data words and trees. We define the **word type** as a function $\mathsf{type}_{\mathbf{w}} : \mathsf{pos}(\mathbf{w}) \to \{\triangleright, \bar{\triangleright}\}$ that specifies whether a position has a next element or not. That is, $\mathsf{type}_{\mathbf{w}}(i) = \triangleright$ iff $(i+1) \in \mathsf{pos}(\mathbf{w})$.

2.2. **Unranked ordered finite trees.** We define $Trees(\mathbb{E})$, the set of finite, ordered and unranked trees over an alphabet $\mathbb{E}$. A **position** in the context of a tree is an element of $\mathbb{N}^*$. The root's position is the empty string and we note it '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. Along this work we use x, y, z, w, v as variables for positions, while i, j, k, l, m, n as variables for numbers. Thus, for example $x \cdot i$ is a position which is not the root, and that has x as parent position, and there are $i - 1$ siblings to the left of $x \cdot i$.

Formally, we define $\mathsf{POS} \subseteq \wp_{<\infty}(\mathbb{N}^*)$ the set of sets of finite tree positions, such that: $X \in \mathsf{POS}$ iff (a) $X \subseteq \mathbb{N}^*, |X| < \infty$; (b) it is prefix-closed; and (c) if $n \cdot (i+1) \in X$ for $i \in \mathbb{N}$, then $n \cdot i \in X$. A tree is a mapping from a set of positions to letters of the alphabet

$$Trees(\mathbb{E}) := \{\mathbf{t} : P \to \mathbb{E} \mid P \in \mathsf{POS}\} .$$

Given a tree $\mathbf{t} \in Trees(\mathbb{E})$, $\mathsf{pos}(\mathbf{t})$ denotes the domain of $\mathbf{t}$, which consists of the set of positions of the tree, and $\mathsf{alph}(\mathbf{t}) = \mathbb{E}$ denotes the alphabet of the tree. From now on, we informally write 'node' to denote a position x together with the value $\mathbf{t}(x)$. We define the *ancestor* partial order $\preceq$ as the prefix relation $x \preceq x \cdot y$ for every $x \cdot y$, and the strict version $\prec$ as the strict prefix relation $x \prec x \cdot y$ for $|y| > 0$. Given a tree $\mathbf{t}$ and $x \in \mathsf{pos}(\mathbf{t})$, '$\mathbf{t}|_x$' denotes the subtree of $\mathbf{t}$ at position x. That is, $\mathbf{t}|_x : \{y \mid x \cdot y \in \mathsf{pos}(\mathbf{t})\} \to \mathsf{alph}(\mathbf{t})$ where $\mathbf{t}|_x(y) = \mathbf{t}(x \cdot y)$. In the context of a tree $\mathbf{t}$, a **siblinghood** is a maximal sequence of siblings. That is, a sequence of positions $x \cdot 1, \ldots, x \cdot l \in \mathsf{pos}(\mathbf{t})$ such that $x \cdot (l+1) \notin \mathsf{pos}(\mathbf{t})$.

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

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. We define the **tree type** as a function $\mathsf{type}_{\mathbf{t}} : \mathsf{pos}(\mathbf{t}) \to \{\triangledown, \bar{\triangledown}\} \times \{\triangleright, \bar{\triangleright}\}$ that specifies whether a node has children and/or siblings to the right. That is, $\mathsf{type}_{\mathbf{t}}(x) := (a, b)$ where $a = \triangledown$ iff $x \cdot 1 \in \mathsf{pos}(\mathbf{t})$, and where $b = \triangleright$ iff $x = x' \cdot i$ and $x' \cdot (i+1) \in \mathsf{pos}(\mathbf{t})$. The notation for the set of data values used in a data tree is

$$data(\mathbf{a} \otimes \mathbf{d}) := \{\mathbf{d}(x) \mid x \in \mathsf{pos}(\mathbf{d})\} .$$

We abuse notation and write $data(X)$ to refer to all the elements of $\mathbb{D}$ contained in X, for whatever object X may be.
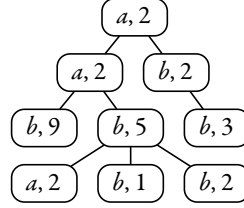
Figure 1: A data tree.

First-child and next-sibling coding. We will make use of the first-child and next-sibling underlying binary tree of an unranked tree $\mathbf{t}$. This is the binary tree whose nodes are the same as in $\mathbf{t}$, a node x is the left child of y if x is the leftmost child of y in $\mathbf{t}$, and a node x is the right child of y if x is the next sibling to the right of y in $\mathbf{t}$. We will sometimes refer to this tree by "the fcns coding of $\mathbf{t}$". Let us define the relation between positions $\preceq_{fcns}$ such that $x \preceq_{fcns} y$ if x is the ancestor of y in the first-child and next-sibling coding of the tree, that is, if y is reachable from x by traversing the tree through the operations 'go to the leftmost child' and 'go to the next sibling to the right'.

2.3. **Properties of languages.** We use the standard definition of **language**. Given an automaton $\mathcal{A}$ over data words (resp. over data trees), let $\mathcal{L}(\mathcal{A})$ denote the set of data words (resp. data trees) that have an accepting run on $\mathcal{A}$. We say that $\mathcal{L}(\mathcal{A})$ is the language of words (resp. trees) recognized by $\mathcal{A}$. We extend this definition to a class of automata $\mathscr{A}$: $\mathcal{L}(\mathscr{A}) = \{\mathcal{L}(\mathcal{A}) \mid \mathcal{A} \in \mathscr{A}\}$, obtaining a class of languages.

Equivalently, given a formula φ of a logic $\mathscr{L}$ over data words (resp. data trees) we denote by $\mathcal{L}(\varphi)$ the set of words (resp. trees) verified by φ. This is also extended to a logic $\mathcal{L}(\mathscr{L}) = \{\mathcal{L}(\varphi) \mid \varphi \in \mathscr{L}\}$.

We say that a class of automata (resp. a logic) $\mathscr{A}$ is **at least as expressive as** another class (resp. logic) $\mathscr{B}$ iff $\mathcal{L}(\mathscr{B}) \subseteq \mathcal{L}(\mathscr{A})$. If additionally $\mathcal{L}(\mathscr{B}) \neq \mathcal{L}(\mathscr{A})$ we say that $\mathscr{A}$ is **more expressive** than $\mathscr{B}$.

We say that a class of automata $\mathscr{A}$ **captures** a logic $\mathscr{L}$ iff there exists a translation $t : \mathscr{L} \to \mathscr{A}$ such that for every $\varphi \in \mathscr{L}$ and model (*i.e.*, a data tree or a data word) $\mathbf{m}$, we have that $\mathbf{m} \models \varphi$ if and only if $\mathbf{m} \in \mathcal{L}(t(\varphi))$.

2.4. **Well-structured transition systems.** The main argument for the decidability of the emptiness of the automata models studied here is based on the theory of well-quasi-orderings. We interpret the automaton's run as a transition system with some good properties, and this allows us to obtain an effective procedure for the emptiness problem. This is known in the literature as a well-structured transition system (WSTS) [see FS01, AČJT96]. Next, we reproduce some standard definitions and known results that we will make use of.

Definition 2.1. For a set S, we define $(S, \leq)$ to be a **well-quasi-order** (wqo) iff '$\leq$' $\subseteq S \times S$ is a relation that is reflexive, transitive and for every infinite sequence $w_1, w_2, \ldots \in S^\omega$ there are two indices $i < j$ such that $w_i \leq w_j$.

Dickson's Lemma ([Dic13]). *Let $\leq_k \subseteq \mathbb{N}_0^k \times \mathbb{N}_0^k$ such that $(x_1, \ldots, x_k) \leq_k (y_1, \ldots, y_k)$ iff $x_i \leq y_i$ for all $i \in [k]$. For all $k \in \mathbb{N}_0$, $(\mathbb{N}_0^k, \leq_k)$ is a well-quasi-order.*

Definition 2.2. Given a quasi-order $(S, \leq)$ we define the **embedding order** as the relation $\sqsubseteq \subseteq S^* \times S^*$ such that $x_1 \cdots x_n \sqsubseteq y_1 \cdots y_m$ iff there exist $1 \leq i_1 < \cdots < i_n \leq m$ with $x_j \leq y_{i_j}$ for all $j \in [n]$.

Higman's Lemma ([Hig52]). *Let $(S, \leq)$ be a well-quasi-order. Let $\sqsubseteq \subseteq S^* \times S^*$ be the embedding order over $(S, \leq)$. Then, $(S^*, \sqsubseteq)$ is a well-quasi-order.*

Corollary 2.3 (of Higman's Lemma). *Let S be a finite alphabet. Let $\sqsubseteq$ be the subword relation over $S^* \times S^*$ (i.e., $x \sqsubseteq y$ if x is the result of removing some (possibly none) positions from y). Then, $(S^*, \sqsubseteq)$ is a well-quasi-order.*

Proof. It suffices to realize that $\sqsubseteq$ is indeed the embedding order over $(S, =)$, which is trivially a wqo since S is finite. $\qquad\square$

Definition 2.4. Given a transition system $(S, \rightarrow)$, and $T \subseteq S$ we define $Succ(T) := \{a' \in S \mid \exists\, a \in T \text{ with } a \rightarrow a'\}$, and $Succ^*$ to its reflexive-transitive closure. Given a wqo $(S, \leq)$ and $T \subseteq S$, we define the *upward closure* of T as $\uparrow T := \{a \mid \exists\, a' \in T, a' \leq a\}$, and the *downward closure* as $\downarrow T := \{a \mid \exists\, a' \in T, a \leq a'\}$. We say that T is **downward-closed** (resp. **upward-closed**) iff $\downarrow T = T$ (resp. $\uparrow T = T$).

Definition 2.5. We say that a transition system $(S, \rightarrow)$ is **finitely branching** iff $Succ(\{a\})$ is finite for all $a \in S$. If $Succ(\{a\})$ is also computable for all a, we say that $(S, \rightarrow)$ is **effective**.

Definition 2.6 (rdc). A transition system $(S, \rightarrow)$ is **reflexive downward compatible** (or rdc for short) with respect to a wqo $(S, \leq)$ iff for every $a_1, a_2, a_1' \in S$ such that $a_1' \leq a_1$ and $a_1 \rightarrow a_2$, there exists $a_2' \in S$ such that $a_2' \leq a_2$ and either $a_1' \rightarrow a_2'$ or $a_1' = a_2'$.

Our forthcoming decidability results on alternating register automata of Sections 3.2 and 5.1 will be shown as a consequence of the following propositions.

Proposition 2.7 ([FS01, Proposition 5.4]). *If $(S, \leq)$ is a wqo and $(S, \rightarrow)$ a transition system such that (1) it is rdc, (2) it is effective, and (3) $\leq$ is decidable; then for any finite $T \subseteq S$ it is possible to compute a finite set $U \subseteq S$ such that $\uparrow U = \uparrow Succ^*(T)$.*

Lemma 2.8. *Given $(S, \leq, \rightarrow)$ as in Proposition 2.7, a recursive downward-closed set $V \subseteq S$, and a finite set $T \subseteq S$. The problem of whether there exists $a \in T$ and $b \in V$ such that $a \rightarrow^* b$ is decidable.*

Proof. Applying Proposition 2.7, let $U \subseteq S$ finite, with $\uparrow U = \uparrow Succ^*(T)$. Since T is finite and V is recursive, we can test for every element of $b \in U$ if $b \in V$. On the one hand, if there is one such b, then by definition $b \in \uparrow Succ^*(T)$, or in other words $a \rightarrow^* a' \leq b$ for some $a \in T$, $a' \in Succ^*(a)$. But since V is downward-closed, $a' \in V$ and hence the property is true. On the other hand, if there is no such b in U, it means that there is no such b in $\uparrow U$ either, as V is downward-closed. This means that there is no such b in $Succ^*(T)$ and hence that the property is false. $\qquad\square$

Definition 2.9 ($\leq_\wp$). Given an ordering $(S, \leq)$, we define the **majoring ordering** over $\leq$ as $(\wp_{<\infty}(S), \leq_\wp)$, where $\mathcal{S} \leq_\wp \mathcal{S}'$ iff there exists an injective function $f : \mathcal{S} \rightarrow \mathcal{S}'$ so that for every $a \in \mathcal{S}$, $a \leq f(a)$.

Proposition 2.10. *If $(S, \leq)$ is a wqo, then the majoring order over $(S, \leq)$ is a wqo.*

Proof. The fact that this order is reflexive and transitive is immediate from the fact that $\leq$ is a quasi-order. The fact of being a *well*-quasi-order is a simple consequence of Higman's Lemma. Each finite set $\{a_1, \ldots, a_n\}$ can be seen as a sequence of elements $a_1, \ldots, a_n$, in any order. In this context, the embedding order is stricter than the majoring order. In other words, if $a_1, \ldots, a_n \sqsubseteq a'_1, \ldots, a'_m$, then $\{a_1, \ldots, a_n\} \leq_\wp \{a'_1, \ldots, a'_m\}$. By Higman's Lemma the embedding order over $(S, \leq)$ is a wqo, implying that the majoring order is as well. $\square$

The following technical Proposition will become useful in Section 5.1 for transferring the decidability results obtained for class of automata $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread})$ on data words to the class $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ on data trees.

Proposition 2.11. *Let* $\leq, \rightarrow_1^1, \rightarrow_1^2, \rightarrow_1^3 \subseteq S \times S$, *let* $\mathcal{P} \subseteq S$ *be downward- and upward-closed set with respect to* $\leq$, *let* $\leq_\wp, \rightarrow_2 \subseteq \wp_{<\infty}(S) \times \wp_{<\infty}(S)$ *where* $\leq_\wp$ *is the* majoring order *over* $(S, \leq)$ *so that* $\mathcal{S} \rightarrow_2 \mathcal{S}'$ *iff either*

- $\mathcal{S} = \{a\} \cup \hat{\mathcal{S}}$, $\mathcal{S}' = \{b_1, b_2\} \cup \hat{\mathcal{S}}$, $a \in \mathcal{P}$, *and* $a \rightarrow_1^i b_i$ *for all* $i \in \{1, 2\}$, *or*
- $\mathcal{S} = \{a\} \cup \hat{\mathcal{S}}$, $\mathcal{S}' = \{b\} \cup \hat{\mathcal{S}}$, $a \notin \mathcal{P}$, *and* $a \rightarrow_1^i b$ *for some* $i \in \{1, 2\}$, *or*
- $\mathcal{S} = \{a\} \cup \hat{\mathcal{S}}$, $\mathcal{S}' = \{b\} \cup \hat{\mathcal{S}}$, *and* $a \rightarrow_1^3 b$.

Suppose that $(S, \leq)$ *is a* wqo *which is* rdc *with respect to* $\rightarrow_1^i$, *for every* $i \in \{1, 2, 3\}$. *Then,* $(\wp_{<\infty}(S), \leq_\wp)$ *is a* wqo *which is* rdc *with respect to* $\rightarrow_2$.

Proof. The fact that $(\wp_{<\infty}(S), \leq_\wp)$ is a wqo is given by Proposition 2.10. The rdc property with respect to $\rightarrow_2$ is simple. Suppose that $a \in \mathcal{P}$ and

$$\mathcal{S}' \quad \leq_\wp \quad \{a\} \cup \mathcal{S} \quad \rightarrow_2 \quad \{b_1, b_2\} \cup \mathcal{S}$$

with $a \rightarrow_1^i b_i$ for all $i \in \{1, 2\}$. Let f be the witnessing function for $\mathcal{S}' \leq_\wp \mathcal{S}$. If $f^{-1}(a)$ does not exist, then $\mathcal{S}' \leq_\wp \mathcal{S}$, and the relation is reflexive compatible since this means that $\mathcal{S}' \leq_\wp \{b_1, b_2\} \cup \mathcal{S}$. Otherwise, let $a' = f^{-1}(a)$ and note that $\mathcal{S}' \setminus \{a'\} \leq_\wp \mathcal{S}$. Since $a \in \mathcal{P}$ and $\mathcal{P}$ is downward-closed, then $a' \in \mathcal{P}$. Further, since $a' \leq a \rightarrow_1^i b_i$ we obtain, by rdc of $\rightarrow_1^i$, that either

- $a' \leq b_1$ or $a' \leq b_2$, in which case $\mathcal{S}' \leq_\wp \{b_1, b_2\} \cup \mathcal{S}$, or
- there are b'_1, b'_2 so that $a' \rightarrow_1^i b'_i$ and $b'_i \leq b_i$ for all $i \in \{1, 2\}$, in which case $\mathcal{S}' \rightarrow_2 \mathcal{S}' \setminus \{a'\} \cup \{b'_1, b'_2\} \leq_\wp \{b_1, b_2\} \cup \mathcal{S}$.

The second possiblity—where $a \notin \mathcal{P}$ and we have $\mathcal{S}' \leq_\wp \{a\} \cup \mathcal{S} \rightarrow_2 \{b\} \cup \mathcal{S}$—is analogous, this time using the fact that $\mathcal{P}$ is upward-closed. The third possibility is also easy. $\square$

Part 1. Data words

In this first part, we start our study on data words. In Section 3 we introduce our automata model $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread})$, and we show that the emptiness problem for these automata is decidable. This extends the results of [DL09]. To prove decidability, we adopt a different approach than the one of [DL09] that enables us to show the decidability of some extensions, and to simplify the decidability proofs of Part 2, which can be seen as a corollary of this proof. In Section 4 we introduce a temporal logic with registers and quantification over data values. This logic is shown to have a decidable satisfiability problem by a reduction to the emptiness problem of $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread})$ automata.

3. ARA model

An **Alternating Register Automaton** (ARA) consists in a one-way automaton on data words with alternation and *one* register to store and test data. In [DL09] it was shown that the emptiness problem is decidable and non-primitive-recursive. Here, we consider an extension of ARA with two operators: spread and guess. We call this model ARA(spread, guess).

Definition 3.1. An alternating register automaton of ARA(spread, guess) is a tuple $\mathcal{A} = \langle \mathbb{A}, Q, q_I, \delta \rangle$ such that

- $\mathbb{A}$ is a finite alphabet;
- Q is a finite set of states;
- $q_I \in Q$ is the initial state; and
- $\delta : Q \to \Phi$ is the transition function, where Φ is defined by the grammar

$$a \mid \bar{a} \mid \odot? \mid \mathsf{store}(q) \mid \mathsf{eq} \mid \overline{\mathsf{eq}} \mid q \wedge q' \mid q \vee q' \mid \rhd q \mid \mathsf{guess}(q) \mid \mathsf{spread}(q, q')$$

where $a \in \mathbb{A}, q, q' \in Q, \odot \in \{\rhd, \bar{\rhd}\}$.

This formalism without the guess and spread transitions is equivalent to the automata model of [DL09] on finite data words, where $\rhd$ is to move to the next position to the right on the data word, $\mathsf{store}(q)$ stores the current datum in the register and eq (resp. $\overline{\mathsf{eq}}$) tests that the current node's value is (resp. is not) equal to the stored. We call a state $q \in Q$ **moving** if $\delta(q) = \rhd q'$ for some $q' \in Q$.

As this automaton is one-way, we define its semantics as a set of 'threads' for each node that progress synchronously. That is, all threads at a node move one step forward simultaneously and then perform some non-moving transitions independently. This is done for the sake of simplicity of the formalism, which simplifies both the presentation and the decidability proof.

Next we define a *configuration* and then we give a notion of a *run* over a data word $\mathbf{w}$. A **configuration** is a tuple $\langle i, \alpha, \gamma, \Delta \rangle$ that describes the partial state of the execution at position i. The number $i \in \mathsf{pos}(\mathbf{w})$ is the *position* in the data word $\mathbf{w}$, $\gamma = \mathbf{w}(i) \in \mathbb{A} \times \mathbb{D}$ is the current position's letter and datum, and $\alpha = \mathsf{type}_{\mathbf{w}}(i)$ is the *word type* of the position i. Finally, $\Delta \in \wp_{<\infty}(Q \times \mathbb{D})$ is a finite set of active **threads**, each thread (q, d) consisting in a state q and the value d stored in the register. We will always note the set of threads of a configuration with the symbol Δ, and we write $\Delta(d) = \{q \in Q \mid (q, d) \in \Delta\}$ for $d \in \mathbb{D}$, $\Delta(q) = \{d \in \mathbb{D} \mid (q, d) \in \Delta\}$ for $q \in Q$. By $\mathcal{C}_{\mathrm{ARA}}$ we denote the set of all configurations. Given a set of threads Δ we write $data(\Delta) := \{d \mid (q, d) \in \Delta\}$, and $data(\langle i, \alpha, (a, d), \Delta \rangle) := \{d\} \cup data(\Delta)$. We say that a configuration is **moving** if for every $(q, d) \in \Delta$, q is moving.

To define a run we first introduce three transition relations over *node configurations*: the *non-moving* relation $\to_\varepsilon$ and the *moving* relation $\to_\rhd$. We start with $\to_\varepsilon$. If the transition corresponding to a thread is a $\mathsf{store}(q)$, the automaton *sets* the register with current data value and continues the execution of the thread with state q; if it is eq, the thread *accepts* (and in this case disappears from the configuration) if the current datum is *equal* to that of the register, otherwise the computation for that thread cannot continue. The reader can check that the rest of the cases defined in Figure 2 follow the intuition of an alternating automaton.

The cases that follow correspond to our extensions to the model of [DL09]. The guess instruction extends the model with the ability of storing *any* datum from the domain $\mathbb{D}$.

$$\rho \to_\varepsilon \langle i, \alpha, (a,d), \{(q_j, d')\} \cup \Delta \rangle \qquad \text{if } \delta(q) = q_1 \vee q_2, j \in \{1,2\} \qquad (3.1)$$

$$\rho \to_\varepsilon \langle i, \alpha, (a,d), \{(q_1, d'), (q_2, d')\} \cup \Delta \rangle \qquad \text{if } \delta(q) = q_1 \wedge q_2 \qquad (3.2)$$

$$\rho \to_\varepsilon \langle i, \alpha, (a,d), \{(q', d)\} \cup \Delta \rangle \qquad \text{if } \delta(q) = \mathsf{store}(q') \qquad (3.3)$$

$$\rho \to_\varepsilon \langle i, \alpha, (a,d), \Delta \rangle \qquad \text{if } \delta(q) = \mathsf{eq} \text{ and } d = d' \qquad (3.4)$$

$$\rho \to_\varepsilon \langle i, \alpha, (a,d), \Delta \rangle \qquad \text{if } \delta(q) = \overline{\mathsf{eq}} \text{ and } d \neq d' \qquad (3.5)$$

$$\rho \to_\varepsilon \langle i, \alpha, (a,d), \Delta \rangle \qquad \text{if } \delta(q) = \alpha? \qquad (3.6)$$

$$\rho \to_\varepsilon \langle i, \alpha, (a,d), \Delta \rangle \qquad \text{if } \delta(q) = b \text{ and } b = a \qquad (3.7)$$

$$\rho \to_\varepsilon \langle i, \alpha, (a,d), \Delta \rangle \qquad \text{if } \delta(q) = \bar{b} \text{ and } b \neq a \qquad (3.8)$$

Figure 2: Definition of the transition relation $\to_\varepsilon \subseteq \mathcal{C}_{\mathrm{ARA}} \times \mathcal{C}_{\mathrm{ARA}}$, given a configuration $\rho = \langle i, \alpha, (a,d), \{(q, d')\} \cup \Delta \rangle$.

Whenever $\delta(q) = \mathsf{guess}(q')$ is executed, a data value (nondeterministically chosen) is saved in the register.

$$\rho \to_\varepsilon \langle i, \alpha, (a,d), \{(q', e)\} \cup \Delta \rangle \quad \text{if } \delta(q) = \mathsf{guess}(q'), e \in \mathbb{D} \qquad (3.9)$$

Note that the store instruction may be simulated with the guess, eq and $\wedge$ instructions, while guess cannot be expressed by the ARA model.

The 'spread' instruction is an unconventional operator in the sense that it depends on the data of *all* threads in the current configuration with a certain state. Whenever $\delta(q) = \mathsf{spread}(q_2, q_1)$ is executed, a new thread with state q_1 and datum d is created for each thread $\langle q_2, d \rangle$ present in the configuration. With this operator we can code a universal quantification over all the data values that appeared so far (*i.e.*, that appeared in smaller positions). We demand that this transition may only be applied if all other possible $\to_\varepsilon$ kind of transitions were already executed. In other words, only spread transitions or *moving* transitions are present in the configuration.

$$\rho \to_\varepsilon \langle i, \alpha, (a,d), \{\langle q_1, d \rangle \mid \langle q_2, d \rangle \in \Delta \} \cup \Delta \rangle \qquad (3.10)$$

iff $\delta(q) = \mathsf{spread}(q_2, q_1)$ and for all $\langle \tilde{q}, \tilde{d} \rangle \in \Delta$ either $\delta(\tilde{q})$ is a spread or a moving instruction. We also use a weaker one-argument version of spread.

$$\rho \to_\varepsilon \langle i, \alpha, (a,d), \{\langle q_1, d \rangle \mid \exists q_2. \langle q_2, d \rangle \in \Delta \} \cup \Delta \rangle \qquad (3.11)$$

iff $\delta(q) = \mathsf{spread}(q_1)$ and for all $\langle \tilde{q}, \tilde{d} \rangle \in \Delta$ either $\delta(\tilde{q})$ is a spread or a moving instruction. Notice that the one-argument version of spread can be simulated with the two-argument version, and hence we do not include it in the definition of the automaton. Also, note that we enforce the spread operation to be executed once all other non-moving transitions have been applied, in order to take into account all the data values that may have been introduced in these transitions, as a result of guess operations. This behavior simplifies the reduction from the satisfiability of forward-XPath we will show later. This reduction will only need to use the weak one-argument version of spread.

The $\to_{\rhd}$ transition advances all threads of the node simultaneously, and is defined, for any type $\alpha' \in \{\rhd, \bar{\rhd}\}$ and symbol and with data value $\gamma' \in \mathbb{A} \times \mathbb{D}$,

$$\langle i, \rhd, \gamma, \Delta \rangle \to_{\rhd} \langle i+1, \alpha', \gamma', \Delta_{\rhd} \rangle \qquad (3.12)$$

iff
  (i) for all $(q, d) \in \Delta$, $\delta(q)$ is moving; and
  (ii) $\Delta_{\triangleright} = \{\langle q', d \rangle \mid (q, d) \in \Delta, \delta(q) = \triangleright q'\}$.

  Finally, we define the transition between configurations as $\rightarrowtail \;:=\; \rightarrow_{\triangleright} \cup \rightarrow_{\varepsilon}$.

  A *run* over a data word $\mathbf{w} = \mathbf{a} \otimes \mathbf{d}$ is a nonempty sequence $\mathcal{C}_1 \rightarrowtail \cdots \rightarrowtail \mathcal{C}_n$ with $\mathcal{C}_1 = \langle 1, \alpha_0, \gamma_0, \Delta_0 \rangle$ and $\Delta_0 = \{\langle q_I, \mathbf{d}(1) \rangle\}$ (*i.e.*, the thread consisting in the initial state with the first datum), such that for every $j \in [n]$ with $\mathcal{C}_j = \langle i, \alpha, \gamma, \Delta \rangle$: (1) $i \in \mathsf{pos}(\mathbf{w})$; (2) $\gamma = \mathbf{w}(i)$; and (3) $\alpha = \mathsf{type}_{\mathbf{w}}(i)$. We say that the run is *accepting* iff $\mathcal{C}_n = \langle i, \alpha, \gamma, \emptyset \rangle$ contains an empty set of threads. If for an automaton $\mathcal{A}$ we have that $\mathcal{L}(\mathcal{A}) \neq \emptyset$ we say that $\mathcal{A}$ is *nonempty*.

3.1. **Properties.** We show the following two statements
  - $\mathcal{L}(\mathsf{ARA}(\mathsf{guess}, \mathsf{spread}))$ is not closed under complementation,
  - the $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread})$ class is more expressive than ARA.
In fact in the proof below we show the first one, that implies the second one, given the fact that the ARA model is closed under complementation.

Proposition 3.2 (Expressive power)**.**
  (a) *the* $\mathsf{ARA}(\mathsf{guess})$ *class is more expressive than* ARA*;*
  (b) *the* $\mathsf{ARA}(\mathsf{spread})$ *class is more expressive than* ARA*.*

Proof. Let $\mathbf{w} = \mathbf{a} \otimes \mathbf{d}$ be a data word. To prove (a), consider the property "*There exists a datum d and a position i with $\mathbf{d}(i) = d$, $\mathbf{a}(i) = a$, and there is no position $j \leq i$ with $\mathbf{d}(j) = d$, $\mathbf{a}(j) = b$.*". This property can be easily expressed by $\mathsf{ARA}(\mathsf{guess})$. It suffices to guess the data value d and checks that we can reach an element (a, d), and that for every previous element (b, d') we have that $d \neq d'$. We argue that this property cannot be expressed by the ARA model. Suppose ad absurdum that it is expressible. This means that its negation would also be expressible by ARA (since they are closed under complementation). The negation of this property states

$$\text{``For every data value } d\text{, if there is an element } (a, d)$$
$$\text{in the word, then there is a previous element } (b, d)\text{.''} \tag{P1}$$

With this kind of property one can code an accepting run of a Minsky machine, whose emptiness problem is undecidable. This would prove that $\mathsf{ARA}(\mathsf{guess})$ have an undecidable emptiness problem, which is in contradiction with the decidability proof that we will give in Section 3.2. Let us see how the reduction works.

  The emptiness problem for Minsky machine is known to be undecidable even with an alphabet consisting of one symbol, so we disregard the letters read by the automaton in the following description. Consider then a 2-counter alphabet-blind Minsky machine whose instructions are of the form (q, ℓ, q') with $\ell \in \{\mathsf{inc}, \mathsf{dec}, \mathsf{ifzero}\} \times \{1, 2\}$ being the operation over the counters, and q, q' states from the automaton's set of states Q. A run on this automaton is a sequence of applications of transition rules, for example

$$(q_1, \mathsf{inc}_1, q_2)\,(q_2, \mathsf{inc}_2, q_3)\,(q_3, \mathsf{inc}_1, q_2)\,(q_2, \mathsf{dec}_1, q_1)\,(q_1, \mathsf{dec}_1, q_2)\,(q_2, \mathsf{ifzero}_1, q_3)$$

This run has an associated data word over the alphabet

$$Q \times \{\mathsf{inc}, \mathsf{dec}, \mathsf{ifzero}\} \times \{1, 2\} \times Q,$$
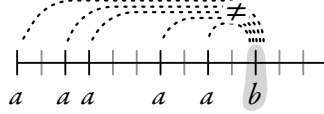
Figure 3: A property not expressible in ARA.

where the corresponding data value of each instruction is used to match increments with decrements, for example,

$$(q_1, \mathsf{inc}_1, q_2)\,(q_2, \mathsf{inc}_2, q_3)\,(q_3, \mathsf{inc}_1, q_2)\,(q_2, \mathsf{dec}_1, q_1)\,(q_1, \mathsf{dec}_1, q_2)\,(q_2, \mathsf{ifzero}_1, q_3).$$
$$123214$$

Using the ARA model we can make sure that (i) all increments have different data values and all decrements have different data values; and (ii) for every (, inc_i,) element with data value d that occurs to the left of a (, ifzero_i,), there must be a (, dec_i,) element with data value d that occurs in between. [DL09] shows how to express these properties using ARA. However, properties (i) and (ii) are not enough to make sure that every prefix of the run ending in a ifzero_i instruction must have as many increments as decrements of counter i. Indeed, there could be more decrements than increments — but not the opposite, thanks to (ii).

The missing condition to verify that the run is correct is: (iii) for every decrement there exists a previous increment with the same data value. In fact, we can see that property (P1) can express condition (iii): we only need to change a by a decrement transition in the coding, and b by an increment transition of the same counter. But then, assuming that property (P1) can be expressed by ARA(guess), the emptiness problem for ARA(guess) is undecidable. This is absurd, as the emptiness problem is decidable, as we will show later on in Theorem 3.5.

Using a similar reasoning as before, we show (b): the ARA(spread) class is more expressive than ARA. Consider the property: "*There exists a position i labeled b such that $\mathbf{d}(i) \neq \mathbf{d}(j)$ for all $j < i$ with $\mathbf{a}(j) = a$.*" as depicted in Figure 3. Let us see how this can be coded into ARA(spread). Assuming q_0 is the initial state, the transitions should reflect that every datum with label a seen along the run is saved with a state q_a, and that this state is in charge of propagating this datum. Then, we guess a position labeled with b and check that all these stored values under q_a are different from the current one. For succinctness we write the transitions as positive boolean combinations of the basic operations.

$$\delta(q_0) = (b \wedge \mathsf{spread}(q_a, q_1)) \ \vee \ \big((\bar{a} \vee \mathsf{store}(q_a)) \wedge \triangleright q_0\big),$$
$$\delta(q_1) = \overline{\mathsf{eq}}, \qquad \delta(q_a) = (\bar{\triangleright}? \vee \triangleright q_a).$$

This property cannot be expressed by the ARA model. Were it expressible, then its negation

$$\text{"\textit{for every element } b \textit{ there exists a previous one}}$$
$$\textit{labeled } a \textit{ with the same data value"} \qquad \text{(P2)}$$

would also be. Just as before we can use property (P2) to express condition (iii), and force that for every decrement in a coding of a Minsky machine there exists a corresponding previous increment. This leads to a contradiction by proving that the emptiness problem for ARA(spread) is undecidable. □

Corollary 3.3. ARA(guess), ARA(spread) *and* ARA(guess, spread) *are not closed under complementation.*

Proof. If they were closed under complementation, then we could express some of the properties described in the proof of Proposition 3.2, resulting in an undecidable model, which is in contradiction with Theorem 3.5. □

We then have the following properties of the automata model.

Proposition 3.4 (Boolean operations)**.** *The class* $\mathcal{L}(\mathsf{ARA}(\mathsf{spread}, \mathsf{guess}))$ *has the following properties:*

   (i) it is closed under union,
  (ii) it is closed under intersection,
 (iii) it is not closed under complementation.

Proof sketch. Items (i) and (ii) are straightforward if we notice that the first argument of spread ensures that this transition is always relative to the states of one of the automata being under intersection or union. Item (iii) follows from Corollary 3.3. □

3.2. **Emptiness problem.** This section is dedicated to show the following theorem.

Theorem 3.5. *The emptiness problem for* ARA(guess, spread) *is decidable.*

As already mentioned, decidability for ARA was proved in [DL09]. Here we propose an alternative approach that simplifies the proof of decidability of the two extensions spread and guess.

The proof goes as follows. We will define a wqo $\precsim$ over $\mathcal{C}_{\mathrm{ARA}}$ and show that $(\mathcal{C}_{\mathrm{ARA}}, \precsim)$ is rdc with respect to $\rightarrowtail$ (Lemma 3.10). Note that strictly speaking $\rightarrowtail$ is an infinite-branching transition system as $\rightarrow_{\triangleright}$ may take *any* value from the infinite set $\mathbb{D}$, and $\rightarrow_{\varepsilon}$ can also guess any value. However, it can trivially be restricted to an effective *finitely* branching one. Then, by Lemma 2.8, $(\mathcal{C}_{\mathrm{ARA}}, \rightarrowtail)$ has a computable upward-closed reachability set, and this implies that the emptiness problem of ARA(guess, spread) is decidable.

Since our model of automata only cares about equality or inequality of data values, it is convenient to work modulo renaming of data values.

Definition 3.6 ($\sim$)**.** We say that two configurations $\rho, \rho' \in \mathcal{C}_{\mathrm{ARA}}$ are **equivalent** (notation $\rho \sim \rho'$) if there is a bijection $f : data(\rho) \rightarrow data(\rho')$ such that $f(\rho) = \rho'$, where $f(\rho)$ stands for the replacement of every data value d by $f(d)$ in ρ.

Definition 3.7 ($\preceq$)**.** We first define the relation $(\mathcal{C}_{\mathrm{ARA}}, \preceq)$ such that

$$\langle i, \alpha, \gamma, \Delta \rangle \ \preceq \ \langle i', \alpha', \gamma', \Delta' \rangle$$

iff $\alpha = \alpha'$, $\gamma = \gamma'$, and $\Delta \subseteq \Delta'$.

Notice that by the definition above we are 'abstracting away' the information concerning the position i. We finally define $\precsim$ to be $\preceq$ modulo $\sim$.

Definition 3.8 ($\precsim$)**.** We define $\rho \precsim \rho'$ iff there is $\rho'' \sim \rho'$ with $\rho \preceq \rho''$.

The following lemma follows from the definitions.

Lemma 3.9. $(\mathcal{C}_{\mathrm{ARA}}, \precsim)$ *is a well-quasi-order.*

Proof. The fact that $\precsim$ is a *quasi-order* (*i.e.*, reflexive and transitive) is immediate from its definition. To show that it is a *well*-quasi-order, suppose we have an infinite sequence of configurations $\rho_1\rho_2\rho_3\cdots$. It is easy to see that it contains an infinite subsequence $\tau_1\tau_2\tau_3\cdots$ such that all its elements are of the form $\langle i, \alpha_0, (a_0, d), \Delta \rangle$ with

- α_0 and a_0 fixed, and
- $\Delta(d) = \mathcal{C}_0$ fixed,

This is because we can see each of these elements as a finite *coloring*, and apply the pigeonhole principle on the infinite set $\{\rho_i\}_i$.

Consider then the function $g_\Delta : \wp(Q) \to \mathbb{N}_0$, such that $g_\Delta(\mathcal{S}) = |\{d \mid \mathcal{S} = \Delta(d)\}|$ (we can think of g_Δ as a tuple of $(\mathbb{N}_0)^{\wp(Q)}$). Assume the relation $\precsim^\dagger$ defined as $\Delta \precsim^\dagger \Delta'$ iff $g_\Delta(\mathcal{S}) \leq g_{\Delta'}(\mathcal{S})$ for all $\mathcal{S}$. By Dickson's Lemma $\precsim^\dagger$ is a wqo, and then there are two $\tau_i = \langle i', \alpha_0, (a_0, d_i), \Delta_i \rangle$, $\tau_j = \langle j', \alpha_0, (a_0, d_j), \Delta_j \rangle$, $i < j$ such that $\Delta_i \precsim^\dagger \Delta_j$. For each $\mathcal{S} \subseteq Q$, there exists an injective mapping $f_\mathcal{S} : \{d \mid \Delta_i(d) = \mathcal{S}\} \to \{d \mid \Delta_j(d) = \mathcal{S}\}$ such that $f_{\mathcal{C}_0}(d_i) = d_j$, as the latter set is bigger than the former by $\precsim^\dagger$. We define the injection $f : data(\tau_i) \to data(\tau_j)$ as the (disjoint) union of all $f_\mathcal{S}$'s. The union is disjoint since for every data value d and set of threads Δ, there is a unique set $\mathcal{S}$ such that $d \in \{d' \mid S = \Delta(d')\}$. We then have that $\tau_i \sim f(\tau_i) \preceq \tau_j$. Hence, $\tau_i \precsim \tau_j$. $\square$

The core of this proof is centered in the following lemma.

Lemma 3.10. $(\mathcal{C}_{\mathrm{ARA}}, \to_\varepsilon)$ *and* $(\mathcal{C}_{\mathrm{ARA}}, \to_\triangleright)$ *(and therefore* $(\mathcal{C}_{\mathrm{ARA}}, \rightarrowtail)$*) are* rdc *with respect to* $(\mathcal{C}_{\mathrm{ARA}}, \precsim)$.

Proof. We shall show that for all $\rho, \tau, \rho' \in \mathcal{C}_{\mathrm{ARA}}$ such that $\rho \rightarrowtail \tau$ and $\rho' \precsim \rho$, there is τ' such that $\tau' \precsim \tau$ and either $\rho' \rightarrowtail \tau'$ or $\tau' = \rho'$. Since by definition of $\precsim$ we work modulo $\sim$, we can further assume that $\rho' \preceq \rho$ without any loss of generality. The proof is a simple case analysis of the definitions for $\rightarrowtail$. All cases are treated alike, here we present the most representative. Suppose first that $\rho \to_\varepsilon \tau$, then one of the definition conditions of $\to_\varepsilon$ must apply.

If Eq. (3.4) of the definition of $\to_\varepsilon$ (Fig. 2) applies, let

$$\rho = \langle i, \alpha, (a, d), \{(q, d)\} \cup \Delta \rangle \to_\varepsilon \tau = \langle i, \alpha, (a, d), \Delta \rangle$$

with $\delta(q) = \mathsf{eq}$. Let $\rho' = \langle i', \alpha, (a, d), \Delta' \rangle \preceq \rho$. If $(q, d) \in \Delta'$, we can then apply the same $\to_\varepsilon$-transition obtaining $\rho \succeq \rho' \to_\varepsilon \tau' \preceq \tau$. If there is no such (q, d), we can safely take $\rho' = \tau'$ and check that $\tau' \preceq \tau$.

If Eq. (3.3) applies, let

$$\rho = \langle i, \alpha, (a, d), \{(q, d')\} \cup \Delta \rangle \to_\varepsilon \tau = \langle i, \alpha, (a, d), \{(q', d)\} \cup \Delta \rangle$$

with $\rho \to_\varepsilon \tau$ and $\delta(q) = \mathsf{store}(q')$. Again let $\rho' \preceq \rho$ containing $(q, d') \in \Delta'$. In this case we can apply the same $\to_\varepsilon$-transition arriving to τ' where $\tau' \preceq \tau$. Otherwise, if $(q, d') \notin \Delta'$, we take $\rho' = \tau'$ and then $\tau' \preceq \tau$.

If a guess is performed (Eq. (3.9)), let

$$\rho = \langle i, \alpha, (a, d), \{(q, d')\} \cup \Delta \rangle \to_\varepsilon \tau = \langle i, \alpha, (a, d), \{(q', e)\} \cup \Delta \rangle$$

with $\delta(q) = \mathsf{guess}(q')$. Let $\rho' = \langle i', \alpha, (a, d), \Delta' \rangle \preceq \rho$. Suppose there is $(q, d') \in \Delta'$, then we then take a guess transition from ρ' obtaining some τ' by guessing e and hence $\tau' \preceq \tau$. Otherwise, if $(q, d') \notin \Delta'$, we take $\tau' = \rho'$ and check that $\tau' \preceq \tau$.

Finally, if a spread is performed (Eq. (3.10)), let

$$\rho = \langle i, \alpha, \gamma, \{(q, d')\} \cup \Delta \rangle \rightarrow_\varepsilon \tau = \langle i, \alpha, \gamma, \{(q_1, d) \mid (q_2, d) \in \Delta\} \cup \Delta \rangle$$

with $\delta(q) = \mathsf{spread}(q_2, q_1)$. Let $\rho' = \langle i', \alpha, \gamma, \Delta' \rangle \preceq \rho$ and suppose there is $(q, d') \in \Delta'$ (otherwise $\tau' = \rho'$ works). We then take a spread instruction $\rho' \rightarrow_\varepsilon \tau'$ and see that $\tau' \preceq \tau$, because any (q_1, e) in τ' generated by the spread must come from (q_2, e) of ρ', and hence there is some (q_2, e) in ρ; now by the spread applied on ρ, (q_1, d') is in τ.

The remaining cases of $\rightarrow_\varepsilon$ are only easier.

Finally, there can be a 'moving' application of $\rightarrowtail$. Suppose that we have

$$\rho = \langle i, \rhd, (a, d), \Delta \rangle \rightarrow_\rhd \tau = \langle i + 1, \alpha_1, (a_1, d_1), \Delta_1 \rangle.$$

Let $\rho' = \langle i', \rhd, (a, d), \Delta' \rangle \preceq \rho$. If ρ' is such that $\rho' \preceq \tau$, the relation is trivially compatible. Otherwise, we shall prove that there is τ' such that $\rho' \rightarrowtail \tau'$ and $\tau' \precsim \tau$. Condition i of $\rightarrow_\rhd$ (*i.e.*, that all states are moving) holds for ρ', because all the states present in ρ' are also in ρ (by definition of $\preceq$) where the condition must hold. Then, we can apply the $\rightarrow_\rhd$ transition to ρ' and obtain τ' of the form $\langle i' + 1, \alpha_1, (a_1, d_1), \Delta_1' \rangle$. Notice that we are taking α_1, a_1 and d_1 exactly as in τ, and that Δ_1' is completely determined by the $\rightarrow_\rhd$ transition from Δ'. We only need to check that $\tau' \preceq \tau$. Take any $(q, d') \in \Delta_1'$. There must be some $(q', d') \in \Delta'$ with $\delta(q') = \rhd q$. Since $\Delta' \subseteq \Delta$, we also have $(q, d) \in \Delta_1$. Hence, $\Delta_1' \subseteq \Delta_1$ and then $\tau' \preceq \tau$. $\square$

We just showed that $(\mathcal{C}_{\mathrm{ARA}}, \rightarrowtail)$ is rdc with respect to $(\mathcal{C}_{\mathrm{ARA}}, \precsim)$. The only missing ingredient to have decidability is the following, which is trivial.

Lemma 3.11. *The set of accepting configurations of $\mathcal{C}_{\mathrm{ARA}}$ is downward closed with respect to $\precsim$.*

We write $\mathcal{C}_{\mathrm{ARA}}/\sim$ to the set of configurations modulo $\sim$, by keeping one representative for every equivalence class. Note that the transition system $(\mathcal{C}_{\mathrm{ARA}}/\sim, \rightarrowtail)$ is effective. This is just a consequence of the fact that the $\rightarrowtail$-image of any configuration has only a finite number of configurations modulo $\sim$, and representatives for every class are computable. Hence, we have that $(\mathcal{C}_{\mathrm{ARA}}/\sim, \precsim, \rightarrowtail)$ verify conditions (1) and (2) from Proposition 2.7. Finally, condition (3) holds since $(\mathcal{C}_{\mathrm{ARA}}, \precsim)$ is computable. We can then apply Lemma 2.8, obtaining that for a given $\mathcal{A} \in \mathsf{ARA}(\mathsf{guess}, \mathsf{spread})$, testing wether there exists a final configuration τ and an element ρ in

$$\{\langle 1, \alpha, (a, d_0), \{(q_I, d_0)\}\rangle \mid \alpha \in \{\rhd, \bar{\rhd}\}, a \in \mathbb{A}\}$$

—for any fixed d_0— such that $\rho \sim \rho' \rightarrowtail^* \tau$ (for some ρ') is decidable. Thus, we can decide the emptiness problem and Theorem 3.5 follows.

3.3. **Ordered data.** We show here that the previous decidability result holds even if we add *order* to the data domain. Let $(\mathbb{D}, <)$ be a linear order, like for example the reals or the natural numbers with the standard ordering. Let us replace the instructions eq and $\overline{\mathsf{eq}}$ with

$$\delta(q) := \ \ldots \ \mid \ \mathsf{test}(>) \ \mid \ \mathsf{test}(<) \ \mid \ \mathsf{test}(=) \ \mid \ \mathsf{test}(\neq)$$

and let us call this model of automata $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread}, <)$. The semantics is as expected. $\mathsf{test}(<)$ verifies that the data value of the current position is less than the data value in the

register, test($>$) that is greater, and test($=$) (resp. test($\neq$)) that both are (resp. are not) equal. We modify accordingly $\to_\varepsilon$, for $\rho = \langle i, \alpha, (a,d), \{(q,d')\} \cup \Delta \rangle$.

$$\rho \to_\varepsilon \langle i, \alpha, (a,d), \Delta \rangle \quad \text{if } \delta(q) = \text{test}(<) \text{ and } d < d' \tag{3.13}$$

$$\rho \to_\varepsilon \langle i, \alpha, (a,d), \Delta \rangle \quad \text{if } \delta(q) = \text{test}(>) \text{ and } d > d' \tag{3.14}$$

$$\rho \to_\varepsilon \langle i, \alpha, (a,d), \Delta \rangle \quad \text{if } \delta(q) = \text{test}(=) \text{ and } d = d' \tag{3.15}$$

$$\rho \to_\varepsilon \langle i, \alpha, (a,d), \Delta \rangle \quad \text{if } \delta(q) = \text{test}(\neq) \text{ and } d \neq d' \tag{3.16}$$

All the remaining definitions are preserved. We can show that the emptiness problem for this extended model of automata is still decidable.

Theorem 3.12. *The emptiness problem for* ARA(guess, spread, $<$) *is decidable.*

As in the proof in the previous Section 3.2, we show that there is a wqo $\ll \subseteq \mathcal{C}_{\text{ARA}} \times \mathcal{C}_{\text{ARA}}$ that is rdc with respect to $\rightarrowtail$, such that the set of final states is $\ll$-downward closed. However, we need to be more careful when showing that we can always work modulo an equivalence relation.

Definition 3.13. A function $f : D \to \mathbb{D}$ is an **order-preserving bijection** on $D \subseteq \mathbb{D}$ iff it is injective, and furthermore for every $\{d, d'\} \subseteq D$, if $d < d'$ then $f(d) < f(d')$.

The following Lemma is straightforward from the definition just seen.

Lemma 3.14. *Let $D \subseteq \mathbb{D}$, $|D| < \infty$. There exists an order-preserving bijection f on D such that*

- *for every $\{d, d'\} \subseteq D$ such that $d < d'$ there exists $\tilde{d}$ such that $f(d) < \tilde{d} < f(d')$,*
- *for every $d \in D$ there exists $\tilde{d}$ such that $f(d) < \tilde{d}$, and there exists $\tilde{d}$ such that $\tilde{d} < f(d)$.*

Definition 3.15 ($\sim_{ord}$)**.** Let ρ, ρ' be two configurations. We define $\rho \sim_{ord} \rho'$ iff $f(\rho) = \rho'$ for some order-preserving bijection f on $data(\rho)$.

Remark 3.16. If $\rho \rightarrowtail \rho'$ then there exists $\hat{d} \in \mathbb{D}$ such that $\{\hat{d}\} \cup data(\rho) = \{\hat{d}\} \cup data(\rho')$. This is a simple consequence of the definition of $\rightarrowtail$.

Let us define a version of $\rightarrowtail$ that works modulo $\sim_{ord}$, and let us call it $\rightarrowtail_{ord}$.

Definition 3.17. Let ρ_1, ρ_2 be two configurations. We define $\rho_1 \rightarrowtail_{ord} \rho_2$ iff $\rho'_1 \rightarrowtail \rho'_2$ for some $\rho'_1 \sim_{ord} \rho_1$ and $\rho'_2 \sim_{ord} \rho_2$.

In the previous section, when we had that $\sim$ was simply a bijection and we could not test any linear order $<$, it was clear that we could work modulo $\sim$. However, here we are working modulo a more complex relation $\sim_{ord}$. In the next lemma we show that working with $\rightarrowtail$ or working with $\rightarrowtail_{ord}$ is equivalent.

Lemma 3.18. *If $\rho_1 \rightarrowtail_{ord} \cdots \rightarrowtail_{ord} \rho_n$, then there exist $\rho'_1, \ldots, \rho'_n$ so that $\rho'_1 \rightarrowtail \cdots \rightarrowtail \rho'_n$, with $\rho'_i \sim_{ord} \rho_i$ for every i.*

Proof. The case $n = 1$ is trivial. Otherwise, if $n > 1$, we have $\rho_1 \rightarrowtail_{ord} \cdots \rightarrowtail_{ord} \rho_{n-1} \rightarrowtail_{ord} \rho_n$. Then, by inductive hypothesis, we obtain $\rho'_1 \rightarrowtail \cdots \rightarrowtail \rho'_{n-1}$ and $\rho''_{n-1} \rightarrowtail \rho''_n$ with $\rho'_i \sim_{ord} \rho_i$ for every $i \in \{1, \ldots, n-1\}$, and $\rho''_j \sim_{ord} \rho_j$ for every $j \in \{n-1, n\}$. Let g be the witnessing bijection such that $g(\rho''_{n-1}) = \rho'_{n-1}$, and let us assume that $\{\hat{d}\} \cup data(\rho''_{n-1}) = \{\hat{d}\} \cup data(\rho''_n)$ by Remark 3.16.

Let f be an order-preserving bijection on $\bigcup_{i \leq n-1} data(\rho_i')$ as in Lemma 3.14. We can then pick a data value $\tilde{d}$ such that

- for every $d > \hat{d}$ with $d \in data(\rho_{n-1}'')$, $f(g(d)) > \tilde{d}$, and
- for every $d < \hat{d}$ with $d \in data(\rho_{n-1}'')$, $f(g(d)) < \tilde{d}$.

Let $h := (g \circ f)[\hat{d} \mapsto \tilde{d}]$. We then have

- $h(\rho_{n-1}'') \rightarrowtail h(\rho_n'')$,
- $f(\rho_1') \rightarrowtail \cdots \rightarrowtail f(\rho_{n-1}')$,
- $f(\rho_{n-1}') = h(\rho_{n-1}'')$.

In other words, $f(\rho_1') \rightarrowtail \cdots \rightarrowtail f(\rho_{n-1}') \rightarrowtail h(\rho_n'')$, with $h(\rho_n'') \sim_{ord} \rho_n$ and $f(\rho_i') \sim_{ord} \rho_i$ for every $i \leq n-1$. $\qquad \square$

Now that we proved that we can work modulo $\sim_{ord}$, we show that we can decide if we can reach an accepting configuration by means of $\rightarrowtail_{ord}$, by introducing some suitable ordering $\ll$ and showing the following lemmas.

Lemma 3.19. $(\mathcal{C}_{\mathrm{ARA}}, \ll)$ *is a well-quasi-order.*

Lemma 3.20. $(\mathcal{C}_{\mathrm{ARA}}, \ll)$ *is* rdc *with respect to* $\rightarrowtail_{ord}$.

Lemma 3.21. *The set of accepting configurations of* $\mathcal{C}_{\mathrm{ARA}}$ *is downward closed with respect to* $\ll$.

We next define the order $\ll$ and show that the aforementioned lemmas are valid. In the same spirit as before, $\ll$ is defined as $\preceq$ modulo $\sim_{ord}$.

Definition 3.22 ($\ll$). $\rho_1 \ll \rho_2$ iff $\rho_1' \preceq \rho_2'$ for some $\rho_1' \sim_{ord} \rho_1$ and $\rho_2' \sim_{ord} \rho_2$.

To prove Lemma 3.19, given a configuration $\rho = \langle i_0, \alpha, (a, d), \Delta \rangle$, with $data(\rho) = \{d_1 < \cdots < d_n\}$ we define

$$abs(d_j) = \Delta(d_j) \cup \{\star \mid d_j = d\} \subseteq Q \cup \{\star\}$$
$$abs(\rho) = abs(d_1), \ldots, abs(d_n) \in (\wp(Q \cup \{\star\}))^*$$

where $\star \notin Q$ is to denote that the data value is the one of the current configuration.

Proof of Lemma 3.19. This is a consequence of Higman's Lemma stated as in Corollary 2.3. As stated above, we can see each configuration $\rho = (i, \alpha, (a, d), \Delta)$ as a word over $(\wp(Q \cup \{\star\}))^*$. As shown in Lemma 3.9 if there is an infinite sequence, there is an infinite subsequence $\rho_1, \rho_2, \ldots$, with the same type α and letter a. Then for the infinite sequence $abs(\rho_1), abs(\rho_2), \ldots$, Corollary 2.3 tells us that there are $i < j$ such that $abs(\rho_i)$ is a substring of $abs(\rho_j)$. This implies that they are in the $\ll$ relation. $\qquad \square$

Proof of Lemma 3.20. Note that although $\ll$ is a more restricted wqo, for all the non-moving cases in which the register is not modified (that is, all except guess, spread, and store), the $\rightarrow_\varepsilon$ transition continues to be rdc. This is because for any $\tau \ll \rho \rightarrow_\epsilon \rho'$, $\rho = \langle i, \alpha, \gamma, \Delta \rangle$ and $\rho' = \langle i', \alpha', \gamma', \Delta' \rangle$ are similar in the following sense. Firstly $data(\rho) = data(\rho')$, and moreover the only difference between ρ and ρ' is that Δ' is the result of removing some thread (q, d) from Δ and inserting another one (q', d) with the same data value d. This kind of operation is compatible, since τ can perform the same operation $\tau \rightarrow_\varepsilon \tau'$ on the data value d', supposing that d' is the preimage of d given by the $\ll$ ordering. In this case, $\tau' \ll \rho'$. Otherwise, if there is no preimage of d, then $\tau \ll \rho'$. The compatibility of spread is shown equivalently.

Regarding the store instruction, we see that the operation consists in removing some (q, d) from Δ and inserting some (q', d_0) with d_0 the datum of the current configuration. This is downwards compatible since τ can perform the same operation on the configuration's data value, which is necessarily the preimage of d_0.

For the remaining two cases (guess and $\rhd$) we rely on the premise that we work modulo $\sim_{ord}$. The idea is that we can always assume that we have enough data values to choose from in between the existing ones. That is, for every pair of data values $d < d'$ in a configuration, there is always one in between. We can always assume this since otherwise we can apply a bijection as the one described by Lemma 3.14 to obtain this property. Thus, at each point where we need to guess a data value (as a consequence of a guess(q) or a $\rhd q$ instruction) we will have no problem in performing a symmetric action, preserving the embedding relation.

More concretely, suppose the execution of a transition $\delta(q) = \text{guess}(q')$ on a thread (q, d_j) of configuration ρ with $data(\rho) = \{d_1 < \cdots < d_n\}$ guesses a data value d with $d_i < d < d_{i+1}$. Then, for any configuration $\tau \ll \rho$ with $data(\tau) = \{e_1 < \cdots < e_m\}$ and the property just described, there must be an order-preserving injection $f : data(\tau) \to data(\rho)$ with $f(\tau) \preceq \rho$. If τ contains a thread $(q, e_{j'})$ with $f(e_{j'}) = d_j$ the operation is simulated by guessing a data value e such that $e > e_\ell$ for all e_ℓ such that $f(e_\ell) \leq d_i$ and $e < e_k$ for all e_k such that $f(e_k) > d_i$. Such data value e exists as explained before. The rdc compatibility of a $\delta(q) = \rhd q'$ instruction is shown in an analogous fashion. $\qquad\square$

Proof of Lemma 3.21. Given that $\ll$ is a subset of $\preceq$, and that by Lemma 3.11 the set of accepting configurations is $\preceq$-downward closed, it follows that this set is also $\ll$-downward closed. $\qquad\square$

Finally, we should note that $(\mathcal{C}_{\text{ARA}}/\!\sim_{ord}, \rightarrowtail_{ord})$ is also finitely branching and effective. As in the proof of Section 3.2, by Lemmas 3.19, 3.20 and 3.21 we have that all the conditions of Proposition 2.7 are met and by Lemma 2.8 we conclude that the emptiness problem for ARA(guess, spread, $<$) is decidable.

Remark 3.23. Notice that this proof works independently of the particular ordering of $(\mathbb{D}, <)$. It could be dense or discrete, contain accumulation points, be open or closed, etc. In some sense, this automata model is blind to these kind of properties. If there is an accepting run on $(\mathbb{D}, <)$ then there is an accepting run on $(\mathbb{D}, <')$ for any linear order $<'$.

Open question 3.24. *It is perhaps possible that these results can be extended to prove decidability when $(\mathbb{D}, <)$ is a tree-like partial order, this time making use of Kruskal's tree theorem [Kru60] instead of Higman's Lemma. We leave this issue as an open question.*

Remark 3.25 (constants). One can also extend this model with a finite number of constants $\{c_1, \ldots, c_n\} \subseteq \mathbb{D}$. In this case, we extend the transitions with the possibility of testing that the data value stored in the register is (or is not) equal to c_i, for every i. In the proof, it suffices to modify $\sim_{ord}$ to take into account every constant c_i. In this case we define that $\rho \sim_{ord} \tau$ iff $f(\rho) = \rho'$ for some order-preserving bijection f on $data(\rho) \cup \{c_1, \ldots, c_n\}$ such that $f(c_i) = c_i$ for every i. In this case Lemma 3.14 does not hold anymore, as there could be finitely many elements in between two constants. This is however an easily surmountable obstacle, by adapting Lemma 3.14 to work separately on the $n + 1$ intervals defined by $c_1, \ldots, c_n$. Suppose $c_1 < \cdots < c_n$. Without any loss of generality we can assume that between c_i and c_{i+1} there are infinitely many data values, or none (we can always add some constants to ensure this). Then, for every infinite interval $[c_i, c_{i+1}]$, we will have a lemma like Lemma 3.14 that we can apply separately.

3.4. **Timed automata.** Our investigation on register automata also yields new results on the class of timed automata. An *alternating 1-clock timed automaton* is an automaton that runs over *timed words.* A timed word is a finite sequence of *events.* Each event carries a symbol from a finite alphabet and a *timestamp* indicating the quantity of time elapsed from the first event of the word. A timed word can hence be seen as a data word over the rational numbers, whose data values are strictly increasing. The automaton has alternating control and contains one *clock* to measure the lapse of time between two events (that is, the difference between the data of two positions of the data word). It can reset the clock, or test whether the clock contains a number equal, less or greater than a constant, from some finite set of constants. For more details on this automaton we refer the reader to [AD94].

Register automata over ordered domains have a strong connection with timed automata. The work in [FHL10] shows that the problems of nonemptiness, language inclusion, language equivalence and universality are equivalent—modulo an EXPTIME reduction—for timed automata and register automata over a linear order. That is, any of these problems for register automata can be reduced to the same problem on timed automata, preserving the number of registers equal to the number of clocks, and the mode of computation (nondeterministic, alternating). And in turn, any of these problems for timed automata can also be reduced to a similar problem on register automata over a linear order. We argue that this is also true when the automata are equipped with guess and spread.

Consider an extension of 1-clock alternating timed automata, with spread and guess, where

- the operator $\mathsf{spread}(q, q')$ works in the same way as for register automata, duplicating all threads with state q as threads with state q', and
- the $\mathsf{guess}(q)$ operator resets the clock to any value, non deterministically chosen, and continues the execution with state q.

The coding technique of [FHL10] can be adapted to deal with the guessing of a clock (the spread operator being trivially compatible), and one can show the following statement.

Remark 3.26. The emptiness problem for alternating 1-clock timed automata extended with guess and spread reduces to the emptiness problem for the class $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread}, <)$.

Hence, by Remark 3.26 cum Theorem 3.12 we obtain the following.

Remark 3.27. The emptiness problem for alternating 1-clock timed automata extended with guess and spread is decidable.

3.5. **A note on complexity.** Although the $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread})$ and $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread}, <)$ classes have both non-primitive-recursive complexity, we must remark that the decision procedure for the latter has much higher complexity. While the former can be roughly bounded by the Ackermann function applied to the number of states, the complexity of the decision procedure we give for $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread}, <)$ majorizes every multiply-recursive function (in particular, Ackermann's). In some sense this is a consequence of relying on Higman's Lemma instead of Dickson's Lemma for the termination arguments of our algorithm.

More precisely, it can be seen that the emptiness problem for $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread}, <)$ sits in the class $\mathfrak{F}_{\omega^\omega}$ in the Fast Growing Hierarchy [LW70]—an extension of the Grzegorczyk Hierarchy for non-primitive-recursive functions—by a reduction to the emptiness problem

for timed one clock automata (see §3.4), which are known to be in this class.[3] However, the emptiness problem for ARA(guess, spread) belongs to $\mathfrak{F}_\omega$ in the hierarchy. The lower bound follows by a reduction from Incrementing Counter Automata [DL09], which are hard for $\mathfrak{F}_\omega$ [Sch10, FFSS11]. The upper bound is a consequence of using a saturation algorithm with a wqo that is the component-wise order of the coordinates of a vector of natural numbers in a controlled way. The proof that it belongs to $\mathfrak{F}_\omega$ goes similarly as for Incrementing Counter Automata [see FFSS11, §7.2]. We do not know whether the emptiness problem for ARA(guess, spread, $<$) is also in $\mathfrak{F}_\omega$.

4. LTL with registers

The logic $\mathsf{LTL}^\downarrow(\mathsf{U}, \mathsf{X})$ is a logic for data words that corresponds to the extension of the Linear-time Temporal Logic $\mathsf{LTL}(\mathsf{U}, \mathsf{X})$ on data words, with the ability to use one *register* for storing a data value for later comparisons, studied in [DL09, DLN05]. It contains the next (X) and until (U) temporal operators to navigate the data word, and two extra operators. The *freeze* operator $\downarrow \varphi$ permits to *store* the current datum in the register and continue the evaluation of the formula φ. The operator $\uparrow$ *tests* whether the current data value is equal to the one stored in the register.

As it was shown in [DL09], if we allow more than one register to store data values, satisfiability of $\mathsf{LTL}^\downarrow(\mathsf{U}, \mathsf{X})$ becomes undecidable. We will then focus on the language that uses only *one* register. We study an extension of this language with a restricted form of *quantification* over data values. We will actually add *two* sorts of quantification. On the one hand, the operators $\forall^\downarrow_\leq$ and $\exists^\downarrow_\leq$ quantify universally or existentially over all the data values occurring *before* the current point of evaluation. Similarly, $\forall^\downarrow_\geq$ and $\exists^\downarrow_\geq$ quantify over the *future* elements on the data word. For our convenience and without any loss of generality, we will work in Negation Normal Form (nnf), and we use $\bar{\mathsf{U}}$ to denote the dual operator of U, and similarly for $\bar{\mathsf{X}}$. Sentences of $\mathsf{LTL}^\downarrow_{\mathsf{nnf}}(\mathsf{U}, \bar{\mathsf{U}}, \mathsf{X}, \bar{\mathsf{X}}, \mathcal{O})$, where $\mathcal{O} \subseteq \{\forall^\downarrow_\leq, \exists^\downarrow_\leq, \forall^\downarrow_\geq, \exists^\downarrow_\geq\}$ are defined as follows,

$$\varphi ::= a \mid \neg a \mid \uparrow \mid \neg \uparrow \mid \downarrow \varphi \mid \mathsf{X}\varphi \mid \bar{\mathsf{X}}\varphi \mid \mathsf{U}(\varphi, \varphi) \mid \bar{\mathsf{U}}(\varphi, \varphi) \mid \mathsf{op}\, \varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$

where a is a symbol from a finite alphabet $\mathbb{A}$, and $\mathsf{op} \in \mathcal{O}$. For economy of space we write $\mathsf{LTL}^\downarrow_{\mathsf{nnf}}(\mathfrak{F}, \mathcal{O})$ to denote this logic. In this notation, $\mathfrak{F}$ is to mark that we have all the *forward* modalities: $\mathsf{U}, \bar{\mathsf{U}}, \mathsf{X}, \bar{\mathsf{X}}$. Notice that the *future* modality can be defined $\mathsf{F}\varphi := \mathsf{U}(\varphi, \top)$ and its dual $\mathsf{G}\varphi$ as the nnf of $\neg\mathsf{F}\neg\varphi$.

Figure 4 shows the definition of the satisfaction relation $\models$. For example, in this logic we can express properties like "*for every a element there is a future b element with the same data value*" as $\mathsf{G}(\neg a \vee \downarrow (\mathsf{F}(b \wedge \uparrow)))$. We say that φ satisfies $\mathbf{w} = \mathbf{a} \otimes \mathbf{d}$, written $\mathbf{w} \models \varphi$, if $\mathbf{w}, 1 \models^{\mathbf{d}(1)} \varphi$.

4.1. **Satisfiability problem.** This section is dedicated to the satisfiability problem for $\mathsf{LTL}^\downarrow_{\mathsf{nnf}}(\mathfrak{F}, \forall^\downarrow_\leq, \exists^\downarrow_\geq)$. But first let us show that $\exists^\downarrow_\leq$ and $\forall^\downarrow_\geq$ result in an undecidable logic.

Theorem 4.1. *Let $\exists^\downarrow_<$ be the operator $\exists^\downarrow_\leq$ restricted only to the data values occurring strictly before the current point of evaluation. Then, on finite data words:*

---

[3]The emptiness problem for timed one clock automata can be at the same time reduced to that of Lossy Channel Machines [ADOW05], which are known to be 'complete' for this class, *i.e.*, in $\mathfrak{F}_{\omega^\omega} \setminus \mathfrak{F}_{<\omega^\omega}$ [see CS08].

$$
\begin{aligned}
(\mathbf{w}, i) &\models^d a &\text{iff}\quad &\mathbf{a}(i) = a \\
(\mathbf{w}, i) &\models^d \uparrow &\text{iff}\quad &d = \mathbf{d}(i) \\
(\mathbf{w}, i) &\models^d \downarrow \varphi &\text{iff}\quad &(\mathbf{w}, i) \models^{\mathbf{d}(i)} \varphi \\
(\mathbf{w}, i) &\models^d \mathsf{U}(\varphi, \psi) &\text{iff}\quad &\text{for some } i \leq j \in \mathsf{pos}(\mathbf{w}) \text{ and for all } i \leq k < j \\
& & &\text{we have } (\mathbf{w}, j) \models^d \varphi \text{ and } (\mathbf{w}, k) \models^d \psi \\
(\mathbf{w}, i) &\models^d \mathsf{X}\varphi &\text{iff}\quad &i + 1 \in \mathsf{pos}(\mathbf{w}) \text{ and } (\mathbf{w}, i+1) \models^d \varphi \\
(\mathbf{w}, i) &\models^d \exists^{\downarrow}_{\geq}\varphi &\text{iff}\quad &\text{there exists } i \leq j \in \mathsf{pos}(\mathbf{w}) \text{ such that } (\mathbf{w}, i) \models^{\mathbf{d}(j)} \varphi \\
(\mathbf{w}, i) &\models^d \forall^{\downarrow}_{\leq}\varphi &\text{iff}\quad &\text{for all } 1 \leq j \leq i \text{ we have } (\mathbf{w}, i) \models^{\mathbf{d}(j)} \varphi
\end{aligned}
$$

Figure 4: Semantics of $\mathsf{LTL}^{\downarrow}(\mathsf{U}, \bar{\mathsf{U}}, \mathsf{X}, \bar{\mathsf{X}}, \exists^{\downarrow}_{\geq}, \forall^{\downarrow}_{\leq})$ for a data word $\mathbf{w} = \mathbf{a} \otimes \mathbf{d}$ and $i \in \mathsf{pos}(\mathbf{w})$.

(1) satisfiability of $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathsf{F}, \mathsf{G}, \exists^{\downarrow}_{<})$ is undecidable; and
(2) satisfiability of $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathsf{F}, \mathsf{G}, \forall^{\downarrow}_{\geq})$ is undecidable.

Proof. We prove (1) and (2) by reduction of the halting problem for Minsky machines. We show that these logics can code an accepting run of a 2-counter Minsky machine as in Proposition 3.2. Indeed, we show that the same kind of properties are expressible in this logic. To prove this, we build upon some previous results [FS09] showing that $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathsf{F}, \mathsf{G})$ can code conditions (i) and (ii) of the proof of Proposition 3.2. Here we show that both $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathsf{F}, \mathsf{G}, \exists^{\downarrow}_{<})$ and $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathsf{F}, \mathsf{G}, \forall^{\downarrow}_{\geq})$ can express condition (iii), ensuring that for every decrement (dec_i) there is a previous increment (inc_i) with the same data value. Let us see how to code this.

(1) The $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathsf{F}, \mathsf{G}, \exists^{\downarrow}_{<})$ formula

$$\mathsf{G}(\mathsf{dec}_i \to \exists^{\downarrow}_{<} \uparrow)$$

states that the data value of every decrement must *not* be new, and in the context of this coding this means that it must have been introduced by an increment instruction.

(2) The $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathsf{F}, \mathsf{G}, \forall^{\downarrow}_{\geq})$ formula

$$\forall^{\downarrow}_{\geq}(\mathsf{F}(\mathsf{dec}_i \wedge \uparrow) \to \mathsf{F}(\mathsf{inc}_i \wedge \uparrow))$$

evaluated at the first element of the data word expresses that for every data value: if there is a *decrement* with value d, then there is an *increment* with value d. It is then easy to ensure that they appear in the correct order (first the increment, then the decrement).

The addition of any of these conditions to the coding of [FS09] results in a coding of an n-counter Minsky machine, whose emptiness problem is undecidable. $\qquad\square$

Corollary 4.2. *The satisfiability problem for both $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \exists^{\downarrow}_{\leq})$ and $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \forall^{\downarrow}_{\geq})$ are undecidable.*

Proof. The property of item (1) in the proof of Theorem 4.1 can be equally coded in $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \exists^{\downarrow}_{\leq})$ as $\mathsf{G}(\mathsf{X}(\mathsf{dec}_i) \to \exists^{\downarrow}_{\leq}(\mathsf{X} \uparrow))$. The undecidability of $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \forall^{\downarrow}_{\geq})$ follows directly from Theorem 4.1, item (2). □

We now turn to our decidability result. We show that $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \forall^{\downarrow}_{\leq}, \exists^{\downarrow}_{\geq})$ has a decidable satisfiability problem by a translation to $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread})$.

The translation to code $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \forall^{\downarrow}_{\leq}, \exists^{\downarrow}_{\geq})$ into $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread})$ is standard, and follows same lines as [DL09][4] (which at the same time follows the translation from $\mathsf{LTL}^{\downarrow}(\mathsf{U}, \mathsf{X})$ to alternating finite automata). We then obtain the following result.

Proposition 4.3. $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread})$ *captures* $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \exists^{\downarrow}_{\geq}, \forall^{\downarrow}_{\leq})$.

From Proposition 4.3 and Theorem 3.5 it will follow the main result, stated next.

Theorem 4.4. *The satisfiability problem for* $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \exists^{\downarrow}_{\geq}, \forall^{\downarrow}_{\leq})$ *is decidable.*

We now show how to make the translation to $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread})$ in order to obtain our decidability result.

Proof of Proposition 4.3. Let $\eta \in \mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \forall^{\downarrow}_{\leq}, \exists^{\downarrow}_{\geq})$. We show that for every formula $\eta \in \mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \forall^{\downarrow}_{\leq}, \exists^{\downarrow}_{\geq})$ there exists a computable $\mathsf{ARA}(\mathsf{spread}, \mathsf{guess})$ $\mathcal{A}_\eta$ such that for every data word $\mathbf{w}$,

$$\mathbf{w} \text{ satisfies } \eta \text{ iff } \mathcal{A}_\eta \text{ accepts } \mathbf{w}.$$

In the construction of $\mathcal{A}_\varphi$, we first make sure to maintain all the data values seen so far as *threads* of the configuration. We can do this by having 4 special states q_0, q_1, q_2, q_{save} in $\mathcal{A}_\eta$, defining q_0 as the initial state, and δ as follows.

$$\delta(q_0) = q_\eta \wedge q_1 \quad \delta(q_1) = \mathsf{store}(q_2) \quad \delta(q_2) = (\bar{\triangleright}? \vee \triangleright q_1) \wedge q_{save} \quad \delta(q_{save}) = \bar{\triangleright}? \vee \triangleright q_{save}$$

Now we can assume that at any point of the run, we maintain the data values of all the previous elements of the data word as threads (q_{save}, d). Note that these threads are maintained until the last element of the data word, at which point the test $\bar{\triangleright}?$ is satisfied and they are accepted.

Now we show how to define $\mathcal{A}_\eta$. We proceed by induction on $|\eta|$. If $\eta = a$ or $\neg a$, we simply define the set of states as $Q = \{q_0, q_1, q_2, q_{\mathsf{save}}, q_\eta\}$ and $\delta(q_\eta) = a$ or $\delta(q_\eta) = \bar{a}$ ($\delta(q_0)$, $\delta(q_1)$, $\delta(q_2)$ and $\delta(q_{\mathsf{save}})$ are defined as above). If $\eta = \downarrow\psi$ (or $\eta = \uparrow$, $\eta = \neg\uparrow$) we define it as follows. We add one new state q_η to the set of states of $\mathcal{A}_\psi$, we extend the definition of δ with $\delta(q_\eta) = \mathsf{store}(q_\psi)$ (or $\delta(q_\eta) = \mathsf{eq}(q_\psi)$, $\delta(q_\eta) = \overline{\mathsf{eq}}(q_\psi)$), and we redefine $\delta(q_1)$ with $\delta(q_1) = \mathsf{store}(q_2) \wedge q_\eta$. If $\eta = \mathsf{F}\psi$, $\eta = \mathsf{G}\psi$, or $\eta = \mathsf{U}(\psi, \psi')$, it is easy to define $\delta(q_\eta)$ from the definition of $\mathcal{A}_\psi$ and $\mathcal{A}_{\psi'}$, perhaps adding some new states. On the other hand, if $\varphi = \psi \wedge \psi'$ or $\varphi = \psi \vee \psi'$, it is also straightforward as it corresponds to the alternation and nondeterminism of the automaton.

Suppose now that $\eta = \forall^{\downarrow}_{\leq}\psi$. We define $\mathcal{A}_\eta$ as $\mathcal{A}_\psi$ with the extra state q_η and we define $\delta(q_\eta) = \mathsf{spread}(q_{\mathsf{save}}, q_\psi) \wedge \downarrow q_\psi$. This ensures that at the moment of execution of this instruction, all the previous data values in the data word will be taken into account, including the current one.

---

[4]Note that this logic already contains $\mathsf{LTL}^{\downarrow}(\mathsf{U}, \mathsf{X})$.

$$(\mathbf{w}, i) \models^d \uparrow_> \quad \text{iff} \quad d > \mathbf{d}(i)$$
$$(\mathbf{w}, i) \models^d \uparrow_< \quad \text{iff} \quad d < \mathbf{d}(i)$$

Figure 5: Semantics for the operators $\uparrow_>, \uparrow_<$ for a data word $\mathbf{w} = \mathbf{a} \otimes \mathbf{d}$ and $i \in \mathsf{pos}(\mathbf{w})$.

Finally, if $\eta = \exists^{\downarrow}_{\geq} \psi$, we build $\mathcal{A}_\eta$ from $\mathcal{A}_\psi$ in the same fashion as before, adding new states $q_\eta, q'_\eta, q''_\eta$, defining $\delta(q_\eta) = \mathsf{guess}(q'_\eta)$, $\delta(q'_\eta) = q_\psi \wedge q''_\eta$, $\delta(q''_\eta) = \mathsf{eq} \vee \triangleright q''_\eta$. Note that q''_η checks that the guessed data value appears somewhere in a future position of the word. $\square$

Moreover, we argue that these extensions add expressive power.

Proposition 4.5. *On finite data words:*

(i) *The logic* $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \forall^{\downarrow}_{\leq})$ *is more expressive than* $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F})$;

(ii) *The logic* $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \exists^{\downarrow}_{\geq})$ *is more expressive than* $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F})$.

Proof. This is a consequence of $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F})$ being closed under negation and Theorem 4.1. Ad absurdum, if one of these logics were as expressive as $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F})$, then it would be closed under negation, and then we could express conditions (1) or (2) of the proof of Theorem 4.1 and hence obtain that $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F})$ is undecidable. But this leads to a contradiction since by Theorem 4.4 $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F})$ is decidable. $\square$

Remark 4.6. The translation of Proposition 4.3 is far from using all the expressive power of spread. In fact, we can consider a binary operator $\forall^{\downarrow}_{\leq}(\varphi, \psi)$ defined

$$\mathbf{w}, i \models^d \forall^{\downarrow}_{\leq}(\varphi, \psi) \text{ iff for all } j \leq i \text{ such that } \mathbf{w}, j \models^{\mathbf{d}(j)} \psi, \text{ we have } \mathbf{w}, i \models^{\mathbf{d}(j)} \varphi.$$

with $\psi \in \mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F})$. This operator can be coded into $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread})$, using the same technique as in Proposition 4.3. The only difference is that instead of 'saving' every data value in q_{save}, we use several states $q_{\mathrm{save}(\psi)}$. Intuitively, only the data values that verify the test $\downarrow \psi$ are stored in $q_{\mathrm{save}(\psi)}$. Then, a formula $\forall^{\downarrow}_{\leq}(\varphi, \psi)$ is translated as $\mathsf{spread}(q_{\mathrm{save}(\psi)}, q_\varphi)$.

4.2. **Ordered data.** If we consider a linear order over $\mathbb{D}$ as done in Section 3.3, we can consider $\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \exists^{\downarrow}_{\geq}, \forall^{\downarrow}_{\leq})$ with richer tests

$$\varphi ::= \quad \uparrow_> \mid \uparrow_< \mid \dots$$

that access to the linear order and compare the data values for $=, <, >$. The semantics are extended accordingly, as in Figure 5. Let us call this logic *ord*-$\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \exists^{\downarrow}_{\geq}, \forall^{\downarrow}_{\leq})$. The translation from *ord*-$\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \exists^{\downarrow}_{\geq}, \forall^{\downarrow}_{\leq})$ to $\mathsf{ARA}(\mathsf{guess}, \mathsf{spread}, <)$ as defined in Section 3.3 is straightforward. Thus we obtain the following result.

Proposition 4.7. *The satisfiability problem for ord-*$\mathsf{LTL}^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \exists^{\downarrow}_{\geq}, \forall^{\downarrow}_{\leq})$ *is decidable.*

Part 2. **Data trees**

The second part of this work deals with logics and automata for *data trees*. In Section 5, we extend the model ARA(guess, spread) to the new model ATRA(guess, spread) that runs over data trees (instead of data words). The decidability of the emptiness follows easily from the decidability result shown for ARA(guess, spread). As in the case of data trees, this model allows to show the decidability of a logic.

In Section 6 we introduce 'forward XPath', a logic for XML documents. The satisfiability problem for this logic will follow by a reduction to the emptiness problem of ATRA(guess, spread). This reduction is not as easy as that of the first part, since XPath is closed under negation and our automata model is not closed under complementation. Indeed, ATRA(guess, spread) and forward XPath have incomparable expressive power.

5. ATRA MODEL

Herein, we introduce the class of Alternating Tree Register Automata by slightly adapting the definition for alternating (word) register automata. This model is essentially the same automaton presented in Part 1, that works on a (unranked, ordered) *data tree* instead of a data word. The only difference is that instead of having one instruction $\rhd$ that means 'move to the next position', we have two instructions $\rhd$ and $\triangledown$ meaning 'move to the next sibling to the right' and 'move to the leftmost child'. This class of automata is known as ATRA(guess, spread). This model of computation will enable us to show decidability of a large fragment of XPath.

An **Alternating Tree Register Automaton** (ATRA) consists in a top-down tree walking automaton with alternating control and *one* register to store and test data. [JL11] shows that its emptiness problem is decidable and non-primitive-recursive. Here, as in the Part 1, we consider an extension with the operators spread and guess. We call this model ATRA(spread, guess).

Definition 5.1. An alternating tree register automaton of ATRA(spread, guess) is a tuple $\mathcal{A} = \langle \mathbb{A}, Q, q_I, \delta \rangle$ such that $\mathbb{A}$ is a finite alphabet; Q is a finite set of states; $q_I \in Q$ is the initial state; and $\delta : Q \to \Phi$ is the transition function, where Φ is defined by the grammar

$$a \mid \bar{a} \mid \odot? \mid \mathsf{store}(q) \mid \mathsf{eq} \mid \overline{\mathsf{eq}} \mid q \wedge q' \mid q \vee q' \mid \triangledown q \mid \rhd q \mid \mathsf{guess}(q) \mid \mathsf{spread}(q, q')$$

where $a \in \mathbb{A}, q, q' \in Q, \odot \in \{\triangledown, \bar{\triangledown}, \rhd, \bar{\rhd}\}$.

We only focus on the differences with respect to the ARA class. $\triangledown$ and $\rhd$ are to move to the leftmost child or to the next sibling to the right of the current position, and as before '$\odot?$' tests the current type of the position of the tree. For example, using $\bar{\triangledown}?$ we test that we are in a leaf node, and by $\rhd?$ that the node has a sibling to its right. $\mathsf{store}(q)$, eq and $\overline{\mathsf{eq}}$ work in the same way as in the ARA model. We say that a state $q \in Q$ is **moving** if $\delta(q) = \rhd q'$ or $\delta(q) = \triangledown q'$ for some $q' \in Q$.

We define two sorts of configurations: *node* configurations and *tree* configurations. In this context a **node configuration** is a tuple $\langle x, \alpha, \gamma, \Delta \rangle$ that describes the partial state of the execution at a position x of the tree. $x \in \mathsf{pos}(\mathbf{t})$ is the current position in the tree $\mathbf{t}$, $\gamma = \mathbf{t}(x) \in \mathbb{A} \times \mathbb{D}$ is the current node's symbol and datum, and $\alpha = \mathsf{type}_{\mathbf{t}}(x)$ is the tree type of x. As before, $\Delta \in \wp_{<\infty}(Q \times \mathbb{D})$ is a finite collection of execution threads. $\mathcal{N}_{\mathrm{ATRA}}$ is the set of all node configurations. A **tree configuration** is just a finite set of node

configurations, like $\{\langle \epsilon, \alpha, \gamma, \Delta \rangle, \langle 1211, \alpha', \gamma', \Delta' \rangle, \dots \}$. We call $\mathcal{T}_{\text{ATRA}} = \wp_{<\infty}(\mathcal{N}_{\text{ATRA}})$ the set of all tree configurations.

We define the *non-moving* relation $\to_\varepsilon$ over node configurations just as in page 10. As a difference with the ARA mode, we have two types of moving relations. The *first-child* relation $\to_\nabla$, to move to the leftmost child, and the *next-sibling* relation $\to_\triangleright$ to move to the next sibling to the right.

The $\to_\nabla$ and $\to_\triangleright$ are defined, for any $\alpha_1 \in \{\nabla, \bar{\nabla}, \triangleright, \bar{\triangleright}\}$, $\gamma, \gamma_1 \in \mathbb{A} \times \mathbb{D}$, $h \in \{\triangleright, \bar{\triangleright}\}$, $v \in \{\nabla, \bar{\nabla}\}$, as follows

$$\langle x, (\nabla, h), \gamma, \Delta \rangle \to_\nabla \langle x\cdot 1, \alpha_1, \gamma_1, \Delta_\nabla \rangle, \tag{5.1}$$

$$\langle x\cdot i, (v, \triangleright), \gamma, \Delta \rangle \to_\triangleright \langle x\cdot(i+1), \alpha_1, \gamma_1, \Delta_\triangleright \rangle \tag{5.2}$$

iff (i) the configuration is 'moving' (*i.e.*, all the threads (q, d) contained in Δ are of the form $\delta(q) = \nabla q'$ or $\delta(q) = \triangleright q'$); and (ii) for $\odot \in \{\nabla, \triangleright\}$, $\Delta_\odot = \{(q', d) \mid (q, d) \in \Delta, \delta(q) = \odot q'\}$.

Let $\to := \to_\varepsilon \cup \to_\nabla \cup \to_\triangleright \subseteq \mathcal{N}_{\text{ATRA}} \times \mathcal{N}_{\text{ATRA}}$. Note that through $\to$ we obtain a run over a *branch* of the tree (if we think about the underlying binary tree according to the first-child and next-sibling relations). In order to maintain all information about the run over all branches we need to lift this relation to tree configurations. We define the transition between tree configurations that we write $\twoheadrightarrow$. This corresponds to applying a 'non-moving' $\to_\varepsilon$ to a node configuration, or to apply a 'moving' $\to_\nabla$, $\to_\triangleright$, or both to a node configuration according to its type. That is, we define $\mathcal{S}_1 \twoheadrightarrow \mathcal{S}_2$ iff one of the following conditions holds:

(1) $\mathcal{S}_1 = \{\rho\} \cup \mathcal{S}'$, $\mathcal{S}_2 = \{\tau\} \cup \mathcal{S}'$, $\rho \to_\varepsilon \tau$;
(2) $\mathcal{S}_1 = \{\rho\} \cup \mathcal{S}'$, $\mathcal{S}_2 = \{\tau\} \cup \mathcal{S}'$, $\rho = \langle x, (\nabla, \bar{\triangleright}), \gamma, \Delta \rangle$, $\rho \to_\nabla \tau$;
(3) $\mathcal{S}_1 = \{\rho\} \cup \mathcal{S}'$, $\mathcal{S}_2 = \{\tau\} \cup \mathcal{S}'$, $\rho = \langle x, (\bar{\nabla}, \triangleright), \gamma, \Delta \rangle$, $\rho \to_\triangleright \tau$;
(4) $\mathcal{S}_1 = \{\rho\} \cup \mathcal{S}'$, $\mathcal{S}_2 = \{\tau_1, \tau_2\} \cup \mathcal{S}'$, $\rho = \langle x, (\nabla, \triangleright), \gamma, \Delta \rangle$, $\rho \to_\nabla \tau_1$, $\rho \to_\triangleright \tau_2$.

A *run* over a data tree $\mathbf{t} = \mathbf{a} \otimes \mathbf{d}$ is a nonempty sequence $\mathcal{S}_1 \twoheadrightarrow \cdots \twoheadrightarrow \mathcal{S}_n$ with $\mathcal{S}_1 = \{\langle \epsilon, \alpha_0, \gamma_0, \Delta_0 \rangle\}$ and $\Delta_0 = \{(q_I, \mathbf{d}(\epsilon))\}$ (*i.e.*, the thread consisting in the initial state with the root's datum), such that for every $i \in [n]$ and $\langle x, \alpha, \gamma, \Delta \rangle \in \mathcal{S}_i$: (1) $x \in \mathsf{pos}(\mathbf{t})$; (2) $\gamma = \mathbf{t}(x)$; and (3) $\alpha = \mathsf{type}_{\mathbf{t}}(x)$. As before, we say that the run is *accepting* if

$$\mathcal{S}_n \subseteq \{\langle x, \alpha, \gamma, \emptyset \rangle \mid \langle x, \alpha, \gamma, \emptyset \rangle \in \mathcal{N}_{\text{ATRA}}\}.$$

Note that the transition relation that defines the run replaces a node configuration in a given position by one or two node configurations in children positions of the fcns coding. Therefor the following holds.

Remark 5.2. Every run is such that any of its tree configurations never contains two node configurations in a descendant/ancestor relation of the fcns coding.

The ATRA model is closed under all boolean operations [JL11]. However, the extensions introduced guess and spread, while adding expressive power, are not closed under complementation as a trade-off for decidability. It is not surprising that the same properties as for the case of data words apply here.

Proposition 5.3. ATRA(spread, guess) *models have the following properties:*
  (i) they are closed under union,
  (ii) they are closed under intersection,
 (iii) they are not closed under complementation.

Example 5.4. We show an example of the expressiveness that guess adds to ATRA. Although as a corollary of Proposition 3.2 we have that the ATRA(guess, spread) class is more
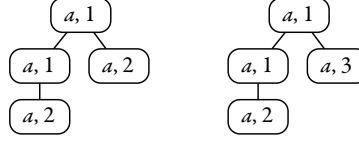
Figure 6: Two indistinguishable data trees for ATRA.

expressive than ATRA, we give an example that inherently uses the tree structure of the model. We force the node at position 2 and the node at position $1 \cdot 1$ of a data tree to have the same data value without any further data constraints. Note that this datum does not necessarily have to appear at some common ancestor of these nodes. Consider the ATRA(guess) defined over $\mathbb{A} = \{a\}$ with

$$\delta(q_0) = \mathsf{guess}(q_1), \qquad \delta(q_1) = \triangledown q_2, \qquad \delta(q_2) = q_3 \wedge q_4,$$
$$\delta(q_3) = \triangledown q_5, \qquad \delta(q_4) = \triangleright q_5, \qquad \delta(q_5) = \mathsf{eq}.$$

For the data trees of Figure 6, any ATRA either accepts both, or rejects both. This is because when a thread is at position 1, and performs a moving operation splitting into two threads, one at position $1 \cdot 1$, the other at position 2, none of these configurations contain the data value 2. Otherwise, the automaton should have read the data value 2, which is not the case. But then, we see that the continuation of the run from the node configuration at position 2 and at position 1 are isomorphic independently of which data values we choose (either the data 2 and 3, or the data 2 and 2). Hence, both trees are accepted or rejected. However, the ATRA(guess) we just built distinguishes them, as it can introduce the data value 2 in the configuration, without the need of reading it from the tree. Equivalently, the property of "there are two leaves with the same data values" is expressible in ARA(guess) and not in ARA.

5.1. **Emptiness problem.** We show that the emptiness problem for this model is decidable, reusing the results of Part 1. We remind the reader that the decidability of the emptiness of ATRA was proved in [JL11]. Here we extend the approach used for ARA and show the decidability of the two extensions spread and guess.

Theorem 5.5. *The emptiness problem of* ATRA(guess, spread) *is decidable.*

Proof. The proof goes as follows. We will reuse the wqo $\precsim$ used in Section 3.2, which here is defined over the *node configurations*. The only difference being that to use $\precsim$ over $\mathcal{N}_{\mathrm{ATRA}}$ we work with *tree* types instead of word types. Since $\rightarrow_{\triangleright}$ and $\rightarrow_{\triangledown}$ are analogous, by the same proof as in Lemma 3.10 we obtain the following.

Lemma 5.6. $(\mathcal{N}_{\mathrm{ATRA}}, \rightarrow_{\triangledown})$, $(\mathcal{N}_{\mathrm{ATRA}}, \rightarrow_{\triangleright})$, $(\mathcal{N}_{\mathrm{ATRA}}, \rightarrow_{\varepsilon})$, *(and therefore* $(\mathcal{N}_{\mathrm{ATRA}}, \rightarrow)$*)* *are* rdc *with respect to* $(\mathcal{N}_{\mathrm{ATRA}}, \precsim)$.

We now lift this result to *tree configurations*. We instantiate Proposition 2.11 by taking

- S as $\mathcal{N}_{\mathrm{ATRA}}$, $\leq$ as $\precsim$, and $\leq_{\wp}$ as the majoring order over $(\mathcal{N}_{\mathrm{ATRA}}, \precsim)$,
- $\rightarrow_1^1$ as $\rightarrow_{\triangleright}$, $\rightarrow_1^2$ as $\rightarrow_{\triangledown}$, $\rightarrow_1^3$ as $\rightarrow_{\varepsilon}$,
- $\mathcal{P} \subseteq \mathcal{N}_{\mathrm{ATRA}}$ as the set of all configurations ρ so that $\rho = \langle x, (\triangledown, \triangleright), \gamma, \Delta \rangle$ (it is immediate to se that it is downward- and upward-closed with respect to $\precsim$),
- $\rightarrow_2$ to be $\twoheadrightarrow$, as it verifies the hypotheses demanded in the Lemma.

We thus obtain the following.

Lemma 5.7. $(\mathcal{T}_{\text{ATRA}}, \twoheadrightarrow)$ *is* rdc *with respect to* $(\mathcal{T}_{\text{ATRA}}, \leq_\wp)$.

Hence, condition (1) of Proposition 2.7 is met. Let us write $\equiv$ for the equivalence relation over $\mathcal{T}_{\text{ATRA}}$ such that $\mathcal{S} \equiv \mathcal{S}'$ iff $\mathcal{S} \leq_\wp \mathcal{S}'$ and $\mathcal{S}' \leq_\wp \mathcal{S}$. Similarly as for Part 1, we have that $(\mathcal{T}_{\text{ATRA}}/\equiv, \twoheadrightarrow)$ is finitely branching and effective. That is, the $\twoheadrightarrow$-image of any configuration has only a finite number of configurations up to isomorphism of the data values contained (remember that only equality between data values matters), and representatives for every class are computable. Hence, we have that $(\mathcal{T}_{\text{ATRA}}/\equiv, \twoheadrightarrow, \leq_\wp)$ verifies condition (2) of Proposition 2.7. Finally, condition (3) holds as $(\mathcal{T}_{\text{ATRA}}/\equiv, \leq_\wp)$ is a wqo (by Proposition 2.10) that is a computable relation. We conclude the proof by the following obvious statement.

Lemma 5.8. *The set of accepting tree configurations is downwards closed with respect to* $\leq_\wp$.

Hence, by Lemma 2.8, we conclude as before that the emptiness problem for the class ATRA(guess, spread) is decidable. □

6. Forward-XPath

We consider a navigational fragment of XPath 1.0 with data equality and inequality. In particular this logic is here defined over *data trees*. However, an XML document may typically have not *one* data value per node, but a set of *attributes*, each carrying a data value. This is not a problem since every attribute of an XML element can be encoded as a child node in a data tree labeled by the attribute's name (*cf.* Section 6.3). Thus, all the decidability results hold also for XPath with attributes over XML documents.

Let us define a simplified syntax for this logic. XPath is a two-sorted language, with *path* expressions $(\alpha, \beta, \dots)$ and *node* expressions $(\varphi, \psi, \dots)$. We write $\mathsf{XPath}(\mathcal{O}, =)$ to denote the data-aware fragment with the set of axes $\mathcal{O} \subseteq \{\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \leftarrow, {}^*\!\leftarrow, \uparrow, \uparrow^*\}$. It is defined by mutual recursion as follows,

$$\alpha, \beta \; ::= \; o \;\mid\; [\varphi] \;\mid\; \alpha\beta \;\mid\; \alpha \cup \beta \qquad\qquad\qquad o \in \mathcal{O} \cup \{\varepsilon\}$$

$$\varphi, \psi \; ::= \; a \;\mid\; \neg\varphi \;\mid\; \varphi \vee \psi \;\mid\; \varphi \wedge \psi \;\mid\; \langle\alpha\rangle \;\mid\; \langle\alpha = \beta\rangle \;\mid\; \langle\alpha \neq \beta\rangle \qquad a \in \mathbb{A}$$

where $\mathbb{A}$ is a finite alphabet. A *formula* of $\mathsf{XPath}(\mathcal{O}, =)$ is either a node expression or a path expression. We define the 'forward' set of axes as $\mathfrak{F} := \{\downarrow, \downarrow^*, \rightarrow, \rightarrow^*\}$, and consequently the fragment 'forward-XPath' as $\mathsf{XPath}(\mathfrak{F}, =)$. We also refer by $\mathsf{XPath}^\varepsilon(\mathfrak{F}, =)$ to the fragment considered in [JL11] where data tests are of the restricted form $\langle\varepsilon = \alpha\rangle$ or $\langle\varepsilon \neq \alpha\rangle$.[5]

There have been efforts to extend XPath to have the full expressivity of MSO, e.g. by adding a least fix-point operator (*cf.* [tC06, Sect. 4.2]), but these logics generally lack clarity and simplicity. However, a form of recursion can be added by means of the Kleene star, which allows us to take the transitive closure of any path expression. Although in general this is not enough to already have MSO [tCS08], it does give an intuitive language

---

[5][JL11] refers to $\mathsf{XPath}^\varepsilon(\mathfrak{F}, =)$ as 'forward XPath'. Here, 'forward XPath' is the unrestricted fragment $\mathsf{XPath}(\mathfrak{F}, =)$, as we believe is more appropriate.

$$\llbracket \varepsilon \rrbracket^{\mathbf{t}} = \{(x,x) \mid x \in \mathsf{pos}(\mathbf{t})\} \qquad\qquad \llbracket \downarrow \rrbracket^{\mathbf{t}} = \{(x, x{\cdot}i) \mid x{\cdot}i \in \mathsf{pos}(\mathbf{t})\}$$

$$\llbracket \alpha \cup \beta \rrbracket^{\mathbf{t}} = \llbracket \alpha \rrbracket^{\mathbf{t}} \cup \llbracket \beta \rrbracket^{\mathbf{t}} \qquad\qquad \llbracket \rightarrow \rrbracket^{\mathbf{t}} = \{(x{\cdot}i, x{\cdot}(i+1)) \mid x{\cdot}(i+1) \in \mathsf{pos}(\mathbf{t})\}$$

$$\llbracket \varphi \wedge \psi \rrbracket^{\mathbf{t}} = \llbracket \varphi \rrbracket^{\mathbf{t}} \cap \llbracket \psi \rrbracket^{\mathbf{t}} \qquad\qquad \llbracket \alpha^* \rrbracket^{\mathbf{t}} = \text{the reflexive transitive closure of } \llbracket \alpha \rrbracket^{\mathbf{t}}$$

$$\llbracket a \rrbracket^{\mathbf{t}} = \{x \in \mathsf{pos}(\mathbf{t}) \mid \mathbf{a}(x) = a\} \qquad \llbracket \alpha\beta \rrbracket^{\mathbf{t}} = \{(x,z) \mid \text{ there exists } y \text{ such that}$$

$$\llbracket \neg\varphi \rrbracket^{\mathbf{t}} = \mathsf{pos}(\mathbf{t}) \setminus \llbracket \varphi \rrbracket^{\mathbf{t}} \qquad\qquad\qquad (x,y) \in \llbracket \alpha \rrbracket^{\mathbf{t}}, (y,z) \in \llbracket \beta \rrbracket^{\mathbf{t}}\}$$

$$\llbracket \langle \alpha \rangle \rrbracket^{\mathbf{t}} = \{x \in \mathsf{pos}(\mathbf{t}) \mid \exists y.(x,y) \in \llbracket \alpha \rrbracket^{\mathbf{t}}\} \qquad \llbracket [\varphi] \rrbracket^{\mathbf{t}} = \{(x,x) \mid x \in \llbracket \varphi \rrbracket^{\mathbf{t}}\}$$

$$\llbracket \langle \alpha{=}\beta \rangle \rrbracket^{\mathbf{t}} = \{x \in \mathsf{pos}(\mathbf{t}) \mid \exists y, z.(x,y) \in \llbracket \alpha \rrbracket^{\mathbf{t}}, \quad \llbracket \langle \alpha{\neq}\beta \rangle \rrbracket^{\mathbf{t}} = \{x \in \mathsf{pos}(\mathbf{t}) \mid \exists y, z.(x,y) \in \llbracket \alpha \rrbracket^{\mathbf{t}},$$

$$(x,z) \in \llbracket \beta \rrbracket^{\mathbf{t}}, \mathbf{d}(y) = \mathbf{d}(z)\} \qquad\qquad (x,z) \in \llbracket \beta \rrbracket^{\mathbf{t}}, \mathbf{d}(y) \neq \mathbf{d}(z)\}$$

Figure 7: Semantics of $\mathsf{regXPath}(\mathfrak{F},=)$ for a data tree $\mathbf{t} = \mathbf{a} \otimes \mathbf{d}$.

with a counting ability. By $\mathsf{regXPath}(\mathcal{O},=)$ we refer to the enriched language where path expressions are extended by allowing the Kleene star on *any* path expression.

$$\alpha, \beta ::= o \mid [\varphi] \mid \alpha\beta \mid \alpha \cup \beta \mid \alpha^* \qquad o \in \mathcal{O} \cup \{\varepsilon\}$$

Let $\mathbf{t}$ be a data tree. The semantics of XPath is defined as the set of elements (in the case of node expressions) or pairs of elements (in the case of path expressions) selected by the expression. The data aware expressions are the cases $\langle \alpha = \beta \rangle$ and $\langle \alpha \neq \beta \rangle$. The formal definition of its semantics is in Figure 7. We write $\mathbf{t} \models \varphi$ to denote $\llbracket \varphi \rrbracket^{\mathbf{t}} \neq \emptyset$, and in this case we say that $\mathbf{t}$ *satisfies* φ.

Example 6.1. In the model of Figure 1 on page 7,

$$\llbracket \, \langle \, \downarrow^*[b \, \wedge \langle \downarrow[b] \neq \downarrow[b]\rangle] \, \rangle \, \rrbracket^{\mathbf{t}} = \{\epsilon, \, 1, \, 1{\cdot}2\}.$$

We define $\mathsf{sub}(\varphi)$ to denote the set of all substrings of φ which are formulæ, $\mathsf{psub}(\varphi) := \{\alpha \mid \alpha \in \mathsf{sub}(\varphi), \alpha \text{ is a path expression}\}$, and $\mathsf{nsub}(\varphi) := \{\psi \mid \psi \in \mathsf{sub}(\varphi), \psi \text{ is a node expression}\}$.

6.1. **Key constraints.** It is worth noting that $\mathsf{XPath}(\mathfrak{F},=)$ —contrary to $\mathsf{XPath}^\varepsilon(\mathfrak{F},=)$— can express unary *key constraints*. That is, whether for some symbol a, all the a-elements in the tree have different data values.

Lemma 6.2. *For every $a \in \mathbb{A}$ let $key(a)$ be the property over a data tree $\mathbf{t} = \mathbf{a} \otimes \mathbf{d}$: "For every two different positions $x, x' \in \mathsf{pos}(\mathbf{t})$ of the tree, if $\mathbf{a}(x) = \mathbf{a}(x') = a$, then $\mathbf{d}(x) \neq \mathbf{d}(x')$." Then, $key(a)$ is expressible in $\mathsf{XPath}(\mathfrak{F},=)$ for any a.*

Proof. It is easy to see that the *negation* of this property can be tested by first *guessing* the closest common ancestor of two different a-elements with equal datum in the underlying first-child next-sibling binary tree. At this node, we verify the presence of two a-nodes with equal datum, one accessible with a "$\downarrow^*$" relation and the other with a compound "$\rightarrow^+\downarrow^*$" relation (hence the nodes are different). The expressibility of the property then follows from the logic being closed under negation. The reader can check that the following formula expresses the property

$$key(a) \equiv \neg\langle \, \downarrow^*[\, \langle \varepsilon[a] = \downarrow^+[a]\rangle \vee \langle \downarrow^*[a] = \rightarrow^+\downarrow^*[a]\rangle \,] \, \rangle$$

where '$\downarrow^+$' = '$\downarrow\downarrow^*$' and '$\rightarrow^+$' = '$\rightarrow\rightarrow^*$'. $\qquad\qquad\square$

Note that while ATRA cannot express, for instance, that there are two different nodes with the same data value, ATRA(guess) can express it. However, on the other hand, ATRA(guess, spread) cannot express the *negation* of the property.

Lemma 6.3. *The class* ATRA(guess, spread) *cannot express the property "all the data values of the data tree are different".*

Proof. Ad absurdum, suppose that there exists an automaton $\mathcal{A}$ expressing the property, and let Q be its set of states. Notice that for any accepting run of minimal length there are no more that $f(|Q|, n)$ consecutive ϵ-transitions (*i.e.*, $\twoheadrightarrow$ transitions that are fired by an underlying $\rightarrow_\epsilon$ transition on node configurations), for some fixed function f, where n is the number of moving transitions from the initial configuration. This is because of the rdc property of $(\mathcal{T}_{\text{ATRA}}, \twoheadrightarrow)$ with respect to $(\mathcal{T}_{\text{ATRA}}, \leq_\wp)$ (Lemma 5.7) and the fact that $\precsim$ preserves current position's letter and datum. Using this, one can see that, if there is an accepting run $\mathcal{S}_1 \twoheadrightarrow \cdots \twoheadrightarrow \mathcal{S}_N$ on a data tree $\mathbf{t}$ containing non-moving $\mathcal{S}_i \twoheadrightarrow \cdots \twoheadrightarrow \mathcal{S}_{i+k}$ so that $\mathcal{S}_i \leq_\wp \mathcal{S}_{i+k}$, then there are $\mathcal{S}'_{i+k+1} \leq_\wp \mathcal{S}_{i+k+1}, \ldots, \mathcal{S}'_N \leq_\wp \mathcal{S}_N$ so that $\mathcal{S}_1 \twoheadrightarrow \cdots \twoheadrightarrow \mathcal{S}_i \twoheadrightarrow \mathcal{S}'_{i+k+1} \twoheadrightarrow \cdots \twoheadrightarrow \mathcal{S}'_N$ is an accepting run on $\mathbf{t}$. Since $\leq_\wp$ is a wqo this must always happen for a sufficiently large k, computable from $\mathcal{A}$ and i.

Consider a data tree $\mathbf{t}$ with

$$\mathsf{pos}(\mathbf{t}) = \{\epsilon\} \cup \{1, 2, 3, \ldots, N\} \cup \{11, 111, \ldots, \underbrace{111 \cdots 1}_{N \text{ times}}\}$$

for $N = f(|Q|, 0) + f(|Q|, 1) + 4$ That is, the root of the tree has N children, and the first child has a long branch with N nodes. All positions of $\mathbf{t}$ have different data values, and they all carry the same label, say a.

Consider a minimal accepting run of $\mathcal{A}$ on $\mathbf{t}$, $\mathcal{S}_1 \twoheadrightarrow \cdots \twoheadrightarrow \mathcal{S}_n$. Let $\mathcal{S}_i$ be the first tree configuration containing a node configuration with position 2. That is, the first configuration after the second moving transition. Since there are at most $f(|Q|, 0) + f(|Q|, 1)$ non-moving transitions before $\mathcal{S}_i$ in the run, we have that $i \leq f(|Q|, 0) + f(|Q|, 1) + 2$. In particular, this means that $\mathcal{S}_i$ cannot contain more than $f(|Q|, 0) + f(|Q|, 1) + 2$ different data values. Therefore, there is a position x with $2 \leq x \leq N$ and a position y with $y \succeq 1$ such that neither $\mathbf{d}(x)$ or $\mathbf{d}(y)$ are in $data(\mathcal{S}_i)$. (Remember that by definition of $\mathbf{t}$, $\mathbf{d}(x) \neq \mathbf{d}(y)$.) This is because there are $N - 1 = f(|Q|, 0) + f(|Q|, 1) + 3$ different possible data values of x and for y.

Consider $\mathbf{t}'$ as the result of replacing $\mathbf{d}(x)$ by $\mathbf{d}(y)$ in $\mathbf{t}$. Clearly, $\mathbf{t}'$ does not have the property "all the data values of the data tree are different". Now, consider the run obtained as the result of replacing $\mathbf{d}(x)$ by $\mathbf{d}(y)$ in $\mathcal{S}_1, \ldots, \mathcal{S}_n$. Note that this is still a run, and it is still accepting. Therefore $\mathcal{A}$ accepts $\mathbf{t}$ and thus $\mathcal{A}$ does not express the property. $\square$

6.2. **Satisfiability problem.** This section is mainly dedicated to the decidability of the satisfiability problem for $\mathsf{XPath}(\mathfrak{F}, =)$, known as 'forward-XPath'. This is proved by a reduction to the emptiness problem of the automata model ATRA(guess, spread) introduced in Section 5.

[JL11] shows that ATRA captures the fragment $\mathsf{XPath}^\varepsilon(\mathfrak{F}, =)$. It is immediate to see that ATRA can also easily capture the Kleene star operator on any path formula, obtaining decidability of $\mathsf{regXPath}^\varepsilon(\mathfrak{F}, =)$. However, these decidability results cannot be further generalized to the full unrestricted forward fragment $\mathsf{XPath}(\mathfrak{F}, =)$ as ATRA is not powerful enough to capture the full expressivity of the logic. Indeed, while $\mathsf{XPath}(\mathfrak{F}, =)$ can
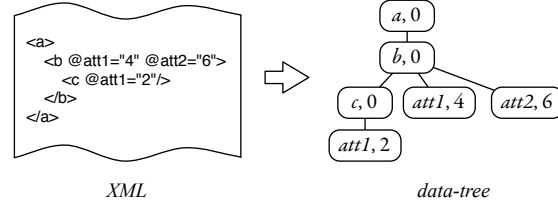
Figure 8: From XML documents to data-trees.

express that all the data values of the leaves are different, ATRA(guess, spread) cannot (Lemma 6.3). Although ATRA(guess, spread) cannot capture XPath($\mathfrak{F}$, =), in the sequel we show that there exists a reduction from the satisfiability of regXPath($\mathfrak{F}$, =) to the emptiness of ATRA(guess, spread), and hence that the former problem is decidable. This result settles an open question regarding the decidability of the satisfiability problem for the forward-XPath fragment XPath($\mathfrak{F}$, =). The main results that will be shown in Section 6.4 are the following.

Theorem 6.4. *Satisfiability of* regXPath($\mathfrak{F}$, =) *in the presence of DTDs (or any regular language) and unary key constraints is decidable, non-primitive-recursive.*

And hence the next corollary follows from the logic being closed under boolean operations.

Corollary 6.5. *The query containment and the query equivalence problems are decidable for* XPath($\mathfrak{F}$, =).

Moreover, these decidability results hold for regXPath($\mathfrak{F}$, =) and even for two extensions:
- a navigational extension with *upward* axes (in Section 6.5), and
- a generalization of the data tests that can be performed (in Section 6.6).

6.3. **Data trees and XML documents.** Although our main motivation for working with trees is related to static analysis of logics for XML documents, we work with *data trees*, being a simpler formalism to work with, from where results can be transferred to the class of XML documents. We discuss briefly how all the results we give on XPath over data trees, also hold for the class of XML documents.

While a data tree has *one* data value for each node, an XML document may have *several* attributes at a node, each with a data value. However, every attribute of an XML element can be encoded as a child node in a data tree labeled by the attribute's name, as in Figure 8. This coding can be enforced by the formalisms we present below, and we can thus transfer all the decidability results to the class of XML documents. In fact, it suffices to demand that all the attribute symbols can only occur at the leaves of the data tree and to interpret attribute expressions like '@*attrib1*' of XPath formulæ as child path expressions '↓[*attrib1*]'.

6.4. **Decidability of forward XPath.** This section is devoted to the proof of the following statement.

Proposition 6.6. *For every* $\eta \in$ regXPath($\mathfrak{F}$, =) *there is a computable* ATRA(guess, spread) *automaton* $\mathcal{A}$ *such that* $\mathcal{A}$ *is nonempty iff* η *is satisfiable.*
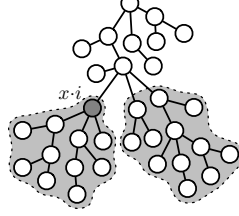
Figure 9: The disjoint values property states that for every position $x{\cdot}i$, the intersection of the grey zones is present in the last configuration for $x{\cdot}i$ appearing in the run.

Markedly, the $\mathsf{ATRA}(\mathsf{guess},\mathsf{spread})$ class does not capture $\mathsf{regXPath}(\mathfrak{F},=)$. However, given a formula η, it is possible to construct an automaton that tests a property that guarantees the existence of a data tree verifying η.

Disjoint values property. To show the above proposition, we need to work with runs with the *disjoint values property* as stated next.

Definition 6.7. A run $\mathcal{S}_1 \twoheadrightarrow \cdots \twoheadrightarrow \mathcal{S}_n$ on a data tree $\mathbf{t}$ has the **disjoint values property** if for every $x{\cdot}i \in \mathsf{pos}(\mathbf{t})$ and ρ a *moving* node configuration of the run with position $x{\cdot}i$, then

$$data(\mathbf{t}|_{x{\cdot}i}) \ \cap \bigcup_{\substack{x{\cdot}j \in \mathsf{pos}(\mathbf{t}) \\ j > i}} data(\mathbf{t}|_{x{\cdot}j}) \ \subseteq \ data(\rho) \ .$$

Figure 9 illustrates this property.

The proof of Proposition 6.6 can be sketched as follows:
(1) We show that for every nonempty automaton $\mathcal{A} \in \mathsf{ATRA}(\mathsf{guess},\mathsf{spread})$ there is an accepting run on a data tree with the disjoint values property.
(2) We give an effective translation from an arbitrary forward XPath formula η to an $\mathsf{ATRA}(\mathsf{guess},\mathsf{spread})$ automaton $\mathcal{A}$ such that
   - any tree accepted by a run of the automaton $\mathcal{A}$ with the disjoint values property verifies the XPath formula η, and
   - any tree verified by the formula η is accepted by a run of the automaton $\mathcal{A}$ with the disjoint values property.

We start by proving the disjoint values property normal form.

Proposition 6.8. *For any nonempty automaton $\mathcal{A} \in \mathsf{ATRA}(\mathsf{guess},\mathsf{spread})$ there exists an accepting run over some data tree with the disjoint values property.*

Proof. Given any accepting run $\mathcal{S}_1 \twoheadrightarrow \cdots \twoheadrightarrow \mathcal{S}_n$ on a data tree $\mathbf{t} = \mathbf{a} \otimes \mathbf{d}$, we show how to modify the run and the tree in order to satisfy the disjoint values property. We only need to replace some of the data values, so that the resulting tree and accepting run will be essentially the same.

The idea is as follows. For a given position $x{\cdot}k$ of the tree, we consider the moving node configuration of the run at position $x{\cdot}k$, and we replace all the data values from $\mathbf{t}|_{x{\cdot}k}$ by fresh ones except for those present in the node configuration, obtaining a new data tree $\mathbf{t}'$. We also make the same replacement of data values for all node configurations in the run

of nodes below $x \cdot k$. Thus, we end up with a modified data tree $\mathbf{t}'$ and accepting run that satisfy the disjoint values property at $x \cdot k$. That is, such that

$$data(\mathbf{t}'|_{x \cdot k}) \cap \bigcup_{\substack{x \cdot j \in \mathsf{pos}(\mathbf{t}) \\ j > k}} data(\mathbf{t}'|_{x \cdot j}) \subseteq data(\rho_{x \cdot k})$$

where $\rho_{x \cdot k}$ is the moving node configuration at position $x \cdot k$ of the run. If we repeat this procedure for all nodes of the tree, we obtain a run and tree with the disjoint values property. Next, we formalize this transformation.

Take any $x \cdot k \in \mathsf{pos}(\mathbf{t})$, and let $\rho_{x \cdot k} \in \mathcal{S}_i$ for some i be such that $\rho_{x \cdot k}$ is a moving node configuration with position $x \cdot k$. Consider any injective function

$$f \ : \ data(\mathbf{t}|_{x \cdot k}) \setminus data(\rho_{x \cdot k}) \ \rightarrow \ \mathbb{D} \setminus data(\mathbf{t})$$

and let $\hat{f} : data(\mathbf{t}|_{x \cdot k}) \rightarrow \mathbb{D}$ be such that $\hat{f}(d) = d$ if $d \in data(\rho_{x \cdot k})$, or $\hat{f}(d) = f(d)$ otherwise. Note that $\hat{f}$ is injective. Let us consider then $\mathcal{S}'_1, \ldots, \mathcal{S}'_n$ where $\mathcal{S}'_j$ consists in replacing every node configuration $\rho \in \mathcal{S}_j$ with $h(\rho)$, where[6]

$$h(\rho) = \begin{cases} \hat{f}(\rho) & \text{if } \rho \text{ has a position } y \succ x \cdot k \\ \rho & \text{otherwise.} \end{cases}$$

Take $\mathbf{t}'$ to be the data tree that results from the replacement in $\mathbf{t}$ of every data value of a position $y \succ x \cdot k$ by $\hat{f}(\mathbf{d}(y))$.

Claim 1.
$$data(\mathbf{t}'|_{x \cdot k}) \cap \bigcup_{\substack{x \cdot j \in \mathsf{pos}(\mathbf{t}) \\ j > k}} data(\mathbf{t}'|_{x \cdot j}) \subseteq data(\rho_{x \cdot k}) \ .$$

Proof. The only difference between $\mathbf{t}$ and $\mathbf{t}'$ is in the data values below $x \cdot k$. Any node y of $\mathbf{t}'$ which is below $x \cdot k$ contains a data value $\hat{f}(\mathbf{d}(y))$ such that

- $\hat{f}(\mathbf{d}(y))$ is in $data(\rho_{x \cdot k})$, or
- $\hat{f}(\mathbf{d}(y))$ is not in $data(\mathbf{t})$, and therefore it is not in $\bigcup_{\substack{x \cdot j \in \mathsf{pos}(\mathbf{t}) \\ j > k}} data(\mathbf{t}'|_{x \cdot j})$.

$\square$

Claim 2. $\mathcal{S}'_1 \twoheadrightarrow \cdots \twoheadrightarrow \mathcal{S}'_n$ *is an accepting run of $\mathcal{A}$ over $\mathbf{t}'$.*

Proof. Take any leaf y which is rightmost (*i.e.*, with no siblings to its right) and consider the sequence of node configurations $\rho_1 \in \mathcal{S}_1, \ldots, \rho_n \in \mathcal{S}_n$ that are ancestors of y in the first-child next-sibling underlying tree structure. There is exactly one configuration in each $\mathcal{S}_i$ due to Remark 5.2. This is the 'sub-run' that leads to y: for every ρ_i, ρ_{i+1} either $\rho_i = \rho_{i+1}$ or $\rho_i \rightarrow \rho_{i+1}$. Let $\rho'_j = h(\rho_j)$ for all $j \in [n]$.

- If $y \succeq x \cdot k$, take ℓ to be the last index such that $\rho_\ell = \rho_{x \cdot k}$ (there must be one). Note that $\rho'_j = \hat{f}(\rho_j)$ for every $\ell < j \leq n$. The sequence $\rho'_{\ell+1} \in \mathcal{S}'_{\ell+1}, \ldots, \rho'_n \in \mathcal{S}'_n$ is isomorphic, modulo renaming of data values, to $\rho_{\ell+1}, \ldots, \rho_n$ since $\hat{f}$ is injective. We then have that $\rho'_1, \ldots, \rho'_\ell, \ldots \rho'_n$ is a correct run on node configurations, since
  - $\rho'_1, \ldots, \rho'_\ell$ is equal to $\rho_1, \ldots, \rho_\ell$ (it is not modified by h),

---

[6]By $\hat{f}(\rho)$ we denote the replacement of every data value d by $\hat{f}(d)$ in ρ.

- $\rho'_{\ell+1}, \ldots, \rho'_n$ is isomorphic to $\rho_{\ell+1}, \ldots, \rho_n$ (we apply an injection $\hat{f}$ to every data value), and
- the pair $(\rho'_\ell, \rho'_{\ell+1})$ is isomorphic to $(\rho_\ell, \rho_{\ell+1})$, as
  * $\rho'_\ell = f(\rho_{x \cdot k}) = \rho_{x \cdot k} = \rho_\ell$,
  * $\rho'_{\ell+1} = \hat{f}(\rho_{\ell+1})$ is isomorphic to $\rho_{\ell+1}$, and
  * $data(\rho'_\ell) \cap data(\rho'_{\ell+1}) = data(\rho_\ell) \cap data(\rho_{\ell+1})$ since $\hat{f}$ is the identity on $data(\rho_\ell)$, and does not send any data value from $\mathbb{D} \setminus data(\rho_\ell)$ to $data(\rho_\ell)$.
- If $y \not\geq x \cdot k$, then nothing was modified: $\rho'_1 = \rho_1 \in \mathcal{S}_1, \ldots, \rho'_n = \rho_n \in \mathcal{S}_n$.

In any case, we have that $\rho'_1, \ldots, \rho'_n$ is a correct run on node configurations. This means that the modified data values are innocuous for the run. As the structure of the run is not changed, this implies that $\mathcal{S}'_1 \twoheadrightarrow \cdots \twoheadrightarrow \mathcal{S}'_n$ is an accepting run (that verifies the disjoint values property for $x \cdot k$ by the previous Claim). $\qquad\square$

If we perform the same procedure for every position of the tree, we end up with an accepting run and tree with the disjoint values property. $\qquad\square$

We define a translation from $\mathsf{regXPath}(\mathfrak{F}, =)$ formulæ to $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$. Let η be a $\mathsf{regXPath}(\mathfrak{F}, =)$ formula and let $\mathcal{A}$ be the corresponding $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ automaton defined by the translation. We show that (i) if a data tree $\mathbf{t}$ is accepted by $\mathcal{A}$ by a run verifying the disjoint values property, then $\mathbf{t} \models \eta$, and in turn (ii) if $\mathbf{t} \models \eta$, then $\mathbf{t}$ is accepted by $\mathcal{A}$. Thus, by the disjoint values normal form (Proposition 6.8) we obtain our main result of Proposition 6.6, which by decidability of $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ (Theorem 5.5) implies that the satisfiability problem for $\mathsf{regXPath}(\mathfrak{F}, =)$ is decidable.

Normal form. For succinctness and simplicity of the translation, we assume that η is in a normal form such that the $\downarrow$-axis is interpreted as the *leftmost* child. To obtain this normal form, it suffices to replace every appearance of '$\downarrow$' by '$\downarrow \rightarrow^*$'. Also, we assume that every data test subformula of the form $\langle \alpha = \beta \rangle$ or $\langle \alpha \neq \beta \rangle$ is such that $\alpha = \varepsilon$, $\alpha = \downarrow \gamma$, or $\alpha = \rightarrow \gamma$ for some γ, and idem for β. This will simplify the translation and the proofs for correctness. It is easy to see that every expression $\langle \alpha = \beta \rangle$ can be effectively transformed into a disjunction of formulæ of the aforementioned form.

The translation. Let η be a $\mathsf{regXPath}(\mathfrak{F}, =)$ node expression of the above form in negation normal form (nnf for short). Let $\mathbf{t}$ be a data tree and let $z_0 \preceq_{fcns} z_n$ be two positions of $\mathbf{t}$ at distance $n \in \mathbb{N}_0$ such that, if $z_0 \neq z_n$,

$$z_0 \prec_{fcns} z_1 \prec_{fcns} \cdots \prec_{fcns} z_{n-1} \prec_{fcns} z_n$$

for some $z_1, \ldots, z_{n-1}$, we define $str(z_0, z_n) \in \mathbb{A}_\eta^*$ to be the string $a_1 S_1 \cdots a_{n-1} S_{n-1}$ where a_i is either $\rightarrow$ or $\downarrow$ depending on the relation between z_{i-1} and z_i, and $S_i = \{\psi \in \mathsf{nsub}(\eta) \mid z_i \in [\![\psi]\!]^{\mathbf{t}}\}$. If $n = 0$, then $str(z_0, z_0) = \epsilon$. Note that $str(z_0, z_n)$ does not take into account z_0, since we are working under the normal form where no path subformulæ makes tests on the node where they start. For every path expression $\alpha \in \mathsf{psub}(\eta)$, consider a deterministic complete finite automaton $\mathcal{H}_\alpha$ over the alphabet $\mathbb{A}_\eta = 2^{\mathsf{nsub}(\eta)} \cup \{\downarrow, \rightarrow\}$ which corresponds to that regular expression, in the sense that the following claim holds.

Claim 3. *For every $\alpha \in \mathsf{psub}(\eta)$, and $x \preceq_{fcns} y$, we have that $\mathcal{H}_\alpha$ accepts $str(x, y)$ if, and only if, $(x, y) \in [\![\alpha]\!]^{\mathbf{t}}$.*

$$\mathsf{nnf}(\varphi \wedge \psi) := \mathsf{nnf}(\varphi) \wedge \mathsf{nnf}(\psi) \qquad\qquad \mathsf{nnf}(\varphi \vee \psi) := \mathsf{nnf}(\varphi) \vee \mathsf{nnf}(\psi)$$

$$\mathsf{nnf}(\neg(\varphi \wedge \psi)) := \mathsf{nnf}(\neg\varphi) \vee \mathsf{nnf}(\neg\psi) \qquad \mathsf{nnf}(\neg(\varphi \vee \psi)) := \mathsf{nnf}(\neg\varphi) \wedge \mathsf{nnf}(\neg\psi)$$

$$\mathsf{nnf}(\alpha\,\beta) := \mathsf{nnf}(\alpha)\,\mathsf{nnf}(\beta) \qquad\qquad\qquad \mathsf{nnf}([\varphi]) := [\mathsf{nnf}(\varphi)]$$

$$\mathsf{nnf}(\alpha^*) := (\mathsf{nnf}(\alpha))^* \qquad\qquad\qquad \mathsf{nnf}(o) := o \quad o \in \mathfrak{F}$$

$$\mathsf{nnf}(\langle \alpha \circledast \beta \rangle) := \langle \mathsf{nnf}(\alpha) \circledast \mathsf{nnf}(\beta) \rangle \qquad \mathsf{nnf}(\neg\langle \alpha \circledast \beta \rangle) := \neg\langle \mathsf{nnf}(\alpha) \circledast \mathsf{nnf}(\beta) \rangle$$

$$\mathsf{nnf}(\mathsf{a}) := \mathsf{a} \qquad\qquad\qquad\qquad \mathsf{nnf}(\neg\mathsf{a}) := \neg\mathsf{a}$$

$$\mathsf{nnf}(\neg\neg\varphi) := \mathsf{nnf}(\varphi) \qquad\qquad\qquad \mathsf{nnf}(\langle\alpha\rangle) := \langle\mathsf{nnf}(\alpha)\rangle$$

Table 1: Definition of the Negation Normal Form for XPath.

We assume the following names of its components: $\mathcal{H}_\alpha = \langle \mathbb{A}_\eta, \delta_\alpha, Q_\alpha, 0, F_\alpha \rangle$, where $Q_\alpha \subseteq \mathbb{N}_0$ is the finite set of states and $0 \in Q_\alpha$ is the initial state. We assume that Q_α is partitioned into *moving* and *testing* states, such that in every accepting run the states of the run alternate between moving and testing states, starting in the moving state 0.

We next show how to translate η into an ATRA(guess, spread) automaton $\mathcal{A}$. For the sake of readability we define the transitions as positive boolean combinations of $\vee$ and $\wedge$ over the set of basic tests and states. Any of these —take for instance $\delta(q) = (\mathsf{store}(q_1) \wedge \triangledown q_2) \vee (q_3 \wedge \bar{\mathsf{a}})$— can be rewritten into an equivalent ATRA with at most one boolean connector per transition (as in Definition 5.1) in polynomial time. The most important cases are those relative to the following data tests:

(1) $\langle \alpha = \beta \rangle$ (2) $\langle \alpha \neq \beta \rangle$ (3) $\neg\langle \alpha = \beta \rangle$ (4) $\neg\langle \alpha \neq \beta \rangle$

We define the ATRA(guess, spread) automaton

$$\mathcal{A} := \langle \mathbb{A}, Q, (\!|\eta|\!), \delta \rangle$$

with

$$Q := \{(\!|\varphi|\!), (\!|\alpha|\!)_i^\circledast, (\!|\alpha|\!)_{\text{test},i}^\circledast, (\!|\alpha|\!)_{\mathsf{F}}^\circledast, (\!|\alpha,\beta|\!)_{i,j}^\circledast, (\!|\alpha,\beta|\!)_{\text{test},i,j}^\circledast \mid \varphi \in \mathsf{nsub}^\neg(\eta),$$
$$\alpha, \beta \in \mathsf{psub}^\neg(\eta), \circledast \in \{=, \neq, \neg=, \neg\neq\}, i \in Q_\alpha, j \in Q_\beta\}$$

where $\mathsf{op}^\neg$ is the smallest superset of op closed under negation under nnf, *i.e.*, if $\varphi \in \mathsf{op}^\neg(\eta)$ then $\mathsf{nnf}(\neg\varphi) \in \mathsf{op}^\neg(\eta)$ (where nnf is defined as shown in Table 1). The idea is that a state $(\!|\varphi|\!)$ verifies the formula φ. A state $(\!|\alpha|\!)_i^=$ (resp. $(\!|\alpha|\!)_i^{\neq}$) verifies that there is a forward path in the tree ending at a node with the same (resp. different) data value as the one in the register, such that there exists a partial run of $\mathcal{H}_\alpha$ over such path that starts in a moving state i and ends in a final state. Similarly, a state $(\!|\alpha|\!)_{\text{test},i}^=$ or $(\!|\alpha|\!)_{\text{test},i}^{\neq}$ verifies the same when i is a testing state. A state $(\!|\alpha,\beta|\!)_{i,j}^=$ (resp. $(\!|\alpha,\beta|\!)_{i,j}^{\neq}$) verifies that there are *two* paths ending in two nodes with the same (resp. one equal, the other different) data value as that of the register; such that one path has a partial accepting run of $\mathcal{H}_\alpha$ starting in a moving state i, and the other has a partial accepting run of $\mathcal{H}_\beta$ starting in a moving state j. A state like $(\!|\alpha|\!)_{\mathsf{F}}^=$ is simply to mark that the run of $\mathcal{H}_\alpha$ on a path has ended, and the only remaining task is to test for equality of the data value with respect to the register. Finally, a state $(\!|\alpha|\!)_0^{\neg=}$ (resp. $(\!|\alpha|\!)_0^{\neg\neq}$) verifies that every node reachable by α has different (resp. equal) data than the register, and similar for the other universal states of the form $(\!| \cdots |\!)_{\cdots}^{\neg\cdots}$. We

first take care of the boolean connectors and the simplest tests.

$$\delta(\langle\!| a |\!\rangle) := a \qquad \delta(\langle\!| \varphi \vee \psi |\!\rangle) := \langle\!| \varphi |\!\rangle \vee \langle\!| \psi |\!\rangle \qquad \delta(\langle\!| \neg a |\!\rangle) := \bar{a} \qquad \delta(\langle\!| \varphi \wedge \psi |\!\rangle) := \langle\!| \varphi |\!\rangle \wedge \langle\!| \psi |\!\rangle$$

First, we define the transitions associated to each $\mathcal{H}_\alpha$, for $i \in Q_\alpha, \mathcal{C} \subseteq Q_\alpha, \circledast \in \{=, \neq\}$. Here, $\langle\!| \alpha |\!\rangle_\mathsf{F}$ holds at the endpoint of a path matching α.

$$\delta(\langle\!| \alpha |\!\rangle_i^\circledast) := \triangledown \langle\!| \alpha |\!\rangle_{\text{test},\delta_\alpha(\downarrow,i)}^\circledast \quad \vee \quad \triangleright \langle\!| \alpha |\!\rangle_{\text{test},\delta_\alpha(\rightarrow,i)}^\circledast \quad \vee \quad \bigvee_{i \in F_\alpha} \langle\!| \alpha |\!\rangle_\mathsf{F}^\circledast$$

$$\delta(\langle\!| \alpha |\!\rangle_{\text{test},i}^\circledast) := \bigvee_{S \subseteq \mathsf{nsub}(\alpha)} \left(\langle\!| \alpha |\!\rangle_{\delta_\alpha(S,i)}^\circledast \wedge \bigwedge_{\varphi \in S} \langle\!| \varphi |\!\rangle \right)$$

Next, we focus only on the data-aware test formulæ, since the tests $\langle \alpha \rangle$ and $\neg \langle \alpha \rangle$ are interpreted as the equivalent formulæ $\langle \alpha = \alpha \rangle$ and $\neg \langle \alpha = \alpha \rangle$ respectively. Using the guess operator, we can easily define the cases corresponding to the data test cases (1) and (2) as follows.

$$\delta(\langle\!| \alpha = \beta |\!\rangle) := \mathsf{guess}(\langle\!| \alpha, \beta |\!\rangle^=) \qquad \delta(\langle\!| \alpha, \beta |\!\rangle^=) := \langle\!| \alpha |\!\rangle_0^= \wedge \langle\!| \beta |\!\rangle_0^= \qquad \delta(\langle\!| \alpha |\!\rangle_\mathsf{F}^=) := \mathsf{eq}$$

$$\delta(\langle\!| \alpha \neq \beta |\!\rangle) := \mathsf{guess}(\langle\!| \alpha, \beta |\!\rangle^{\neq}) \qquad \delta(\langle\!| \alpha, \beta |\!\rangle^{\neq}) := \langle\!| \alpha |\!\rangle_0^= \wedge \langle\!| \beta |\!\rangle_0^{\neq} \qquad \delta(\langle\!| \alpha |\!\rangle_\mathsf{F}^{\neq}) := \overline{\mathsf{eq}}$$

The test case (4) involves also an *existential* quantification over data values. In fact, $\neg\langle \alpha \neq \beta \rangle$ means that either
 (i) there are no nodes reachable by α,
 (ii) there are no nodes reachable by β, or
(iii) there *exists* a data value d such that both
        (a) all elements reachable by α have datum d, and
        (b) all elements reachable by β have datum d.

$$\delta(\langle\!| \neg \alpha \neq \beta |\!\rangle) := \langle\!| \neg \langle \alpha \rangle |\!\rangle \vee \langle\!| \neg \langle \beta \rangle |\!\rangle \vee \mathsf{guess}(\langle\!| \alpha, \beta |\!\rangle^{\neg \neq})$$

$$\delta(\langle\!| \alpha, \beta |\!\rangle^{\neg \neq}) := \langle\!| \alpha |\!\rangle_0^{\neg \neq} \wedge \langle\!| \beta |\!\rangle_0^{\neg \neq} \qquad \delta(\langle\!| \alpha |\!\rangle_\mathsf{F}^{\neg \neq}) := \mathsf{eq} \qquad \delta(\langle\!| \alpha |\!\rangle_\mathsf{F}^{\neg =}) := \overline{\mathsf{eq}}$$

$$\delta(\langle\!| \alpha |\!\rangle_i^{\neg \circledast}) := (\bar{\triangledown}? \vee \triangledown \langle\!| \alpha |\!\rangle_{\text{test},\delta_\alpha(\downarrow,i)}^{\neg \circledast}) \wedge (\bar{\triangleright}? \vee \triangleright \langle\!| \alpha |\!\rangle_{\text{test},\delta_\alpha(\rightarrow,i)}^{\neg \circledast}) \quad \wedge \quad final$$

$$\text{where } \bar{\varphi} \text{ stands for } \mathsf{nnf}(\neg\varphi) \text{ and } final = \begin{cases} \langle\!| \alpha |\!\rangle_\mathsf{F}^{\neg \circledast} & \text{if } i \in F_\alpha, \\ true & \text{otherwise.} \end{cases}$$

$$\delta(\langle\!| \alpha |\!\rangle_{\text{test},i}^{\neg \circledast}) := \bigwedge_{S \subseteq \mathsf{nsub}(\alpha)} \left(\langle\!| \alpha |\!\rangle_{\delta_\alpha(S,i)}^{\neg \circledast} \wedge \bigwedge_{\varphi \in S} \langle\!| \varphi |\!\rangle \right)$$

The difficult part is the translation of the data test case (3). The main reason for this difficulty is the fact that $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ automata do not have the expressivity to make these kinds of tests. An expression $\neg\langle \alpha = \beta \rangle$ forces the set of data values reachable by an α-path and the set of those reachable by a β-path to be disjoint. We show that nonetheless the automaton can test for a condition that is equivalent to $\neg\langle \alpha = \beta \rangle$ if we assume that the run and tree have the disjoint values property.

Example 6.9. As an example, suppose that $\eta = \neg\langle \downarrow\alpha = \rightarrow\beta \rangle$ is to be checked for satisfiability. One obvious answer would be to test separately α and β. If both tests succeed, one can then build a model satisfying η out of the two witnessing trees by making sure they have disjoint sets of values. Otherwise, η is clearly unsatisfiable. Suppose now

that we have $\eta = \varphi \wedge \neg\langle \downarrow\alpha = \rightarrow\beta \rangle$, where φ is any formula with no data tests of type (3). One could build the automaton for φ and then ask for "$\mathsf{spread}(\langle\!\langle\downarrow\alpha\rangle\!\rangle_0^{\neg=} \vee \langle\!\langle\rightarrow\beta\rangle\!\rangle_0^{\neg=})$" in the automaton. This corresponds to the property "*for every data value d taken into account by the automaton (as a result of the translation of φ), either all elements reachable by α do not have datum d, or all elements reachable by β do not have datum d*". If φ contains a $\langle \alpha' = \beta' \rangle$ formula, this translates to a *guessing* of a witnessing data value d. Then, the use of spread takes care of this particular data value, and indeed of all other data values that were guessed to satisfy similar demands. In other words, it is *not* because of d that $\neg\langle \downarrow\alpha = \rightarrow\beta \rangle$ will be falsified. But then, the disjoint values property ensures that no pair of nodes accessible by α and β share the same datum. This is the main idea we encode next.

We define $\delta(\langle\!\langle \neg\langle\alpha = \beta\rangle\rangle\!\rangle) := \langle\!\langle\alpha,\beta\rangle\!\rangle_{0,0}^{\neg=}$. Given $\neg\langle\alpha = \beta\rangle$, the automaton systematically looks for the closest common ancestor of every pair (x, y) of nodes accessible by α and β respectively, and tests, for every data value d present in the node configuration, that either (1) all data values accessible by the remaining path of α are different from d, or (2) all data values accessible by the remaining path of β are different from d.

$$\delta(\langle\!\langle\alpha,\beta\rangle\!\rangle_{i,j}^{\neg=}) := \mathsf{spread}\big(\langle\!\langle\alpha\rangle\!\rangle_i^{\neg=} \vee \langle\!\langle\beta\rangle\!\rangle_j^{\neg=}\big)$$
$$\wedge \quad (\bar{\nabla}? \vee \nabla\langle\!\langle\alpha,\beta\rangle\!\rangle_{\text{test},\delta_\alpha(\downarrow,i),\delta_\beta(\downarrow,j)}^{\neg=}) \quad \wedge \quad (\bar{\rhd}? \vee \rhd\langle\!\langle\alpha,\beta\rangle\!\rangle_{\text{test},\delta_\alpha(\rightarrow,i),\delta_\beta(\rightarrow,j)}^{\neg=})$$
$$\wedge \quad \textit{final}_\alpha \quad \wedge \quad \textit{final}_\beta$$

where $\textit{final}_\alpha = \begin{cases} \mathsf{store}(\langle\!\langle\beta\rangle\!\rangle_j^{\neg=}) & \text{if } i \in F_\alpha, \\ \textit{true} & \text{otherwise,} \end{cases}$

and $\textit{final}_\beta = \begin{cases} \mathsf{store}(\langle\!\langle\alpha\rangle\!\rangle_i^{\neg=}) & \text{if } j \in F_\beta, \\ \textit{true} & \text{otherwise.} \end{cases}$

$$\delta(\langle\!\langle\alpha,\beta\rangle\!\rangle_{\text{test},i,j}^{\neg=}) := \bigwedge_{S\subseteq\mathsf{nsub}(\alpha)} \Big(\langle\!\langle\alpha,\beta\rangle\!\rangle_{\delta_\alpha(S,i),\delta_\beta(S,j)}^{\neg=} \wedge \bigwedge_{\varphi\in S} \langle\!\langle\varphi\rangle\!\rangle\Big)$$

The first line of $\delta(\langle\!\langle\alpha,\beta\rangle\!\rangle_{i,j}^{\neg=})$ corresponds to the tests (1) and (2) above, and the third line corresponds to the cases where $x \preceq_{fcns} y$ and $y \preceq_{fcns} x$.

Next we show the correctness of this translation. We say that $\mathcal{A}$ has an accepting run from a thread (q, d) on $\mathbf{t}|_x$ if there is a sequence of tree configurations $\mathcal{S}_1 \twoheadrightarrow \cdots \twoheadrightarrow \mathcal{S}_n$ such that $\mathcal{S}_1$ is at position x and contains (q, d), and $\mathcal{S}_n$ is accepting. We say that there is a *dvp-run* if $\mathcal{S}_1 \twoheadrightarrow \cdots \twoheadrightarrow \mathcal{S}_n$ has the disjoint values property.

Lemma 6.10. *For any data tree* $\mathbf{t}$,
- $(\Rightarrow)$ *if* $\mathbf{t} \models \eta$ *then* $\mathcal{A}$ *accepts* $\mathbf{t}$ *with a dvp-run, and*
- $(\Leftarrow)$ *if* $\mathcal{A}$ *accepts* $\mathbf{t}$ *with a dvp-run, then* $\mathbf{t} \models \eta$.

Proof. Let $\mathbf{t} = \mathbf{a}\otimes\mathbf{d}$. We show that for every subformula $\varphi \in \mathsf{sub}(\eta)$ and position $x \in \mathsf{pos}(\mathbf{t})$,

$$\text{if } \mathbf{t}|_x \models \varphi \text{ then } \mathcal{A} \text{ has an accepting dvp-run from } (\langle\!\langle\varphi\rangle\!\rangle, d) \text{ on } \mathbf{t}|_x \text{ for any } d \in \mathbb{D}. \quad (\dagger)$$

And conversely,

$$\text{if } \mathcal{A} \text{ has an accepting dvp-run from } (\langle\!\langle\varphi\rangle\!\rangle, d) \text{ on } \mathbf{t}|_x \text{ for some } d \in \mathbb{D}, \text{ then } \mathbf{t}|_x \models \varphi. \quad (\ddagger)$$

We suppose that η is in nnf. We proceed by induction on the lexicographic order of $(f(x), |\varphi|)$, where $f(x)$ is the length of a maximal path from x to a leaf in the fcns coding

(roughly, the height of $\mathbf{t}|_x$). The base cases when $\varphi = a$ or $\varphi = \neg a$ are easy. Suppose then that the proposition is true for all position $y \succ_{fcns} x$, and for all $\psi \in \mathsf{sub}(\eta)$, $|\psi| < |\varphi|$ at position x. The cases $\varphi = \varphi_1 \wedge \varphi_2$ or $\varphi = \varphi_1 \vee \varphi_2$ are straightforward from the inductive hypothesis. The following claim also follows from the inductive hypothesis.

Claim 4. *For any path subformula α of $\mathsf{psub}(\eta)$, and $y \succeq_{fcns} x'$,*

(a) if $(x, y) \in [\![\alpha]\!]^{\mathbf{t}}$ then there is an accepting dvp-run from $(\langle\!|\alpha|\!\rangle_0^=, \mathbf{d}(y))$ on $\mathbf{t}|_x$,

(b) if there is an accepting dvp-run from $(\langle\!|\alpha|\!\rangle_0^=, d)$ on $\mathbf{t}|_x$, then $(x, y) \in [\![\alpha]\!]^{\mathbf{t}}$ for some y such that $\mathbf{d}(y) = d$,

(c) if $(x, y) \in [\![\alpha]\!]^{\mathbf{t}}$, there is an accepting dvp-run from $(\langle\!|\alpha|\!\rangle_0^{\neq}, d)$ with $d \neq \mathbf{d}(y)$ on $\mathbf{t}|_x$,

(d) if there is an accepting dvp-run from $(\langle\!|\alpha|\!\rangle_0^{\neq}, d)$ on $\mathbf{t}|_x$, then $(x, y) \in [\![\alpha]\!]^{\mathbf{t}}$ for some y with $d \neq \mathbf{d}(y)$.

Proof. By induction, suppose that Claim 4 holds for all α, x', y such that $(f(x'), |\alpha|) <_{lex} (f(x), |\varphi|)$. As a consequence of the inductive hypothesis on (†), for all positions z with $x \prec_{fcns} z \preceq_{fcns} y$, and for every subformula $\psi \in \mathsf{nsub}(\eta)$, if $z \in [\![\psi]\!]^{\mathbf{t}}$ then there is an accepting dvp-run of $\mathcal{A}$ from $(\langle\!|\psi|\!\rangle, d)$ on $\mathbf{t}|_z$ for any $d \in \mathbb{D}$.

(a) Suppose first that $(x, y) \in [\![\alpha]\!]^{\mathbf{t}}$. Therefore, $\mathcal{H}_\alpha$ accepts $str(x, y)$ by Claim 3. Further, suppose that $S \in \mathbb{A}_\eta$ is the $(2i)$-th element of $str(x, y)$, and that z is the position of the i-th element in the path between x and y of the fcns coding. For any $\psi \in S$ we have that $z \in [\![\psi]\!]^{\mathbf{t}}$, and applying the inductive hypothesis on (†) we have that there is an accepting dvp-run from $(\langle\!|\psi|\!\rangle, d)$ on $\mathbf{t}|_z$. Hence, by definition of the transition of $\langle\!|\alpha|\!\rangle_0^=$, we have that there is an accepting dvp-run from $(\langle\!|\alpha|\!\rangle_0^=, d')$ on $\mathbf{t}|_x$ if we take $d' = \mathbf{d}(y)$.

(b) For any z with $x \prec_{fcns} z \preceq_{fcns} y$, if there is an accepting dvp-run from $(\langle\!|\psi|\!\rangle, d)$ on $\mathbf{t}|_z$, by inductive hypothesis on (‡), we have that $z \in [\![\psi]\!]^{\mathbf{t}}$. Therefore, if there is an accepting dvp-run from $(\langle\!|\alpha|\!\rangle^=, d)$ on $\mathbf{t}|_x$, by the definition of the transition relation there must be some $y \succeq_{fcns} x$ such that $str(x, y)$ is accepted by $\mathcal{H}_\alpha$ and further $\mathbf{d}(y) = d$. Hence, $(x, y) \in [\![\alpha]\!]^{\mathbf{t}}$.

An identical reasoning applies to show (c) and (d). $\qquad\qquad\square$

($\Rightarrow$) Suppose that $\mathbf{t}|_x \models \varphi$. We focus on the data test cases. If $\varphi = \langle \alpha = \beta \rangle$, then $\delta(\langle\!|\varphi|\!\rangle) = \mathsf{guess}(\langle\!|\alpha, \beta|\!\rangle^=)$. Since $\delta(\langle\!|\alpha, \beta|\!\rangle^=) = \langle\!|\alpha|\!\rangle_0^= \wedge \langle\!|\beta|\!\rangle_0^=$, we then have that for any data value d, there is an accepting run from $(\langle\!|\varphi|\!\rangle, d)$ on $\mathbf{t}|_x$ if there exists a data value $d' \in \mathbb{D}$ such that $(\langle\!|\alpha|\!\rangle_0^=, d')$ and $(\langle\!|\beta|\!\rangle_0^=, d')$ have accepting dvp-runs. Since $\mathbf{t}|_x \models \langle \alpha = \beta \rangle$, there must be two positions y and y' below x such that $\mathbf{d}(y) = \mathbf{d}(y')$ where $(x, y) \in [\![\alpha]\!]^{\mathbf{t}}$ and $(x, y') \in [\![\beta]\!]^{\mathbf{t}}$. By Claim 4.a there are accepting dvp-runs from $(\langle\!|\alpha|\!\rangle_0^=, d')$ and $(\langle\!|\beta|\!\rangle_0^=, d')$ on $\mathbf{t}|_x$. These two dvp-runs can be simply combined to build an accepting dvp-run from $(\langle\!|\varphi|\!\rangle, d)$. The case $\varphi = \langle \alpha \neq \beta \rangle$ is similar, using Claims 4.a and 4.c.

If $\varphi = \neg\langle \alpha = \beta \rangle$, then $\mathbf{t}|_x$ is such that for every pair of paths ending in some positions y, y', if the paths satisfy α and β respectively, then $\mathbf{d}(y) \neq \mathbf{d}(y')$. Since $\delta(\neg\langle \alpha = \beta \rangle) = \langle\!|\alpha, \beta|\!\rangle^{\neg=}$, we show that there is an accepting run from $(\langle\!|\alpha, \beta|\!\rangle^{\neg=}, d)$, for any $d \in \mathbb{D}$. Using the inductive hypothesis on all the nodes below x (as we did in Claim 4), note that every time that the automaton has an accepting dvp-run from $(\langle\!|\psi|\!\rangle, d)$ at x for some $d \in \mathbb{D}$, then $x \in [\![\psi]\!]^{\mathbf{t}}$. By definition of the transition relation on $\langle\!|\alpha, \beta|\!\rangle^{\neg=}$, this means that a sufficient condition for the automaton to have an accepting dvp-run from $(\langle\!|\alpha, \beta|\!\rangle^{\neg=}, d)$ is: for every node $z \succeq_{fcns} x$, if the states of $\mathcal{H}_\alpha, \mathcal{H}_\beta$ are respectively i, j after reading $str(x, z)$, then there is an accepting dvp-run from $(\mathsf{spread}(\langle\!|\alpha|\!\rangle_i^{\neg=} \vee \langle\!|\beta|\!\rangle_j^{\neg=}), d)$ on $\mathbf{t}|_z$. To simplify the argument, we can assume that the configuration contains *all* the data values of the tree and

therefore that *opspread* takes into account all possible data values (this is not a necessary condition, but certainly a sufficient one to have an accepting dvp-run). Also using the inductive hypothesis, there is an accepting dvp-run from $(\langle\!\langle\alpha\rangle\!\rangle_i^{\neg=}, d')$ on $\mathbf{t}|_z$ if for every node $z' \succeq_{fcns} z$ such that $\mathcal{H}_\alpha$ takes the state i to a final state after reading $str(z, z')$, we have $\mathbf{d}(z') \neq d'$. Therefore, there is an accepting dvp-run from $\mathsf{spread}\big((\langle\!\langle\alpha\rangle\!\rangle_i^{\neg=} \vee \langle\!\langle\beta\rangle\!\rangle_j^{\neg=}\big)$ if for every data value d' and nodes y, y' that complete the paths, it cannot be that $d' = \mathbf{d}(y) = \mathbf{d}(y')$. This is true, since $\mathbf{t}|_x \models \neg\langle\alpha = \beta\rangle$. On the other hand, note that by the same reason, for every node z' such that $(z, z') \in [\![\beta]\!]^{\mathbf{t}}$, there are accepting dvp-runs for $(\langle\!\langle\alpha\rangle\!\rangle_i^{\neg=}, \mathbf{d}(z'))$ and for $(\mathsf{store}(\langle\!\langle\alpha\rangle\!\rangle_i^{\neg=}), d')$ from z' (and respectively swapping α and β). Thus, by combining all the accepting dvp-runs from $(\mathsf{spread}\big((\langle\!\langle\alpha\rangle\!\rangle_i^{\neg=} \vee \langle\!\langle\beta\rangle\!\rangle_j^{\neg=}\big), d)$ for every such z, we obtain an accepting dvp-run from $(\langle\!\langle\varphi\rangle\!\rangle, d)$ on $\mathbf{t}|_x$.

Finally, if $\varphi = \neg\langle\alpha \neq \beta\rangle$, then $\delta(\langle\!\langle\varphi\rangle\!\rangle) = \langle\!\langle\neg\langle\alpha\rangle\rangle\!\rangle \vee \langle\!\langle\neg\langle\beta\rangle\rangle\!\rangle \vee \mathsf{guess}(\langle\!\langle\alpha, \beta\rangle\!\rangle^{\neg\neq})$. Since $x \in [\![\varphi]\!]^{\mathbf{t}}$, we have that either (i), (ii), or (iii) (on page 36) holds for $\mathbf{t}|_x$. The first two conditions correspond to the properties for which $\neg\langle\alpha\rangle$ and $\neg\langle\beta\rangle$ are satisfied, which are equivalent to $\neg\langle\alpha = \alpha\rangle$ and $\neg\langle\beta = \beta\rangle$. By the case already shown, this means that $\langle\!\langle\neg\langle\alpha = \alpha\rangle\rangle\!\rangle$ or $\langle\!\langle\neg\langle\beta = \beta\rangle\rangle\!\rangle$ have accepting dvp-runs on $\mathbf{t}|_x$ respectively. Since the definition of $\langle\!\langle\neg\langle\alpha = \alpha\rangle\rangle\!\rangle$ and $\langle\!\langle\neg\langle\alpha\rangle\rangle\!\rangle$ (idem with β) are treated identically by the automaton, $(\langle\!\langle\varphi\rangle\!\rangle, d)$ has an accepting dvp-run on $\mathbf{t}|_x$.

We now show that the third condition corresponds to having an accepting dvp-run from $(\mathsf{guess}(\langle\!\langle\alpha, \beta\rangle\!\rangle^{\neg\neq}), d)$ on $\mathbf{t}|_x$ for any data value d. By definition of δ, there is an accepting dvp-run if there is a data value d' such that there are accepting dvp-runs from $(\langle\!\langle\alpha\rangle\!\rangle_0^{\neg\neq}, d')$ and $(\langle\!\langle\beta\rangle\!\rangle_0^{\neg\neq}, d')$. By applying the inductive hypothesis on all positions below x and subformulas of $\mathsf{nsub}(\eta)$, we have the following, for $\mathcal{D}_x^\alpha = \{\mathbf{d}(y) \mid (x, y) \in [\![\alpha]\!]^{\mathbf{t}}\}$, $\mathcal{D}_x^\beta = \{\mathbf{d}(y) \mid (x, y) \in [\![\beta]\!]^{\mathbf{t}}\}$.

Claim 5. *For any $e \notin \mathcal{D}_x^\alpha$, there is an accepting dvp-run from $(\langle\!\langle\alpha\rangle\!\rangle^{\neg=}, e)$ on $\mathbf{t}|_x$. If $\mathcal{D}_x^\alpha = \{e\}$, there is an accepting dvp-run from $(\langle\!\langle\alpha\rangle\!\rangle^{\neg\neq}, e)$ on $\mathbf{t}|_x$. Idem for β and $\mathcal{D}_x^\beta$.*

Note that since $\mathbf{t}|_x \models \neg\langle\alpha \neq \beta\rangle$, condition (i) corresponds to $\mathcal{D}_x^\alpha = \emptyset$, condition (ii) to $\mathcal{D}_x^\beta = \emptyset$, and condition (iii) to $\mathcal{D}_x^\alpha = \mathcal{D}_x^\beta = \{e\}$. Therefore, if condition (iii) holds, there are accepting dvp-runs from $(\langle\!\langle\alpha\rangle\!\rangle^{\neg\neq}, e)$ and $(\langle\!\langle\beta\rangle\!\rangle^{\neg\neq}, e)$, and hence also from $(\mathsf{guess}(\langle\!\langle\alpha, \beta\rangle\!\rangle^{\neg\neq}), d)$ for any d, by guessing the data value e. Thus, there is an accepting dvp-run from $(\langle\!\langle\varphi\rangle\!\rangle, d)$ if conditions (i), (ii) or (iii) hold. Therefore, there is an accepting dvp-run from $(\langle\!\langle\varphi\rangle\!\rangle, d)$ on $\mathbf{t}|_x$.

($\Leftarrow$) Consider an accepting dvp-run $\mathcal{S}_1 \twoheadrightarrow \cdots \twoheadrightarrow \mathcal{S}_n$ from $(\langle\!\langle\varphi\rangle\!\rangle, d)$ on $\mathbf{t}|_x$. We show that $\mathbf{t}|_x \models \varphi$.

If $\varphi = \langle\alpha = \beta\rangle$, by definition of δ the automaton guesses a data value d and creates a thread $(\langle\!\langle\alpha, \beta\rangle\!\rangle^=, d)$, which means that there is an accepting dvp-run from $(\langle\!\langle\alpha, \beta\rangle\!\rangle^=, d)$. Then, there are accepting dvp-runs from $(\langle\!\langle\alpha\rangle\!\rangle^=, d)$ and $(\langle\!\langle\beta\rangle\!\rangle^=, d)$. By Claim 4.a, there must be two nodes y, z such that $(x, y) \in [\![\alpha]\!]$ and $(x, z) \in [\![\beta]\!]$ with $d = \mathbf{d}(z) = \mathbf{d}(y)$. Hence, $\mathbf{t}|_x \models \varphi$. The case $\varphi = \langle\alpha \neq \beta\rangle$ is treated in a similar way.

If $\varphi = \neg\langle\alpha = \beta\rangle$, suppose ad absurdum that there is a data value d and two positions y, z such that $(x, y) \in [\![\alpha]\!]^{\mathbf{t}}$ and $(x, z) \in [\![\beta]\!]^{\mathbf{t}}$, and $\mathbf{d}(y) = \mathbf{d}(z) = d$. Let z' be the closest common ancestor of y, z in the fcns coding. Since there is an accepting dvp-run from $\langle\!\langle\alpha, \beta\rangle\!\rangle^{\neg=}$, there must be some states i', j' assumed by $\mathcal{H}_\alpha$ and $\mathcal{H}_\beta$ after reading $str(x, z')$, such that the following holds.

Claim 6. *There is a moving configuration $\mathcal{S}_r$ containing $(\!(\alpha, \beta)\!)_{i',j'}^{\neg =}$ with position z' such that*

- *if $z' = y$, then $((\!(\alpha)\!)_{i'}^{\neg =}, \mathbf{d}(y))$ is in some $\mathcal{S}_s$, $s > r$, with position z',*
- *if $z' = z$, then $((\!(\beta)\!)_{j'}^{\neg =}, \mathbf{d}(z))$ is in some $\mathcal{S}_s$, $s > r$, with position z',*
- *for every data value d of $\mathcal{S}_r$, $((\!(\alpha)\!)^{\neg =}, d)_{i'}$ or $((\!(\beta)\!)^{\neg =}, d)_{j'}$ are in some $\mathcal{S}_s$, $s > r$, with position z'.*

Hence, by Claim 6 there must be a configuration $\mathcal{S}_r$ with position z', where z' is the common ancestor of y and z in the fcns coding, such that for all data values d' of $\mathcal{S}_r$ it is not true that $\mathbf{d}(x) = \mathbf{d}(y) = d'$. But by the fact that this is a dvp-run, this means that there cannot be *any* data value d'' such that $\mathbf{d}(y) = \mathbf{d}(z) = d''$. In particular, d, and thus we have a contradiction. Then, $\mathbf{t}|_x \models \varphi$. The cases of $y \preceq_{fcns} z$ or $z \preceq_{fcns} y$ are only easier.

Finally, if $\varphi = \neg\langle \alpha \neq \beta \rangle$, then $\mathcal{A}$ must have an accepting dvp-run on $\mathbf{t}|_x$ from $((\!(\neg\langle \alpha \rangle)\!), d)$, $((\!(\neg\langle \beta \rangle)\!), d)$, or $((\!(\alpha, \beta)\!)_{0,0}^{\neq}, d)$ for some $d \in \mathbb{D}$. In the first two cases this means that $[\![\langle \alpha \rangle]\!]^{\mathbf{t}|_x} = \emptyset$ or $[\![\langle \beta \rangle]\!]^{\mathbf{t}|_x} = \emptyset$ (by reduction to the case $\neg\langle \alpha = \alpha \rangle$ already treated) and hence that $\mathbf{t} \models \varphi$. In the latter case, by definition of δ, all positions y whose paths from x satisfy α or β are such that $\mathbf{d}(y) = d$. This implies that $\mathbf{t} \not\models \langle \alpha \neq \beta \rangle$, and hence $\mathbf{t} \models \varphi$. $\square$

Lemma 6.10 together with Proposition 6.8 concludes the proof of Proposition 6.6. We then have that Theorem 6.4 holds.

Proof of Theorem 6.4. By Proposition 6.6, satisfiability of $\mathsf{regXPath}(\mathfrak{F}, =)$ is reducible to the nonemptiness problem for $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$. On the other hand, we remark that $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ automata can encode any regular tree language —in particular a DTD, the core of XML Schema, or Relax NG— and are closed under intersection by Proposition 5.3. Also, the logic can express any unary key constraint as stated in Lemma 6.2. Hence, by Theorem 5.5 the decidability follows. $\square$

EXTENSIONS

We consider some operators that can be added to $\mathsf{regXPath}(\mathfrak{F}, =)$ preserving the decidability of the satisfiability problem. For each of these operators, we will see that they can be coded as a $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$ automaton, following the same lines of the translation in Section 6.4.

6.5. Allowing upward axes.

Here we explore one possible decidable extension to the logic $\mathsf{regXPath}(\mathfrak{F}, =)$, whose decidability can be reduced to that of $\mathsf{ATRA}(\mathsf{guess}, \mathsf{spread})$. In this extension we can make use of *upward* ($\uparrow$) and *leftward* ($\leftarrow$) axes to navigate the tree in a restricted way. We can test, for example, that all the ancestors of a given node labeled with a have the same data value as all the descendants labeled with b, with the formula $\neg\langle \uparrow^*[a] \neq \downarrow^*[b] \rangle$, but we cannot test its negation.

Let $\mathsf{regXPath}^{\mathfrak{B}}(\mathfrak{F}, =)$ be the fragment of $\mathsf{regXPath}(\mathfrak{F} \cup \mathfrak{B}, =)$ where $\mathfrak{B} := \{\uparrow, \uparrow^*, \leftarrow, {}^*\!\leftarrow\}$ defined by the grammar

$$\varphi, \psi ::= \neg\mathsf{a} \mid \mathsf{a} \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle \alpha_{\mathfrak{f}} \rangle \mid \langle \alpha_{\mathfrak{b}} \rangle \mid$$
$$\langle \alpha_{\mathfrak{f}} \circledast \beta_{\mathfrak{f}} \rangle \mid \neg\langle \alpha_{\mathfrak{f}} \circledast \beta_{\mathfrak{f}} \rangle \mid \neg\langle \alpha_{\mathfrak{b}} = \beta_{\mathfrak{f}} \rangle \mid \neg\langle \alpha_{\mathfrak{b}} \neq \beta_{\mathfrak{f}} \rangle$$

with $\circledast \in \{=, \neq\}, \mathsf{a} \in \mathbb{A}$, and

$$\alpha_{\mathfrak{f}}, \beta_{\mathfrak{f}} ::= [\varphi] \mid \alpha_{\mathfrak{f}}\beta_{\mathfrak{f}} \mid \alpha_{\mathfrak{f}} \cup \beta_{\mathfrak{f}} \mid o\,\alpha_{\mathfrak{f}} \mid (\alpha_{\mathfrak{f}})^* \qquad o \in \{\downarrow, \rightarrow, \varepsilon\},$$
$$\alpha_{\mathfrak{b}}, \beta_{\mathfrak{b}} ::= [\varphi] \mid \alpha_{\mathfrak{b}}\beta_{\mathfrak{b}} \mid \alpha_{\mathfrak{b}} \cup \beta_{\mathfrak{b}} \mid o\,\alpha_{\mathfrak{b}} \mid (\alpha_{\mathfrak{b}})^* \qquad o \in \{\uparrow, \leftarrow, \varepsilon\}.$$

We must note that $\mathsf{regXPath}^{\mathfrak{B}}(\mathfrak{F}, =)$ contains $\mathsf{regXPath}(\mathfrak{F}, \mathfrak{B})$, that is, the full data-unaware fragment of XPath. We also remark that it is *not* closed under negation. Indeed, we cannot express the negation of "there exists an a such that all its ancestors labeled b have different data value" which is expressed by $\downarrow^*[a \wedge \neg\langle\uparrow^*[b] = \varepsilon\rangle]$. As shown in Proposition 3.2, if the negation of this property were expressible, then its satisfiability would be undecidable. It is not hard to see that we can decide the satisfiability problem for this fragment.

Consider the data test expressions of the types

$$\neg\langle\alpha_{\mathfrak{b}} = \beta_{\mathfrak{f}}\rangle \qquad \text{and} \qquad \neg\langle\alpha_{\mathfrak{b}} \neq \beta_{\mathfrak{f}}\rangle$$

where $\beta_{\mathfrak{f}} \in \mathsf{regXPath}(\mathfrak{F}, =)$ and $\alpha_{\mathfrak{b}} \in \mathsf{regXPath}(\mathfrak{B})$. We can decide the satisfaction of these kinds of expressions by means of $\mathsf{spread}(\ ,\ )$, using carefully its first parameter to select the desired threads from which to collect the data values we are interested in. Intuitively, along the dvp-run we throw threads that save the current data value and try out all possible ways to verify $\alpha_{\mathfrak{b}}^r \in \mathsf{regXPath}(\mathfrak{F}, =)$, where $(\ )^r$ stands for the *reverse* of the regular expression. Let the automaton arrive with a thread $(\langle\!|\alpha_{\mathfrak{b}}\rangle\!|, d)$ whenever $\alpha_{\mathfrak{b}}^r$ is verified. This signals that there is a backwards path from the current node in the relation $\alpha_{\mathfrak{b}}$ that arrives at a node with data value d. Hence, at any given position, the instruction $\mathsf{spread}(\langle\!|\alpha_{\mathfrak{b}}\rangle\!|, \langle\!|\alpha_{\mathfrak{f}}\rangle\!|^{\neg\circledast})$ translates correctly the expression $\neg\langle\alpha_{\mathfrak{b}} \circledast \beta_{\mathfrak{f}}\rangle$. Furthermore, $\alpha_{\mathfrak{b}}$ need not be necessarily in $\mathsf{regXPath}(\mathfrak{B})$, as its intermediate node tests can be formulæ from $\mathsf{regXPath}(\mathfrak{F}, =)$. We then obtain the following result.

Remark 6.11. Satisfiability for $\mathsf{regXPath}^{\mathfrak{B}}(\mathfrak{F}, =)$ under key constraints and DTDs is decidable.

6.6. **Allowing stronger data tests.** Consider the property "*there are three descendant nodes labeled a, b and c with the same data value*". That is, there exists some data value d such that there are three nodes accessible by $\downarrow^*[a]$, $\downarrow^*[b]$ and $\downarrow^*[c]$ respectively, all carrying the datum d. Let us denote the fact that they have the same or different datum by introducing the symbols '$\sim$' and '$\not\sim$', and appending it at the end of the path. Then in this case we write that the elements must satisfy $\downarrow^*[\mathsf{a}]\sim$, $\downarrow^*[\mathsf{b}]\sim$, and $\downarrow^*[\mathsf{c}]\sim$. We then introduce the node expression $\{\!\{\alpha_1 s_1, \ldots, \alpha_n s_n\}\!\}$ where α_i is a path expression and $s_i \in \{\sim, \not\sim\}$ for all $i \in [1..n]$. Semantically, it is a node expression that denotes all the tree positions x from which we can access n positions $x_1, \ldots, x_n$ such that there exists $d \in \mathbb{D}$ where for all $i \in [n]$ the following holds: $(x, x_i) \in [\![\alpha_i]\!]$; if $s_i = \sim$ then $\mathbf{d}(x_i) = d$; and if $s_i = \not\sim$ then $\mathbf{d}(x_i) \neq d$. Note that now we can express $\langle\alpha = \beta\rangle$ as $\{\!\{\alpha\sim, \beta\sim\}\!\}$ and $\langle\alpha \neq \beta\rangle$ as $\{\!\{\alpha\sim, \beta\not\sim\}\!\}$. Let us call $\mathsf{regXPath}^+(\mathfrak{F}, =)$ to $\mathsf{regXPath}(\mathfrak{F}, =)$ extended with the construction just explained. This is a more expressive formalism since the first mentioned property —or, to give another example, $\{\!\{\downarrow^*[\mathsf{a}]\sim, \downarrow^*[\mathsf{b}]\sim, \downarrow^*[\mathsf{a}]\not\sim, \downarrow^*[\mathsf{b}]\not\sim\}\!\}$— is not expressible in $\mathsf{regXPath}(\mathfrak{F}, =)$.

We argue that satisfiability for this extension can be decided in the same way as for $\mathsf{regXPath}(\mathfrak{F}, =)$. It is straightforward to see that *positive* appearances can easily be translated with the help of the guess operator. On the other hand, for negative appearances, like $\neg\{\!\{\alpha_1 s_1, \ldots, \alpha_n s_n\}\!\}$, we proceed in the same way as we did for $\mathsf{regXPath}(\mathfrak{F}, =)$. The only difference being that in this case the automaton will simulate the simultaneous evaluation of

the n expressions and calculate all possible configurations of the closest common ancestors of the endpoints, performing a spread at each of these intermediate points.

Remark 6.12. Satisfiability of regXPath$^+(\mathfrak{F},=)$ under key constraints and DTDs is decidable.

7. Concluding remarks

We presented a simplified framework to work with one-way alternating register automata on data words and trees, enabling the possibility to easily show decidability of new operators by proving that they preserve the downward compatibility of a well-structured transition system. It would be interesting to hence investigate more decidable extensions, to study the expressiveness limits of decidable logics and automata for data trees.

Also, this work argues in favor of exploring computational models that although they might be not closed under all boolean operations, can serve to show decidability of logics closed under negation —such as forward-XPath— or expressive natural extensions of existing logics —such as LTL$^{\downarrow}_{\mathsf{nnf}}(\mathfrak{F}, \exists^{\downarrow}_{\geq}, \forall^{\downarrow}_{\leq})$.

We finally mention that even though XPath$(\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, =)$ (*i.e.*, forward-XPath) has a non-primitive-recursive complexity, the results of [Fig11] suggest that it seems plausible that XPath$(\downarrow, \downarrow^*, \rightarrow^*, =)$ or even XPath$(\downarrow, \downarrow^*, \rightarrow^*, {}^*\!\!\leftarrow, =)$ are decidable in elementary time [see Fig11, Conjecture 1].

Acknowledgements

References

[AČJT96] Parosh Aziz Abdulla, Kārlis Čerāns, Bengt Jonsson, and Yih-Kuen Tsay. General decidability theorems for infinite-state systems. In *Annual IEEE Symposium on Logic in Computer Science (LICS'96)*, pages 313–321, 1996.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[ADOW05] Parosh Aziz Abdulla, Johann Deneux, Joël Ouaknine, and James Worrell. Decidability and complexity results for timed automata via channel machines. In *International Colloquium on Automata, Languages and Programming (ICALP'05)*, pages 1089–1101, 2005.

[BFG08] Michael Benedikt, Wenfei Fan, and Floris Geerts. XPath satisfiability in the presence of DTDs. *Journal of the ACM*, 55(2):1–79, 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.

[CD99] James Clark and Steve DeRose. XML path language (XPath). Website, 1999. W3C Recommendation. http://www.w3.org/TR/xpath.

[CS08] Pierre Chambart and Philippe Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *Annual IEEE Symposium on Logic in Computer Science (LICS'08)*, pages 205–216. IEEE Computer Society Press, 2008.

[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.

[DL09] Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *ACM Transactions on Computational Logic*, 10(3), 2009.

[DLN05] Stéphane Demri, Ranko Lazić, and David Nowak. On the freeze quantifier in constraint LTL: Decidability and complexity. In *International Symposium on Temporal Representation and Reasoning (TIME'05)*, pages 113–121. IEEE Computer Society Press, 2005.

[FFSS11] Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson's lemma. In *Annual IEEE Symposium on Logic in Computer Science (LICS'11)*. IEEE Computer Society Press, 2011.

[FHL10] Diego Figueira, Piotr Hofman, and Sławomir Lasota. Relating timed and register automata. In *International Workshop on Expressiveness in Concurrency (EXPRESS'10)*, 2010.

[Fig09] Diego Figueira. Satisfiability of downward XPath with data equality tests. In *ACM Symposium on Principles of Database Systems (PODS'09)*, pages 197–206. ACM Press, 2009.

[Fig10a] Diego Figueira. Forward-XPath and extended register automata on data-trees. In *International Conference on Database Theory (ICDT'10)*. ACM Press, 2010.

[Fig10b] Diego Figueira. *Reasoning on Words and Trees with Data*. Ph.D. thesis, Laboratoire Spécification et Vérification, ENS Cachan, France, December 2010.

[Fig11] Diego Figueira. A decidable two-way logic on data words. In *Annual IEEE Symposium on Logic in Computer Science (LICS'11)*. IEEE Computer Society Press, 2011.

[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 *International Symposium on Mathematical Foundations of Computer Science (MFCS'09)*, volume 5734 of *LNCS*, pages 331–343. Springer, 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 (STACS'11)*. Springer, 2011.

[GF05] Floris Geerts and Wenfei Fan. Satisfiability of XPath queries with sibling axes. In *International Symposium on Database Programming Languages (DBPL'05)*, volume 3774 of *Lecture Notes in Computer Science*, pages 122–137. Springer, 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.

[Hig52] Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society (3)*, 2(7):326–336, 1952.

[JL07] Marcin Jurdziński and Ranko Lazić. Alternation-free modal mu-calculus for data trees. In *Annual IEEE Symposium on Logic in Computer Science (LICS'07)*, pages 131–140. IEEE Computer Society Press, 2007.

[JL11] Marcin Jurdziński and Ranko Lazić. Alternating automata on data trees and XPath satisfiability. *ACM Transactions on Computational Logic*, 12(3):19, 2011.

[Kru60] Joseph B. Kruskal. Well-quasi-ordering, the tree theorem, and Vazsonyi's conjecture. *Transactions of the American Mathematical Society*, 95(2):210–225, 1960.

[LW70] M.H. Löb and S.S. Wainer. Hierarchies of number theoretic functions, I. *Archiv für Mathematische Logik und Grundlagenforschung*, 13:39–51, 1970.

[Mar04] Maarten Marx. XPath with conditional axis relations. In *International Conference on Extending Database Technology (EDBT'04)*, volume 2992 of *Lecture Notes in Computer Science*, pages 477–494. Springer, 2004.

[Sch10] Philippe Schnoebelen. Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In *International Symposium on Mathematical Foundations of Computer Science (MFCS'10)*, volume 6281 of *Lecture Notes in Computer Science*, pages 616–628. Springer, 2010.

[tC06] Balder ten Cate. The expressivity of XPath with transitive closure. In *ACM Symposium on Principles of Database Systems (PODS'06)*, pages 328–337. ACM Press, 2006.

[tCS08] Balder ten Cate and Luc Segoufin. XPath, transitive closure logic, and nested tree walking automata. In *ACM Symposium on Principles of Database Systems (PODS'08)*, pages 251–260. ACM Press, 2008.