An extension of data automata that captures XPath

Mikołaj Bojańczyk and Sławomir Lasota
Warsaw University
Email: bojan,sl@mimuw.edu.pl

***Abstract*—We define a new kind of automata recognizing properties of data words or data trees and prove that the automata capture all queries definable in Regular XPath. We show that the automata-theoretic approach may be applied to answer decidability and expressibility questions for XPath. Finally, we use the newly introduced automata as a common framework to classify existing automata on data words and trees, including data automata, register automata and alternating register automata.**

***Keywords*-Regular XPath, data automata, register automata.**

I. Introduction

In this paper, we study data trees. In a data tree, each node carries a label from a finite alphabet and a data value from an infinite domain. We study properties of data trees, such as those defined in XPath, which refer to data values only by testing if two nodes carry the same data value. Therefore we define a data tree as a pair $(t, \sim)$ where t is a tree over a finite alphabet and $\sim$ is an equivalence relation on nodes of t. Data values are identified with equivalence classes of $\sim$.

Recent years have seen a lot of interest in automata for data trees and the special case of data words. The general theme is that it is difficult to design an automaton which recognizes interesting properties and has decidable emptiness.

Decidable emptiness is important in XML static analysis. A typical question of static analysis is the implication problem: given two properties φ_1, φ_2 of XML documents (modeled as data trees), decide if every document satisfying φ_1 must also satisfy φ_2. Solving the implication problem boils down to deciding emptiness of $\varphi_1 \wedge \neg\varphi_2$.

A common logic for expressing properties is XPath. For XPath, satisfiability is undecidable in general, even for data words, see [1]. Satisfiability is undecidable also for most other natural logics, including first-order logic with predicates for order (or even just successor) and data equality.

The approach chosen in prior work was to find automata on data words or trees that would have decidable emptiness and recognize interesting, but necessarily weak, logics or fragments of XPath. These include: fragments of XPath without recursion or negation [1], [2]; first-order logic with two variables [3], [4]; forward-only fragments related to alternating automata [5]–[8]. The original automaton model for data words was [9]. See [10] for a survey.

In this paper, we take a different approach. Any model that captures XPath will have undecidable emptiness. We are not discouraged by this, and try to capture XPath by something that feels like an "automaton". Three tangible goals are: 1. use the automaton to decidable emptiness for interesting restrictions of data trees; 2. use the automaton to prove easily that the automaton (and consequently XPath) *cannot* express a property; 3. unify other automata models that have been suggested for data trees and words.

What is our new model? To explain it, we use logic. From a logical point of view, a nondeterministic automaton is a formula of the form $\exists X_1 \dots \exists X_n\ \varphi(X_1, \dots, X_n)$. As often in automata theory, when designing the automaton model, we try to use the prefix of existential set quantifiers as much as possible, in the interest of simplifying the kernel φ. For satisfiability, this is like a free lunch, since deciding satisfiability with or without the prefix are the same problem.

In the automaton model that we propose in this paper, the kernel φ is of the form "for every class X of $\sim$, property $\psi(X, X_1, \dots, X_n)$ holds", where ψ is an MSO formula that can use predicates for navigation (sibling order, descendant), predicates for testing labels from the finite alphabet, but not the predicate $\sim$ for data equality. The data $\sim$ is only used in saying that X is a class. In the case of data words, this model is an extension of the *data automata* introduced in [3], which correspond to the special case when ψ quantifies only over positions from X. For instance, our new model, but not data automata, can express the property "between every two different positions in the same class there is at most one position outside the class with label a".

The principal technical contribution of this paper is that the model above can recognize all boolean queries of XPath. This proof is difficult, and takes over ten pages. We believe the real value of this paper lies in this proof, which demonstrates some powerful normalization techniques for formulas describing properties of data trees. Since the scope of applicability for these techniques will be clear only in the future; and since the appreciation of an "automaton model" may ultimately be a question of taste, we describe in more details the three tangible goals mentioned above.

1. The ultimate goal of this research is to find interesting classes of data trees which yield decidable emptiness for

This work has been partially supported by the Polish government grant no. N206 008 32/0810 and by the FET-Open grant agreement FOX, number FP7-ICT-233599.

XPath. As a proof of concept, we define a simple subclass of data trees, called bipartite data trees, and prove that emptiness of our automata (and consequently of XPath) is decidable for bipartite data trees. This is only a preliminary result, we intend to find new subclasses in the future.

2. We use the automaton to prove that XPath cannot define certain properties. Proving inexpressibilty results for XPath is difficult, because the truth value of an XPath query in a position x might depend on the truth value of a subquery in a position $y < x$, which in turn might depend on the truth value of a subquery in a position $z > y$, and so on. On the other hand, our automaton works in one direction, so it is easier to understand its limitations. We use (an extension of) our automata to prove that for documents with two equivalence relations $\sim_1$ and $\sim_2$, some properties of two-variable first-order logic cannot be captured by XPath, which was an open question.

3. We use the automaton to classify existing models for data words in a single framework. A problem with the research on data words and data trees is that the models are often incomparable in expressive power. We show that all the existing models can be seen as syntactic fragments of our automaton. We hope that this classification underlines more clearly what the differences are between the models.

II. Preliminaries

Trees. Trees are unranked, finite, and labeled by a finite alphabet Σ. We use the terms child, parent, sibling, descendant, ancestor, node in the usual way. The siblings are ordered. We write $x \leq y$ when x is an ancestor of y. Every nonempty set of nodes $x_1, \ldots, x_n$ in a tree has a greatest closest common ancestor (the greatest lower bound wrt $\leq$), which is denoted $\mathrm{gca}(x_1, \ldots, x_n)$.

Let t and s be two trees, over alphabets Σ and Γ, respectively, that have the same sets of nodes. We write $t \otimes s$ for the tree over the product alphabet $\Sigma \times \Gamma$ that has the same nodes as s and t, and where every node has the label from t on the first coordinate, and the label from s on the second coordinate. If X is a set of nodes in a tree t, we write $t \otimes X$ for the tree $t \otimes s$, where s is the tree over alphabet $\{0, 1\}$, whose nodes are the nodes of t and whose labeling is the characteristic function of X.

Regular tree languages and transducers. We use the standard notion of regular tree languages for unranked trees. We also use transductions, which map trees to trees. Let Σ be an input alphabet and Γ an output alphabet. A regular tree language f over the product alphabet $\Sigma \times \Gamma$ can be interpreted as a binary relation, which contains pairs (s, t) such that $s \otimes t \in f$. We use the name *letter-to-letter transducer* for such a relation, underlining that the trees in a pair $(s, t) \in f$ must have the same nodes. In short, we simply say transducer. We often treat a transducer as a function that maps an input tree to a set of output trees, writing $t \in f(s)$ instead of $(s, t) \in f$.

Data trees. A *data tree* is a tree t equipped with an equivalence relation $\sim$ on its nodes that represents data equality. We use the name *class* for equivalence classes of $\sim$.

Queries. Fix an input alphabet. We use the name n-ary query for a function ϕ that maps a tree t over the input alphabet to a set $\phi(t)$ of n-tuples of its nodes. In this paper, we will deal with queries of arities 0,1,2 and 3, which are called boolean, unary, binary and ternary. We also study queries that input a data tree $(t, \sim)$; they output a set of node tuples $\phi(t, \sim)$ as well.

MSO. Logic is a convenient way of specifying queries, both for trees and data trees. We use monadic second-order logic (MSO). In a given tree, or a data tree, a formula of MSO is allowed to quantify over nodes of the tree using individual variables x, y, z, and also over sets of nodes using set variables X, Y, Z. A formula ϕ with free individual variables $x_1, \ldots, x_n$ defines an n-ary query, which selects in a tree t the set $\phi(t)$ of tuples $(x_1, \ldots, x_n)$ that make the formula true. To avoid confusion, we use round parentheses for the tree input of a query, $\phi(t)$, and square parentheses for indicating the free variables of a query. The two parenthesis can appear together, e.g. $\phi[x_1, \ldots, x_n](t)$ will be the set of n-tuples selected in a tree t by a query with free variables $x_1, \ldots, x_n$.

When working over trees without data, MSO formulas use binary predicates for the child, descendant and next-sibling relations, as well as a unary predicate for each label. Queries defined by MSO with these predicates are called *regular queries* (of course, regular queries can also be characterized in terms of automata). When working over data trees, we additionally allow a binary predicate $\sim$ to test data equality. A query using $\sim$ is no longer called regular. For instance, the following formula says that each class contains nodes with the same label.

$$\forall x \forall y \quad x \sim y \Rightarrow \bigvee_{a \in \Sigma} a(x) \wedge a(y)$$

Extended Regular XPath. We define a variant of XPath that works over data trees. For unary queries, the variant is an extension of XPath, thanks to including MSO as part of its syntax. Expressions of extended Regular XPath, which we call XPath in short, are defined below. Each expression defines a unary query.

- Every letter of the alphabet a is a unary query, which selects nodes with label a.
- Let $\Gamma = \{\phi_1, \ldots, \phi_n\}$ be a set of already defined unary queries of Extended Regular XPath, which will be treated as unary predicates. Suppose that $\varphi[x, y_1, y_2]$ be a regular (i.e. not using $\sim$) ternary query of MSO that uses the predicates from Γ. (Data equality $\sim$ might have been used to define the queries from Γ, but inside φ, the queries from Γ are treated as unary predicates.)

Then the following propery of x is a unary query.

$$\exists y_1 \exists y_2 \quad y_1 \sim y_2 \wedge \varphi[x, y_1, y_2]. \tag{1}$$

Likewise for $y_1 \not\sim y_2$ instead of $y_1 \sim y_2$.

Boolean queries can be defined by taking a unary query, and choosing the data trees where the root is selected.

Binary trees. A binary tree is a tree where each node has at most two children. Although the interest of XPath is mainly for unranked trees, we assume in the proofs that trees are binary. This assumption can be made because XPath, as well as the models of automata introduced later on, are stable under the usual first-child / next-sibling encoding in the following sense. A language L of unranked data trees can be expressed by a boolean XPath query if and only if the set of binary encodings of trees from L can be expressed by a boolean XPath query. A similar, though more technical, statement holds for unary queries.

III. Class automata

In this section we define a new type of automaton for data trees, called a class automaton, and state the main result: class automata capture all queries definable in XPath.

A *class automaton* is a type of automaton that recognizes properties of data trees. A class automaton is given by: an input alphabet Σ, a work alphabet Γ, a nondeterministic letter-to-letter tree transducer f from the input alphabet Σ to the work alphabet Γ, and a regular tree language on alphabet $\Gamma \times \{0,1\}$, called the class condition. The class automaton accepts a data tree $(t, \sim)$ over input alphabet Σ if there is some output $s \in f(t)$ such that for every class X, the class condition contains the tree $s \otimes X$.

Example 1. *Consider an input alphabet $\Sigma = \{a, b\}$. Let L be the data trees where some class contains at least three nodes with label a. This language is recognized by a class automaton. The work alphabet is $\Gamma = \{a, c\}$. The transducer guesses three nodes with label a, and outputs a on them, other nodes get c. The class condition consists of trees $s \otimes X$ over alphabet $\Gamma \times \{0,1\}$ where X contains all or none of the nodes with label a. Note that the class condition does not inspect positions outside X.*

Example 2. *Let K be the set of data words over $\Sigma = \{a, b\}$ where each class has exactly two positions $x < y$, and there is at most one a in the positions $\{x+1, \ldots, y-1\}$. In the class automaton recognizing K, the transducer is the identity function, and the class condition is*

$$\Sigma_0^* \cdot \Sigma_1 \cdot b_0^* \cdot (a_0 + \epsilon) \cdot b_0^* \cdot \Sigma_1 \cdot \Sigma_0^*$$

where Σ_i is a shortcut for $\Sigma \times \{i\}$, likewise for a_i and b_i.

Comparison to data automata. Class automata are closely related to *data automata* introduced in [3]. Data automata were defined for data words. Since it is not clear what the correct tree version thereof is, we just present the version for data words. Like a class automaton, a data automaton has an input alphabet Σ, a work alphabet Γ, and a nondeterministic letter-to-letter transducer f (this time only for words). The difference is in the class condition, which is less powerful in a data automaton. In a data automaton, the class condition is a word language over Γ, and not $\Gamma \times \{0,1\}$. The data automaton accepts a data words $(w, \sim)$ if there is some output $v \in f(w)$ such that for every class X, the class condition contains the substring of v obtained by only keeping positions from X. In the realm of data words, data automata can be seen as a special of class automata, where the class condition is only allowed to look at positions from the current class. The language L in Example 1 can be recognized by a data automaton (in the case of words), while the language K in Example 2 is a language that can be recognized by class automata, but not data automata. We will comment more closely on the relationship between data automata and class automata in Section VI, and also on the relationship to other types of automata for data words and data trees, including pebble automata [11], register automata [9] and alternating register automata.

The difference between data automata and class automata is crucial for decidability of emptiness. Data automata have decidable emptiness [3], the proof being a reduction to reachability in Vector Addition Systems with States.

Closure properties. Suppose that $f : \Sigma_1 \to \Sigma_2$ is any function. We extend f to a function $\hat{f}$ from data trees over alphabet Σ_1 to data trees over alphabet Σ_2, by just changing the labels of nodes, and not the tree structure or data values. We use the name relabeling for any such function $\hat{f}$.

Lemma 1. *Languages of data trees recognized by class automata are closed under union, intersection, images under relabelings, and inverse images under relabelings.*

In the proof, one uses Cartesian product for intersection, nondeterminism for union and images. The inverse images are the simplest: the letter-to-letter tree transducer in the data automaton is composed with the relabeling.

Evaluation. The evaluation problem (given an automaton and a data word/tree, check if the latter is accepted by the former) is NP-complete, even for a fixed data automaton (cf. [12]). Hence it is also NP-complete for class automata, which extend data automata.

Class automata as a fragment of MSO. One can see a class automaton as a restricted type of formula of monadic second-order logic. This is a formula of the form:

$$\exists X_1 \cdots \exists X_n \ \forall X \ \text{class}(X) \Rightarrow \varphi(X_1, \ldots, X_n, X) \tag{2}$$

where $X_1, \ldots, X_n, X$ are variables for sets of nodes, the

class formula is defined

$$\text{class}(X) = \qquad \exists y \forall x \quad x \in X \iff y \sim x$$

and φ is a formula of MSO that does not use $\sim$. Formulas of the above form recognize exactly the same languages of data trees as class automata. For translating a class automaton to a formula, one uses the variables $X_1, \ldots, X_n$ to encode the output of the transducer, and the formula φ to test two things: a) the variables $X_1, \ldots, X_n$ encode a legitimate output of the transducer; and b) the class condition holds for X.

Main result. The main result of this paper is that unary XPath queries over data trees can be recognized by class automata. To state the theorem, we need to say how a class automaton recognizes a unary query. We do this by encoding a unary query ϕ over data trees as a language of data trees:

$$L_\phi = \{(t \otimes X, \sim) : (t, \sim) \text{ is a data tree, } X = \phi(t, \sim)\}.$$

In other words, the language consists of data trees decorated with the set of nodes selected by the query. (This encoding does not generalize to binary queries.)

Theorem 1. *Every unary XPath query over data trees can be recognized by a class automaton.*

We begin the proof of Theorem 1, mainly to show where the difficulties appear. Then, we lay out the proof strategy in more detail. When referring to the language of a unary query, we mean the encoding above.

We do an induction on the size of the unary query. The base case, when the query is a label a, is straightforward. Consider now the induction step, with a unary query

$$\phi[x] = \quad \exists y_1 \exists y_2 \quad y_1 \sim y_2 \wedge \varphi[x, y_1, y_2]$$

as in (1). (The same argument works for the case where $y_1 \not\sim y_2$.) Let $\phi_1, \ldots, \phi_n$ be all the unary XPath subqueries that appear in φ. By the induction assumption, the languages of the subqueries are recognized by class automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$. Let the variables $X, X_1, \ldots, X_n$ denote sets of nodes. Consider the set L of data trees

$$(t \otimes X \otimes X_1 \otimes \cdots \otimes X_n, \sim)$$

such that a) for each $i \in \{1, \ldots, n\}$, the data tree $(t \otimes X_i, \sim)$ is accepted by the automaton $\mathcal{A}_i$; and b) X is the set of nodes selected by the query ϕ' obtained from ϕ by replacing each subquery ϕ_i with "has 1 on coordinate corresponding to X_i". Suppose that the language of ϕ' is recognized by a class automaton. Then so is L, by closure of class automata under intersection and inverse images of projections, see Lemma 1. Finally, the language of ϕ is the image of L under the projection which removes the labels describing the sets $X_1, \ldots, X_n$.

It remains to show that ϕ' is recognized by a class automaton (the advantage of ϕ' over ϕ is that the ternary query is now regular). Most of this paper is devoted to this case, which is stated in the following proposition.

Proposition 1. *Class automata can recognize queries*

$$\phi[x] = \quad \exists y_1 \exists y_2 \quad y_1 \sim y_2 \wedge \varphi[x, y_1, y_2],$$

where φ is a regular ternary query. Likewise for $y_1 \not\sim y_2$.

Proof strategy. The construction of the automaton for $\phi[x]$ is spread across several sections. In Section III-B, we introduce the main concepts underlying the proof. In particular, we define a new complexity measure for binary relations on tree domains, called guidance width, that seems to be of independent interest. In Section III-C we start the proof itself, formulate an induction, and reduce Proposition 1 to a more technical Theorem 2. In Section III-D we identify a simplified form of queries appearing in Theorem 2 (how arbitrary queries can be transformed to this simplified form we show in Appendix VII). Finally, Sections IV contains the proof of Theorem 2 for the simplified queries, the heart of the whole proof. We do the proof for the case of words only in Section IV – it already contains some of the important ideas for the general tree case, but is easier to digest. The proof for the general tree case is in Appendix VIII.

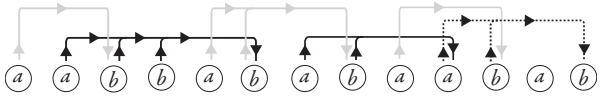
A. Discussion of the proof

In this section, we discuss informally the concepts that appear in the proof of Theorem 1.

We begin our discussion with words without data. For a regular binary query $\varphi[x, y]$, consider the unary query

$$\psi[x] = \exists y \; \varphi[x, y].$$

We use the name witness function in a word w for a function which maps every position x satisfying ψ to some y such that $\varphi[x, y]$ holds. Consider, as an example, the case where $\varphi[x, y]$ says that there exists exactly one z that has label a and satisfies $x < z < y$. The following picture shows a witness function.



The way the picture is drawn is important. The witness function is recovered by following arrowed lines. The arrowed lines are colored black, dashed black, or grey, in such a way that no position is traversed by two arrowed lines of the same color. With the formula ψ in the example, any input word has a witness function that can be drawn with three colors of arrowed lines. This can be generalized to arbitrary MSO binary queries; the number of colors depends only on the query, and not the input word.

The above observation may be used to design a nondeterministic automaton recognizing a property like $\forall x \; \psi[x]$.

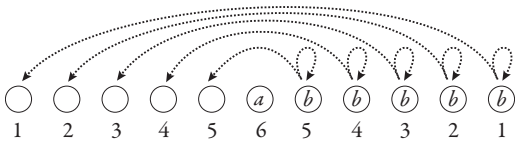
The automaton would guess the labeling by arrows and then verify its correctness. The number of states in the automaton would grow with the number of colors; hence the need for a bound on the number of colors. Of course, there are other ways of recognizing $\forall x \psi[x]$, but we talk about the coloring since this is the technique that will work with data.

We now move to data words. Consider a unary query

$$\psi[x] = \quad \exists y_1 \exists y_2 \quad y_1 \sim y_2 \ \wedge \ \varphi[x, y_1, y_2],$$

where $\varphi[x, y_1, y_2]$ says that $y_1 \leq x \leq y_2$, there is exactly one a label in the positions $\{y_1, \ldots, x\}$ and there is exactly one b label in the positions $\{x, \ldots, y_2\}$. The query $\psi[x]$ is an example of a query as in Proposition 1. Consider the following data word (the labels are blank, a and b).

Look at the first node with label b, which is selected by ψ, and consider pairs (y_1, y_2) required by $\psi[x]$, which we call *witnesses*. The only possibility for y_2 is x itself; thus y_1 is also determined, as the only other position with the same data value. The same situation holds for all other positions with label b, which are the only positions selected by ψ. The drawing below shows how witness pairs are assigned to positions.



We would like to draw this picture with colored arrows, as we did for the first example of witness functions. If we insist on drawing arrows that connect each position x with its corresponding witness y_1, then we will need 5 colors as the middle position (labeled by a) is traversed by 5 arrows; the picture also generalizes to any number of colors. On the other hand, connecting each position x with its corresponding y_2 (a self-loop) requires only one color. We can come up symmetric with instances of data words where connecting each node x to y_2 requires an unbounded number of colors.

A consequence of our main technical result, Theorem 2, is that a bounded number of colors is sufficient if we want to perform the following task: for each position x selected by ψ, choose some witness pair $y_1 \sim y_2$, and connect x to *either* y_1 *or* y_2.

The concepts of witness functions and coloring are defined more precisely below.

B. The core result

Witness functions. We will state some technical results for a structure more general than a data tree, namely a graph tree. A *graph tree* is a tree t together with an arbitrary symmetric binary relation E. A data tree is the special case of a graph tree where E is an equivalence relation.

Let $\varphi[x, y_1, y_2]$ be a regular query (think of Proposition 1), and consider a graph tree (t, E). We are interested in triples (x, y_1, y_2) selected by φ in t such that $(y_1, y_2) \in E$. (Think of E being either the data equivalence relation $\sim$, or its complement.) Consider any such triple. The node x is called the *source node*; the notion of source node is relative to the query φ and relation E, which will be usually clear from the context and not mentioned explicitly. The pair (y_1, y_2) is called the *witness pair*, y_1 is called the *first witness*, and y_2 is called the *second witness*. These notions are all relative to a given x, but if we do not mention the x, then x is quantified existentially. Let X be a set of source nodes in a graph tree (not necessarily containing all source nodes). A *witness function* for φ and X in a graph tree is a function which maps every source node $x \in X$ to some (first or second) witness. There may be many witness functions, since for each node we can choose to use either a first witness or a second one, and there may be multiple witness pairs.

The key technical result of this paper is that one can always find a witness function of low complexity. The notion of complexity is introduced below.

Guidance width. A *guide* in a tree t is given by two nonempty sets of *source nodes* and *target nodes*. The *support* of the guide is the set of all nodes and edges on (the shortest) paths that connect some source node with a target node, including all the source and target nodes. A guide *conflicts* with another guide if their supports intersect. We write π for guides.

A guidance system is a set of guides Π. It induces a relation containing all pairs (x, y) of tree nodes such that x is a source and y a target in some guide in Π. An n-color guidance system is a guidance system whose guides can be colored by n colors so that conflicting guides have different colors. The *guidance width* of a binary relation R on tree nodes is the smallest n such that some n-color guidance system induces R.

In the proof we will only consider guidance systems for relations R that are partial functions. In such cases, it is sufficient to restrict to *deterministic* guides, i.e., those with precisely one target node. From now on, if not stated otherwise, a guidance system will be implicitly assumed to contain only deterministic guides.

Witness functions of bounded width. We are now ready to state the main technical result, which forms the core of the proof of Theorem 1.

Theorem 2. *Let φ be a regular ternary query. There exists a constant m, depending only on φ, such that in every graph tree, every set of source nodes has some witness function of guidance width at most m.*

In other words, regular ternary queries have witness functions of *bounded guidance width*. Before proving the theorem, we show how it implies Theorem 1.

C. From Theorem 2 to Proposition 1

We show how Theorem 2 implies the last remaining piece of Theorem 1, namely Proposition 1. Consider a unary query $\phi[x]$ as in the statement of Proposition 1. We begin with the case when $\phi[x]$ requires $y_1 \sim y_2$. We need to find a class automaton that accepts the data trees $(t \otimes X, \sim)$ where X is the set of all nodes selected by ϕ in the data tree $(t, \sim)$. The class automaton will test the conjunction of two properties:

Completeness. Each node selected by ϕ in $(t, \sim)$ is in X.
Correctness. Each node in X is selected by ϕ in $(t, \sim)$.

We give separate class automata for the two properties. Completeness is simple. It can be rephrased as

> for every class Y and triple (x, y_1, y_2) selected by φ, if $y_1, y_2 \in Y$ then $x \in X$.

This is the type of property class automata are designed for: for every class, test a regular property. (Recall the discussion on class automata as a fragment of MSO.) Correctness is the difficult property, since the order of quantifiers is not the same as in an class automaton:

> for every $x \in X$ there is a class Y and $y_1, y_2 \in Y$ such that (x, y_1, y_2) is selected by φ.

Our solution is to use, as a part of the class automaton to be designed, a guidance system given by Theorem 2.

Apply Theorem 2 to φ, yielding a constant m. The class automaton for the correctness property works as follows. Given an input data tree $(t \otimes X, \sim)$, it guesses an m-color guidance system; let R stand for the induced relation. The automaton then checks the two conditions below.

A. For every $x \in X$ there is some y with xRy.
B. For every class Y, if xRy, $x \in X$, $y \in Y$, then either (x, y, y') or (x, y', y) is in $\varphi(t)$, for some $y' \in Y$.

If the class automaton accepts, then clearly every position in X is a source node. Conversely, if all nodes in X are source nodes, then there is an accepting run of the above class automaton. This accepting run uses the guidance system for the witness function from Theorem 2.

This completes the proof for the case when $\phi[x]$ requires $y_1 \sim y_2$. For the case $y_1 \not\sim y_2$, the proof is almost the same, but for two changes. The first change is that we apply Theorem 2 to the graph trees (t, E), obtained from data trees $(t, \sim)$ by taking as E the complement of $\sim$. This explains why Theorem 2 is formulated for graph trees and not just data trees. The second change is that we write $y' \notin Y$ instead of $y' \in Y$ at the end of condition B.

D. Simplifying assumptions

Before proving Theorem 2, we make two simplifying assumptions about the query $\varphi[x, y_1, y_2]$.

For two nodes x, y in a tree t, we write $\mathrm{word}_t(x, y)$ for the sequence of labels on the unique shortest path from x to y in t, including x and y. We omit the subscript t when a tree is clear from the context. Note that $\mathrm{word}_t(x, y)$ is always nonempty and $\mathrm{word}_t(x, x)$ is the label of x.

The two assumptions about the query $\varphi[x, y_1, y_2]$ are:

1) All selected triples satisfy $y_1 < x < y_2$.
2) Whether or not a triple is selected depends only on the words $\mathrm{word}_t(y_1, x)$ and $\mathrm{word}_t(x, y_2)$. It does not depend on nodes outside the path from y_1 to y_2.

In Appendix VII we show that the above conditions can be assumed without loss of generality. In the case of words, the simplification is standard, but for trees it requires new ideas about guidance systems.

IV. Theorem 2 for words

In this section we prove Theorem 2 for (graph) words, which are the special case of (graph) trees where each node has at most one child. For words we say *position* instead of *node*. The case of trees is solved in Appendix VIII.

Recall the simplifying assumptions from Section III-D. Since the query is regular, then the dependency stated in item 2 is a regular dependency. We use an algebraic approach to represent the regular query. There is a morphism $\alpha : \Sigma^* \to S$, which maps each word to an element of a finite semigroup S. Whether or not a triple (x, y_1, y_2) is selected by φ depends only on the images

$$\alpha(\mathrm{word}_w(y_1, x)) \in S \qquad \alpha(\mathrm{word}_w(x, y_2)) \in S. \tag{3}$$

Forward Ramseyan splits. In a graph word (w, E) we will distinguish two types of edges: *word edges*, which connect positions in the word w with their successors, and *class edges*, which are from the additional structure E. We also have two dummy word edges: one entering the first position, and one leaving the last position. We order word edges according to the positions in the word, with the two dummy word edges coming first and last, respectively. For two word edges $e \leq f$ in a word w, we write $\mathrm{word}_w(e, f)$ for the infix of w that begins in the target of e and ends in the source of f. In particular $\mathrm{word}_w(e, e) = \epsilon$.

A *split of height* n in a word w is a function σ that maps each word edge to a number in $\{1, \ldots, n\}$. We say that two word edges e, f are *neighbours* with respect to a split σ, if σ assigns the same number to e and f, and all word edges between e and f are mapped to at most $\sigma(e)$. A split σ is called *forward Ramseyan* with respect to a morphism α if

$$\alpha(\mathrm{word}_w(e, f)) = \alpha(\mathrm{word}_w(e, g)) \tag{4}$$

holds for every three pairwise neighbouring word edges $e < f < g$. The following theorem was shown in [13].

Theorem 3. *Any word $w \in \Sigma^*$ has a forward Ramseyan split σ_w of height $\mathcal{O}(|S|)$. Furthermore, the split is left-to-right deterministic in the following sense: if two words agree*

on a prefix leading to a word edge e, then the splits also agree on the prefix leading to e, including e.

The reason for the determinism property is that for any morphism α, there is a deterministic left-to-right transducer that produces the splits. We do not need the determinism property for words, but it will be important for trees. In the sequel we assume fixed forward Ramseyan splits σ_w for each word $w \in \Sigma^*$ as stated in Theorem 3.

Factors. Fix a word w. The split σ_w divides the word into pieces, called factors, which are defined as follows. The two word edges e and f are called *visible* if all word edges between them get values strictly smaller than both $\sigma_w(e)$ and $\sigma_w(f)$. A *factor* is a set of positions between visible word edges. These word edges are called the *border edges* of the factor. For convenience assume that the the two dummy edges beginning and ending w are visible. We write F, G, H for factors. Consider a factor F with border edges $e < f$. Let i be the maximal number assigned by the split to word edges inside the factor (word edges strictly between the border edges), and let $e_1, \ldots, e_k$ be all the word edges in the factor that are assigned this maximal number i. It is not difficult to see that the maximal (with respect to inclusion) factors strictly included in F are the factors whose pairs of border edges are, respectively, $(e, e_1), (e_1, e_2), \ldots, (e_{k-1}, e_k), (e_k, f)$. We call these factors the *subfactors* of F. The subfactors form a partition of the factor.

Our proof of Theorem 2 is based on a lemma stated below. The lemma is proved by induction on the height of factors (the height of a factor is the maximal value of the split on the factor) .

To state the Main Lemma, we need one new notion. A guidance system is called *consistent* if each of its guides obeys the following uniqueness requirement: if a subset Z of source nodes is guided to the same node y, then there is a pair (y_1, y_2) that is a witness pair for all nodes Z, with $y_1 = y$ or $y_2 = y$. Roughly speaking: if all nodes in Z agree on the witness they are guided to, then they agree on the other witness as well. The notion of consistency is meaningful only relative to a given ternary query.

Lemma 2 (Main Lemma)**.** *Fix a factor height h. There is a bound $n \in \mathbb{N}$, depending only on φ and h, such that for every graph word (w, E), every factor F in w of height h, and every set $X \subseteq F$ of source nodes, there is a witness function for φ and X in (w, E) induced by a consistent guidance system using at most n colors. Furthermore, this witness function only points to nodes inside or to the right of F.*

Note that we do not require the witnesses for nodes in X to be contained in F. Theorem 2 is a special case of the lemma for F that contains all positions in the word. By Theorem 3, this factor F has bounded height $O(|S|)$.

The proof of the lemma is by induction on the height h. The number of colors n will depend on h and the size of the monoid S recognizing the query φ. When going from height h to height $h+1$, there will be a linear blowup in the number of colors. Therefore, n will be exponential in the height of F. This contrasts with the tree case, where each increment in the height comes with a quadratic blowup in n, which makes n doubly exponential in the height.

Since the witness function will be induced by a guidance system, the last assumption in the Main Lemma could be restated as saying that no guide in the guidance system passes through the left border edge of F.

The induction base, when F is empty or contains just one position, is immediate. Consider now the induction step. Let $G_1, \ldots, G_k$ be all subfactors of F, listed from left to right. We use the term *internal border edge* for any word edge that connects G_i with G_{i+1}. We use the term *external border edge* for the two border edges of F, which are incident with the first position in G_1 and the last position in G_k, respectively.

Let n be the number of colors that is necessary to color guidance systems for sets contained in the subfactors of F. This number is obtained from the induction assumption.

For a node $x \in X$ and a witness (y_1, y_2) we define two numbers m_1, m_2. Let m_1 be the number of internal border edges between y_1 and x, and let m_2 be the number of internal border edges between x and y_2. For technical convenience, we deliberately choose not to count external border edges. We divide the set X into three parts:

1) Nodes $x \in X$ that have a witness with $m_2 \leq 1$.
2) Nodes $x \in X$ that have a witness with $m_1 \leq 1$ and $m_2 \geq 2$.
3) Nodes $x \in X$ that have a witness with $m_1, m_2 \geq 2$.

If some node belongs to two or three of these parts, by having several witnesses, we choose one arbitrarily. We prove the Main Lemma for each of the three parts separately. Next, we combine the three guidance systems into a single guidance system.

Nodes $x \in X$ that have a witness with $m_2 \leq 1$. Take a subfactor G_i, and remove from the graph E all class edges (y_1, y_2) where the path from the inside of G_i to y_2 crosses more than one internal border edge. None of these removed pairs are necessary as witnesses for $X \cap G_i$, so we can apply the induction assumption G_i, producing a guidance system Π_i with at most n colors. Since the Main Lemma prohibits crossing the left border edge of G_i, we infer that inside F the guides of Π_i pass through at most G_i and G_{i+1}, and no other subfactors. Therefore, all the guidance systems Π_i can be combined into a single guidance system with at most $2n$ colors, by using one set of n colors for even-numbered subfactors and another set of n colors for odd-numbered subfactors.

Nodes $x \in X$ that have a witness with $m_1 \leq 1$ and $m_2 \geq$

2**.** As in the previous case, for each subfactor G_i we use the induction assumption to produce an n-color guidance system Π_i with at most n colors for nodes in $X \cap G_i$. Consider a guide π of Π_i that exits the subfactor G_i. Let y_2 be the target of π. By the consistency property of π, we know that there is some y_1 such that the pair $(y_1, y_2) \in E$ is a witness pair for all the source nodes of π. We remove π from Π_i and create a new guide, with a new color, that directs all the sources of π, which are contained in G_i, to y_1. By assumption on $m_1 \leq 1$, we know that the new guide crosses at most one internal border edge. After this modification, the only subfactors that are crossed by the guides of the guidance system are G_i and possibly G_{i-1}. Since we used new colors for the new guides, the new guidance system might now require $2n$ colors. We combine all these guidance systems into a single one using the even/odd strategy from the first case, thus yielding a guidance system with at most $4n$ colors.

Nodes $x \in X$ that have a witness with $m_1, m_2 \geq 2$. In this case we use the forward Ramseyan split. All guides created in this case will point to the right.

Consider a source x with a witness (y_1, y_2). The internal border edges naturally split the words $\mathrm{word}(y_1, x)$ and $\mathrm{word}(x, y_2)$ into $m_1{+}1$ and $m_2{+}1$ words, respectively:

$$\begin{aligned} \mathrm{word}(y_1, x) &= v_0 \cdot v_1 \cdot \ldots \cdot v_{m_1} \\ \mathrm{word}(x, y_2) &= w_0 \cdot w_1 \cdot \ldots \cdot w_{m_2}. \end{aligned}$$

The first letter of v_0 is the label of y_1. The last letter of v_{m_1} and also the first letter of w_0 is the label of x. The last letter of w_{m_2} is the label of y_2 (recall that we do not count an external border edge, so y_1 or y_2 might be outside F). Furthermore, each two consecutive internal border edges are not only visible, but also neighbouring, by definition of subfactors. Hence, the values $\alpha(\mathrm{word}(y_1, x))$ and $\alpha(\mathrm{word}(x, y_2))$ are determined by the first two parts and the last part:

$$\begin{aligned} &\text{(i)} \quad \alpha(\mathrm{word}(y, x)) = \alpha(v_0) \cdot \alpha(v_1) \cdot \alpha(v_{m_y}) \\ &\text{(ii)} \quad \alpha(\mathrm{word}(x, z)) = \alpha(w_0) \cdot \alpha(w_1) \cdot \alpha(w_{m_z}). \end{aligned} \tag{5}$$

Let us fix six values $s_1, \ldots, s_6 \in S$. By splitting the set X into at most $|S|^6$ parts, each requiring a separate guidance system, we can assume that each x has a witness where

$$\begin{aligned} &s_1 = \alpha(v_0) \qquad s_2 = \alpha(v_1) \qquad s_3 = \alpha(v_{m_1}) \\ &s_4 = \alpha(w_0) \qquad s_5 = \alpha(w_1) \qquad s_6 = \alpha(w_{m_2}). \end{aligned}$$

We consider witnesses satisfying the assumptions above.

For each $i = 0, \ldots, k$ we will create a guidance system Π_i that will provide witnesses for all elements of X in the union of subfactors $G_1 \cup \cdots \cup G_i$. The guidance system will use three colors, and will have the following additional property: if e is a word edge that connects a subfactor G_j with G_{j+1}, then at most 2 guides pass through e, all of them directed left-to-right, and at most one of them exits G_{j+1}.

The induction base, when $i = 0$ is immediate, since the union of factors is empty. We now show how to extend Π_i by adding a subfactor G_{i+1} (only the case $i{+}1 \leq k{-}2$ is of interest because of the assumption $m_2 \geq 2$). To this end we need a guidance system Π that induces a witness function for all positions of X inside G_{i+1}. Surprisingly enough, we will *not* apply the Main Lemma to G_{i+1}, as we have:

Claim 1. *There is a single consistent guide π that induces a witness function for all sources in $X \cap G_{i+1}$.*

Let $\Pi_{i+1} := \Pi_i \cup \{\pi\}$ and let π exit the subfactor G_{i+1} towards G_{i+2}.

Now we show that Π_{i+1} has the property specified above: at most two of its guides exit G_{i+1}, and at most one of them exits G_{i+2}. By induction assumption on Π_i, at most one of the guides of Π_i entering the subfactor G_{i+1}, say π', exits this subfactor, hence at most two guides (π and π') of Π_{i+1} exit G_{i+1}. Furthermore, if there are two such guides and none of them has its target in G_{i+2}, then they may be merged into one by the following:

Claim 2. *If targets of both π and π' are not in G_{i+2} then all source nodes of π can be moved to π'.*

The required property of Π_{i+1} is thus demonstrated, which completes the proof of induction step of the Main Lemma as well as the entire proof of Theorem 2 for words.

V. Applications

In this section, we present two applications of our results. The first application is a class of XML documents for which emptiness of XPath is decidable. The second application is a proof that two-variable first-order logic is not captured by XPath, in the presence of two attribute values per node.

Satisfability of XPath. As we said in the introduction, our study on class automata is a first step in a search for structural restrictions on data words and data trees which make XPath satisfiability decidable.

One idea for a structural restriction would be a variant of bounded clique width, or tree width. Maybe bounded clique or tree width are interesting restrictions, but they are not relevant in the study of class automata. This is because bounded clique width or tree width, when defined in the natural way for data trees, guarantees decidable satisfiability for a logic far more powerful than class automata: MSO with navigation and equal data value predicates.

Here we provide a basic example of a restriction on inputs that works for class automata but not for MSO. A data tree is called *bipartite* (bipartite refers to the data) if its nodes can be split into two connected (by the child relation) sets X, Y such that every class has at most one node in X and at most one node in Y.

Satisfiability of MSO, or even FO, with navigation and data equality predicates is undecidable even for bipartite data

words. For instance, a solution to the Post Correspondence Problem can be encoded in a bipartite data word.

This coding, however, cannot be captured by class automata. In the appendix, we prove the following theorem. The proof uses semilinear sets.

Theorem 4. *On bipartite data trees, emptiness is decidable for class automata, and therefore also for XPath.*

Multiple attributes. Heretofore, we have studied data trees, which model XML documents where each node has one data value. In this section, and this one only, we consider the situation where each node x has n data values. Formally, an n-data tree consists of a tree t over the finite alphabet and functions $d_1, \ldots, d_n$ which map the tree nodes to data values. How does XPath deal with multiple data values? Instead of $y_1 \sim y_2$ and $y_1 \not\sim y_2$, we can use any formula of the form

$$d_i(y_1) = d_j(y_2) \qquad \text{where } i, j \in \{1, \ldots, n\}$$

or its negation (for inequality). For $n > 1$ we need more information than just the partitions of nodes into classes of $\sim_i$, for each $i \in \{1, \ldots, n\}$. An example is the property "every node has the same data value on attributes 1 and 2".

How do we extend class automata to read n-data trees? For one data value, the class condition is a language over the alphabet $\Gamma \times \{0,1\}$. For n data values, the class condition is a language over the alphabet $\Gamma \times \{0,1\}^n$. An n-data tree $(t, d_1, \ldots, d_n)$ is accepted if there is an output s of the transducer on t such that for every data value d, the tree

$$s \otimes d_1^{-1}(d) \otimes \cdots \otimes d_n^{-1}(d)$$

is accepted by the class condition. By the same technique as in the proof of Theorem 1, we can prove that the automata capture XPath.

A consequence is that for $n \geq 2$, XPath does not capture two-variable first-order logic (unlike the case of $n = 1$). This was an open question.

Theorem 5. *The following (two-variable) property*

$$\psi = \forall x \forall y \quad d_1(x) = d_1(y) \iff d_2(x) = d_2(y)$$

cannot be defined by a boolean query of XPath.

VI. Classification of automata on data words

In this section, we show how all existing models of automata over data words can be presented as special cases of class automata. In particular, we show how alternating one register automata [5] are captured by a restriction of class automata. We do the comparison for data words only.

We also characterize expressibility of the relevant subclasses of class automata by (restrictions of) counter automata.

Counter automata. Let C be a finite set of names for counters that store natural numbers. A counter automaton is a nondeterministic finite state automaton, whose transitions, allowing also ϵ-transitions, are labeled by counter operations. For each counter name c we have the operations: increment ($c := c + 1$), decrement ($c := c - 1$) and zero test ($c = 0?$). There are restrictions on executing transitions labeled by decrement or zero test: for the decrement, the counter must have value at least 1, for the zero test the counter must have value 0. A counter automaton accepts if in an accepting state all counters equal zero. We write $\mathscr{A}_0$ for the class of counter automata. We identify the following subclasses of $\mathscr{A}_0$.

- $\mathscr{A}_1$: Counter automata without zero tests.
- $\mathscr{A}_2$: Presburger automata. Counter values are in $\mathbb{Z}$. Only increments and decrements are allowed, and no zero tests. Decrements can always be executed. It is easy to encode this as a counter automaton, by representing an integer as the difference of two naturals.
- $\mathscr{A}_3$: Gainy counter automata. Like unrestricted counter automata, but each state must have a self-loop ϵ-transition labeled by $c := c + 1$, for every $c \in C$.

Unlike the unrestricted model, each of the three restricted models has decidable emptiness. For counter automata without zero tests, the underlying problem is reachability for vector addition systems, see [14]. For Presburger automata, the proof uses semilinear Parikh images. For gainy counter automata, the proof uses well-structured transition systems, see e.g. [5].

Restrictions on class condition. Recall the ingredients of a class automaton: a transducer f from the input alphabet Σ to the work alphabet Γ, and a class condition, which is a regular language L over the alphabet $\Gamma \times \{0,1\}$. We write $\mathscr{C}_0$ for all class automata. We identify three restrictions on the class condition L, which limit the way it inspects a word $w \otimes X$, with $w \in \Gamma^*$ and X a set of positions in w. They yield three subclasses of class automata:

- $\mathscr{C}_1$: The class condition L is *local*, i.e., membership $w \otimes X \in L$ depends only on the labels of positions $x \in X$. It does not depend on labels of positions outside X.
- $\mathscr{C}_2$: The class condition is *commutatively local*, i.e., membership $w \otimes X \in L$ depends only on the multiset

  $$\{w|Y : Y \text{ is a maximal interval contained in } X\} \subseteq \Gamma^*.$$

  In the above, we write $w|Y$ for the substring of w corresponding to positions from Y. An interval means a connected set of positions.
- $\mathscr{C}_3$: The class condition is a *tail language*, i.e., there is a regular language K such that $w \otimes X \in L$ if and only if K contains every suffix of $w \otimes X$ that begins in a position from X.

In the appendix we prove that $\mathscr{C}_1$ is equivalent to the model of data automata from [3]. We also show that $\mathscr{C}_3$ captures alternating one register automata (the proof also

Model of counter automata	Class condition	Complexity of emptiness	Languages of data words captured
Counter automata	No restriction	Undecidable	(Extended Regular) XPath
Counter automata w/o zero tests	Local	Decidable, EXPSPACE-hard	Data automata, FO2($<, +1, \sim$)
Presburger automata	Commutatively local	NP-complete	FO2($+1, \sim$)
Gainy counter automata	Tail language	Decidable, non-primitive rec.	Alternating one register automata, one register freeze LTL

Table I
CLASSIFICATION OF AUTOMATA ON DATA WORDS.

works for trees). In consequence $\mathscr{C}_3$ captures weak two-pebble automata [11], as they are simulated by alternating one register automata [15]. The same applies to the top view weak pebble automata introduced in [15].

Classification. We will show that the restrictions on counter automata match the restrictions on class conditions. To compare languages of words without data and languages of data words, we use the following definition. Let L be a set of data words over an alphabet Σ, and $\pi : \Sigma \to \Gamma \cup \{\epsilon\}$ a relabeling of Σ that may erase some letters. We define $\pi(L) \subseteq \Gamma^*$ to be the words without data obtained from L by ignoring the data and applying π to the underlying word. Any such language $\pi(L)$ is called a projection of L.

Theorem 6 (Classification theorem)**.** *For $i \in \{0, 1, 2, 3\}$ the following language classes are equal:*

- *Languages of words without data recognized by counter automata from $\mathscr{A}_i$.*
- *Projections of languages of data words recognized by class automata from $\mathscr{C}_i$.*

The translations in both directions are effective in the following sense. For any automaton $\mathcal{A} \in \mathscr{A}_i$, one can compute an automaton $\mathcal{C} \in \mathscr{C}_i$ and a projection π such that the language of $\mathcal{A}$ is the projection under π of the language of $\mathcal{C}$. Conversely, for any $\mathcal{C} \in \mathscr{C}_i$ and π, one can compute an automaton $\mathcal{A} \in \mathscr{A}_i$ recognizing the projection under π of the language of $\mathcal{C}$. Consequently, the emptiness problems for $\mathscr{A}_i$ and $\mathscr{C}_i$ can be reduced to each other. In most of the cases the reductions are polynomial.

The theorem is illustrated in Table I. In the third column, the table includes the complexities of deciding emptiness for the counter automata. In the last column, the table lists some properties of data words captured by the automata.

The table does not include nondeterministic automata with multiple registers of [9]. Their projections are regular word languages. By [12] we know that nondeterministic register automata are captured by data automata, and therefore they correspond to some restriction on the class condition in a class automaton. But the construction in [12] is quite sophisticated, and we do not know what this restriction is.

We are curious how our undecidable model over data words and trees relates to other undecidable models, such as strong one-way pebble automata, alternating automata with multiple registers, or two-way automata with registers, or combinations thereof.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

[1] M. Benedikt, W. Fan, and F. Geerts, "XPath satisfiability in the presence of DTDs," *J. ACM*, vol. 55, no. 2, 2008.

[2] F. Geerts and W. Fan, "Satisfiability of XPath queries with sibling axes," in *DBPL*, 2005, pp. 122–137.

[3] M. Bojańczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David, "Two-variable logic on words with data," in *LICS*, 2006, pp. 7–16.

[4] M. Bojańczyk, A. Muscholl, T. Schwentick, and L. Segoufin, "Two-variable logic on data trees and XML reasoning," *J. ACM*, vol. 56, no. 3, 2009.

[5] S. Demri and R. Lazić, "LTL with the freeze quantifier and register automata," *ACM Trans. Comput. Log.*, vol. 10, no. 3, 2009.

[6] M. Jurdziński and R. Lazić, "Alternation-free modal mu-calculus for data trees," in *LICS*, 2007, pp. 131–140.

[7] D. Figueira, "Satisfiability of downward XPath with data equality tests," in *PODS*, 2009, pp. 197–206.

[8] ——, "Forward-XPath and extended register automata on data-trees," in *ICDT*, 2010, to appear.

[9] M. Kaminski and N. Francez, "Finite-memory automata," *Theor. Comput. Sci.*, vol. 134, no. 2, pp. 329–363, 1994.

[10] L. Segoufin, "Automata and logics for words and trees over an infinite alphabet," in *CSL*, 2006, pp. 41–57.

[11] F. Neven, T. Schwentick, and V. Vianu, "Finite state machines for strings over infinite alphabets," *ACM Trans. Comput. Log.*, vol. 5, no. 3, pp. 403–435, 2004.

[12] H. Björklund and T. Schwentick, "On notions of regularity for data languages," in *FCT*, 2007, pp. 88–99.

[13] T. Colcombet, "A combinatorial theorem for trees," in *ICALP*, 2007, pp. 901–912.

[14] E. W. Mayr, "An algorithm for the general Petri net reachability problem," in *STOC*, 1981, pp. 238–246.

[15] T. Tan, "On pebble automata for data languages with decidable emptiness problem," in *MFCS*, 2009, pp. 712–723.

VII. Simplifying the query

The goal of this section is to reduce Theorem 2 to the case when $\varphi[x, y_1, y_2]$ is a simplified query (see Section III-D). This simplification is achieved in several steps.

A. Generalized witness functions

Fix any number $n \in \mathbb{N}$, although we will be mainly interested in $n \in \{1, 2\}$. Consider a regular query $\varphi[x, y_1, \ldots, y_n]$ over trees. Consider now a tree t together with a set E of n-tuples of nodes in t. As before, the idea is that E gives a constraint on the witness variables. A *witness tuple* for a node x is a tuple $(y_1, \ldots, y_n) \in E$ such that $(x, y_1, \ldots, y_n)$ is selected by φ in t. In this case, we say that x is a *source*, and y_i is an *i-th witness* for x (the other variables are quantified existentially).

A *witness function* for φ and a set of source nodes X in (t, E) is a function which assigns to each node $x \in X$ some witness (an i-th witness for some i, with i depending on x).

We say that a regular query $\varphi[x, y_1, \ldots, y_n]$ has *witness functions of guidance width* m if for every tree t and every choice E of n-tuples of nodes of t, there is a witness function for φ and any set X of source nodes in (t, E) of guidance width at most m. A query φ has *bounded width witness functions* if some such m exists.

The following lemma says about the case of $n = 1$:

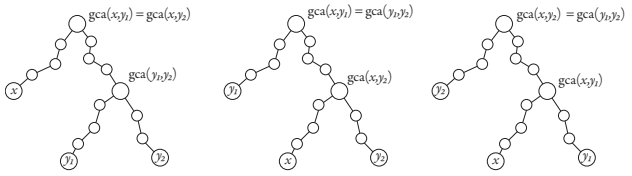
Lemma 3. *Every binary query $\varphi[x, y]$ has bounded width witness functions.*

B. Three arrangements

By an *arrangement* of the nodes x, y_1, y_2 in a tree we mean the information on how these nodes, and their greatest common ancestors

$$\mathrm{gca}(x, y_1) \quad \mathrm{gca}(x, y_2) \quad \mathrm{gca}(y_1, y_2)$$

are related with respect to the descendant ordering. We distinguish three different arrangements, pictured below.



These arrangements correspond, respectively, to the following situations.

$$\mathrm{gca}(x, y_1) = \mathrm{gca}(x, y_2) \leq \mathrm{gca}(y_1, y_2) \qquad \text{(A1)}$$

$$\mathrm{gca}(x, y_1) = \mathrm{gca}(y_1, y_2) \leq \mathrm{gca}(x, y_2) \qquad \text{(A2)}$$

$$\mathrm{gca}(x, y_2) = \mathrm{gca}(y_1, y_2) \leq \mathrm{gca}(x, y_1) \qquad \text{(A3)}$$

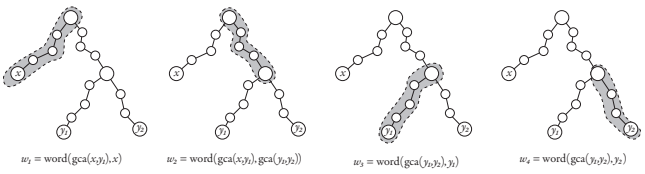
The arrangements are not contradictory, for instance the case $x = y_1 = y_2$ is covered by all three.

Lemma 4. *We may assume without loss of generality that all the triples selected by φ have the same arrangement.*

Proof: Otherwise we can split φ into a union of three queries, one for each arrangement, and then combine the three separate presentation systems. □ ■

C. Path based queries

Let us fix one of the arrangements. There are four words w_1, w_2, w_3, w_4 that will interest us. These are shown on the picture below for the arrangement (A1) only, but the reader can easily see the situation for all other arrangements.



.

The query is called *path based* if its truth value depends only on some regular properties of the four words $w_1, \ldots, w_4$. I.e., there are regular word languages $L_1, \ldots, L_4$ such that φ selects precisely those triples (x, y_1, y_2) where $w_1 \in L_1, \ldots, w_4 \in L_4$.

Lemma 5. *We may assume without loss of generality that φ is path based.*

Proof: We first claim that for any ternary regular query φ which selects only tuples in one arrangement, there is a functional transducer f and a path based query γ such that $\varphi = \gamma \circ f$, i.e. for a tree t the set $\varphi(t)$ of tuples selected by φ in t is the same as the set of tuples selected by γ in $f(t)$. The claim can be proved using logical methods (the transducer computes MSO theories) or using automata methods (the transducer computes state transformations).

To prove the lemma, we need to show that if Theorem 2 is true for the path based queries γ, then it is also true for arbitrary ternary queries φ. But this is straightforward, as φ and γ have the same witness functions in trees t and $f(t)$, respectively. □ ■

In the proofs later on we will use algebra, hence we prefer to define the query φ in algebraic terms. We assume a monoid morphism

$$\alpha : \Sigma^* \to S$$

such that membership $(x, y_1, y_2) \in \varphi(t)$ depends only on the values assigned by α to the words $w_1, \ldots, w_4$. In other words, there is a set of *accepting pairs* $F \subseteq S^4$ such $\varphi[x, y_1, y_2]$ holds if and only if

$$(\alpha(w_1), \ldots, \alpha(w_4)) \in F \ .$$

D. Composing guidance systems

In the sequel we will need to compose the guidance systems as outlined in the lemma below. For two partial

functions f, g on the set of nodes of a tree, the composition $g \circ f$ has the same domain as f, and is defined as follows:

$$(g \circ f)(x) = \begin{cases} g(f(x)) & \text{if g is defined on } f(x) \\ f(x) & \text{otherwise.} \end{cases}$$

Lemma 6. *Let f, g be partial functions on the set of nodes of a tree, of guidance width m_1 and m_2, respectively. Then their composition $g \circ f$ is of guidance width at most $2m_1m_2$.*

Proof: Fix a tree t together with some m_1- and m_2-color guidance systems Π_f and Π_g, inducing f and g, respectively. We will show existence of a $2m_1m_2$-color guidance system for $g \circ f$.

As the first step, combine Π_f and Π_g as follows: a node x is first guided by Π_f, and then, if g is defined on $f(x)$, guided by Π_g to its final destination. Formally, Π contains those guides of Π_f whose destination node is not in the domain of g; and moreover a number of guides that are composed of at least two guides, to be described now. Fix a pair of colors (k, l), where k is a color used in Π_f and l is used in Π_g. A composed guide, colored by the pair (k, l), consists of one l-colored guide from Π_g, say π, and all those k-colored guides from Π_f whose destination node is a source node of π. We will focus on the latter 'composed' guides only. (A 'non-composed' guide in Π, say colored k, may be safely considered as colored by (k, l), for any l.)

The above coloring, using m_1m_2 colors, is not satisfactory as same colored guides may be in conflict. We will show how to resolve these conflicts by introducing an additionally distinguishing piece of data into the colors. Fix a color pair (k, l) as above. Note that a conflict may only arise when the Π_g-part (l-colored in Π_g) of one (k, l)-colored guide, say π_1, conflicts with the Π_f-part (k-colored in Π_f) of another same colored guide, say π_2. Consider an undirected graph G, whose nodes are all (k, l)-colored guides; there is an edge between π_1 and π_2 in the graph if the abovementioned conflict arises.

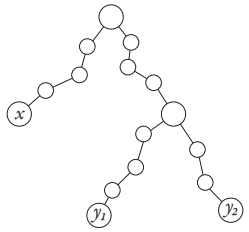
We claim that the graph G is a forest, i.e., a disjoint union of trees. Towards a contradiction, suppose that G has a cycle consisting of n pairwise different guides $\pi_1, \ldots, \pi_n$. Take $\pi_{n+1} = \pi_1$. Let $x_1, \ldots, x_n$ denote arbitrarily chosen nodes witnessing the conflicts, i.e., x_i belongs to the guides π_i and π_{i+1}. In π_{i+1}, for any $i \leq n$, there is a unique path from x_i to x_{i+1} (take x_{n+1} as x_1), denote it p_i; p_i always uses a path of a guide from Π_f, colored k, and a path of a guide from Π_g, colored l. As the k-colored guides never conflict, and likewise the l-colored ones, the k-colored part of p_i is separated from the same colored part of p_{i+1} by at least one l-colored edge; thus the paths p_i are nonempty, i.e., $x_i \neq x_{i+1}$. Assume that $x_1, \ldots, x_n$ are pairwise distinct (if this is not the case, i.e., $x_i = x_j$, consider $x_i, \ldots, x_{j-1}$ instead; and consider $\pi_i, \ldots, \pi_{j-1}$ instead of $\pi_1, \ldots, \pi_n$).

Now we are prepared to obtain a contradiction, thus proving that G is a forest. If two paths p_i and p_{i+1} share an edge adjacent to x_{i+1}, the edge may be removed from both paths; this clearly forces x_{i+1} to be moved appropriately. Thus the paths can be made edge-disjoint; moreover we keep the x_i nodes pairwise distinct, argued as above. Hence the paths $p_1, \ldots, p_n$ form a cycle in the tree t, a contradiction.

Knowing that G is a forest, we may easily label its nodes by two numbers $1, 2$, level by level, starting from an arbitrary leaf in any connected component. This additional numbering, added to the colors of the guides in Π, eliminates the problematic conflicts and makes Π a $2m_1m_2$-color guidance system as required. ■

E. Arrangement (A1)

In this section we show that Theorem 2 holds if all triples selected by $\varphi[x, y_1, y_2]$ have arrangement (A1), pictured below.



Suppose $\tau[x, y]$ is a binary query, and $\sigma[y, y_1, y_2]$ is a ternary query. We define the following ternary query

$$\tau \circ_y \sigma[x, y_1, y_2] \quad = \quad \exists y \ \tau[x, y] \wedge \sigma[y, y_1, y_2] \ .$$

Lemma 7. *Let τ, σ be as above. If τ and σ have bounded width witness functions then $\tau \circ_y \sigma$ too.*

Proof: By considering the witness function for $\tau \circ_y \sigma$ obtained as a composition and applying Lemma 6. □ ■

Lemma 8. *We may assume without loss of generality that $x = \mathrm{gca}(y_1, y_2)$.*

Proof: By the considerations in Section VII-C, we know that a triple (x, y_1, y_2) is selected by φ if and only if the images, under the morphism α, of the four path words w_1, w_2, w_3, w_4 belong to a designated set $F \subseteq S^4$ of accepting tuples.

Let $s_1, \ldots, s_4 \in S$. Let τ_{s_1,s_2} be the binary query that selects a pair (x, y) if

$$\alpha(\mathrm{word}_t(\mathrm{gca}(x, y), x)) = s_1$$
$$\alpha(\mathrm{word}_t(\mathrm{gca}(x, y), y)) = s_2 \ .$$

Likewise, let σ_{s_3,s_4} be the ternary query that selects a triple (y, y_1, y_2) if

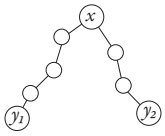
$$\mathrm{gca}(y_1, y_2) = y$$
$$\alpha(\mathrm{word}_t(y, y_1)) = s_3$$
$$\alpha(\mathrm{word}_t(y, y_2)) = s_4 \ .$$

The queries τ_{s_1,s_2} and σ_{s_3,s_4} can be joined to define φ, in the following way.

$$\varphi = \bigcup_{(s_1,s_2,s_3,s_4)\in F} \tau_{s_1,s_2} \circ_y \sigma_{s_3,s_4}.$$

By Lemma 7, we see that the width of witness functions for φ is bounded by the widths of the witness functions for the τ queries, which is bounded by Lemma 3, and the width of the witness functions for the σ queries. The latter are queries where the first variable is the gca of the second and third variables, which concludes the proof of the lemma. □ ■

Thanks to the above lemma, we are left with a query φ that selects triples in the arrangement pictured below (for future reference let us call this arrangement *trivial*).



We will provide a 2-color guidance system that induces a witness function for φ in (t, E). This is guaranteed by the following lemma:

Lemma 9. *Let φ be any (not necessarily regular) query that select only nodes in a trivial arrangement. Then φ has witness functions of guidance width 2.*

Proof: The guidance system is constructed in a single root-to-leaf pass.

More formally, for each set X of nodes that is closed under ancestors, we will provide a guidance system Π_X that directs each source in X to some witness, either y_1 or y_2. The guidance system will have the additional property that no tree edge is traversed by two guides.

The guidance system is constructed by induction on the size of X. The induction base, when X has no nodes, is straightforward. We now show how Π_X should be modified when adding a single x node to X. Since all guides in Π_X originate in nodes from X, any guide that passes through x must also pass through its parent. Using the additional assumption, we conclude that at most one guide π from Π_X passes through x. In particular, either the left or right subtree of x has no guide passing through it. If x has no witness, nothing needs to be done. If x has a witness (y_1, y_2), then we create a new guide that connects x to the node y_i which is in the child subtree of x without guides. □ ■

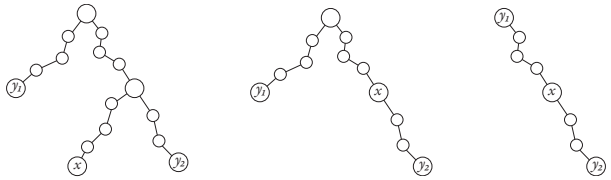
For arrangment (A1) the proof of Theorem 2 is thus completed.

F. Arrangements (A2) *and* (A3)

For the remaining arrangments, in this section we only show how they can be reduced to the simplified ones. Formally, we provide here a link between the Main Lemma and Theorem 2, missing in the body of the paper. We formulate Theorem 7 below, which we will use in this section, and which follows easily from the tree version of Main Lemma (forthcoming Lemma 10). To state the theorem, recall the notion of consistent guidance system introduced in Section IV: a guidance system in a graph tree is called *consistent* wrt. a given ternary query if each of guides obeys the following uniqueness requirement: whenever a set $Z \subseteq X$ of source nodes is guided to the same node y, then there is a pair (y_1, y_2) that is a witness pair for all nodes Z, with $y_1 = y$ or $y_2 = y$. Below, the consistency property will make it possible to combine two guidance systems appropriately.

Theorem 7. *Every simplified regular query has witness functions of bounded guidance width. Furthermore, a consistent guidance system always exists (of the required bounded width).*

Now using Theorem 7 we prove Theorem 2 for arrangements (A2) and (A3). By symmetry, we only consider the arrangement (A2). We simplify the arrangement in two steps. First we claim that without loss of generality x can be assumed to be an ancestor of y_2 – this may be shown by essentially the same technique as in Lemma 8 hence we omit the details. Second, we show that y_1 can be assumed to be an ancestor of x and y_2. The arrangement (A2), as well as its two successive simplifications, are pictured below.



Let our starting arrangement now be the middle one in the picture above, i.e. we assume that the first simplification has been already applied. Wlog we may assume that y_1 is in the left subtree, and x in the right subtree of the $\mathrm{gca}(x, y_1)$ node (thus we again split into two sub-cases), and that both x and y_1 are not equal to $\mathrm{gca}(x, y_1)$.

By the considerations in Section VII-C, we know that a triple (x, y_1, y_2) is selected by φ in t if and only if the images, under the morphism α, of the three path words:

$$\mathrm{word}_t(x, \mathrm{gca}(x, y_1)),\ \mathrm{word}_t(x, y_2),\ \mathrm{word}_t(\mathrm{gca}(x, y_1), y_1),$$

belong to a designated set $F \subseteq S^3$ of accepting tuples.

Fix $(s_1, s_3, s_3) \in F$ (thus the guidance system will be a disjoint union over all triples $(s_1, s_2, s_3) \in F$). Let $\sigma_{s_1,s_2}[x, y, y_2]$ be a simplified query that selects a triple (x, y, y_2) if

$$\alpha(\mathrm{word}_t(x, y)) = s_1$$
$$\alpha(\mathrm{word}_t(x, y_2)) = s_2$$
$$y < x \leq y_2$$

and x is in the right subtree of y. Consider an arbitrary graph tree (t, E) over which φ is evaluated. The idea now is that the query σ_{s_1,s_2} is evaluated over a modified graph tree (t, E_{s_3}), where a pair (y, y_2) is in E_{s_3} iff $(y_1, y_2) \in E$ for some y_1 such that

- $y = \mathrm{gca}(y_1, y_2)$,
- y_1 is in the left subtree of y,
- $\alpha(\mathrm{word}_t(y, y_1)) = s_3$.

(Intuitively, every edge $(y_1, y_2) \in E$ is 'moved' to $(y = \mathrm{gca}(y_1, y_2), y_2)$, but only if $\alpha(\mathrm{word}_t(y, y_1)) = s_3$.)

Let's fix now a graph tree (t, E) and a set X of source nodes wrt. φ and E. By Theorem 7 we know that σ_{s_1,s_2} has a witness function induced by a consistent m-color guidance system Π (m does not depend on t, E or X).

Let $\tau_{s_3,\Pi}[y, y_1, y_2]$ be a ternary query that selects a triple (y, y_1, y_2) in (t, E) if

- $y = \mathrm{gca}(y_1, y_2)$, y_1 is in the left subtree of y and y_2 in the right one,
- y is one of the target nodes of Π,
- $(y_1, y_2) \in E$ is a witness pair for *all* nodes x that are guided by Π to y (by consistency of Π, such a pair (y_1, y_2) exists for any target node y of Π),
- $\alpha(\mathrm{word}_t(y, y_1) = s_3$.

Query $\tau_{s_3,\Pi}$ depends on the guidance system Π, we thus implicitly assume that Π is included in the labeling of a tree. Query $\tau_{s_3,\Pi}$ is not a regular one in general, but it selects only triples in trivial arrangement. Applying Lemma 9 we get a 2-color guidance system that induces a witness function for $\tau_{s_3,\Pi}$. The two guidance systems can be then combined into one $4m$-color guidance system due to Lemma 6. This guidance system induces a witness function for φ and X in (t, E).

As the graph tree (t, E) was chosen arbitrary, this completes the proof.

VIII. Theorem 2 for trees

Fix a query $\varphi[x, y_1, y_2]$ that satisfies the simplifying assumptions from Section III-D. As in the word case, we can assume that query is represented by a morphism $\alpha : \Sigma^* \to S$ in the following sense: φ only selects triples (x, y_1, y_2) with $y_1 \leq x \leq y_2$ and where the values

$$\alpha(\mathrm{word}_t(y_1, x)) \in S \qquad \alpha(\mathrm{word}_t(x, y_2)) \in S. \tag{6}$$

belong to an accepting set of pairs $Acc \subseteq S^2$. We fix this morphism for the rest of this section.

Factors.: As in the word case, we distinguish two types of edges in a graph tree (t, E). The *tree edges* are edges that connect parents with children, as well as a dummy edge going into the root of the tree and dummy edges going out of the leaves. The *class edges* are the edges from E.

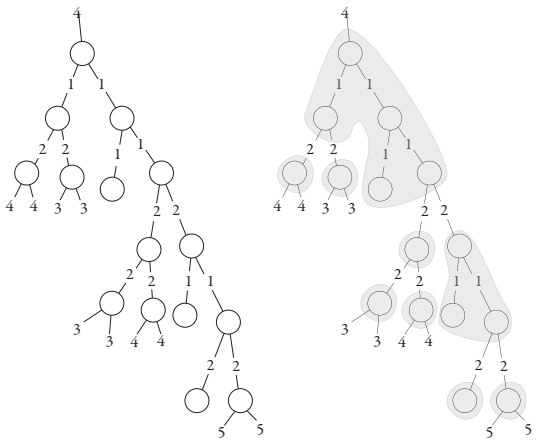
We will use a forward Ramseyan split for the morphism α, as given by Theorem 3. The Ramseyan split is defined for every path (root to leaf) in the tree. By the determinism property, the splits agree on common prefixes of the paths, so we can see the split as a labeling of the tree edges, including the dummy edges. Again by determinism, all tree edges from a node to its children have the same label in the split. Wlog we consider only *complete* binary trees, where each non-leaf node has precisely two children.

Two comparable (i.e., belonging to one path) tree edges e and f are called *visible* if all tree edges between e and f are mapped by the split to values strictly smaller than $\sigma(e)$ and $\sigma(f)$. Visible tree edges naturally determine a nested factorization of t in the following way.

A *pre-factor* in a tree t is a connected set of nodes (connected by tree edges) such that if a node x is in the pre-factor, then either all sons of x are in the pre-factor, or none of them. Each pre-factor of t has a root and some leaves (maximal nodes wrt. $\leq$), and inherits its edges from t. We distinguish *internal* edges of a pre-factor, connecting two nodes in that pre-factor, and *external* edges connecting the root or the leaves with their children outside the pre-factor. This includes the tree edge leading to the root of the pre-factor (called the *root edge* of the pre-factor) and the tree edges going out of the leaves (called the *leaf* edges of the pre-factor). Note that external edges may be either proper tree edges, or dummy edges. As the split σ is assumed to be deterministic, all tree edges leaving a given leaf of a pre-factor are assigned the same number. A pre-factor F is called a *factor* in t if it respects the split σ in the following way: the root edge is visible from each of the leaf edges. This means that on each (shortest) path in a factor from its root edge to a leaf edge, numbers assigned by σ to the internal edges on that path are strictly smaller than those assigned to the two external edges. By the height of a factor we mean the greatest number assigned to an internal edge, or 0 if no such edge exists (the case of one-node pre-factor). Additionally, the whole tree t is also a factor if, wlog, we assume that the root dummy edge is visible with all leaf dummy edges; its height at most equals the height of σ.

A *subfactor* of a factor F is any factor $G \subsetneq F$ that is maximal with respect to inclusion. By the definition of factor, two different subfactors of F are disjoint (have disjoint sets of nodes, but possibly share an external edge). Hence each factor F is the disjoint union of its subfactors. We say a subfactor G is an *ancestor* of a subfactor H if their roots are so related. Likewise we talk about a subfactor being a *child* or *parent* of some other subfactor.

A factor together with its decomposition into subfactors is pictured below.



We are now ready to present the tree version of the Main Lemma.

Lemma 10 (Main Lemma)**.** *Fix a factor height h. There is a bound $n \in \mathbb{N}$, depending only on φ and h, such that for every graph tree (t, E), every factor F in t of height h, and every set $X \subseteq F$ of source nodes, there is a witness function for φ and X in (t, E) induced by a consistent guidance systems using at most n colors. Furthermore, this witness function only points to descendants of the root of F.*

The proof of the lemma is by induction on the height h. The number of colors n will depend on h and the size of the monoid S recognizing the query. It will not depend on t. When going from height h to height $h+1$, there will be a quadratic blowup in the number of colors. Therefore, n will be doubly exponential in the height of F. (Note how this differs from the word case, cf. Section IV.)

Since the witness function will be induced by a guidance system, the last assumption in Lemma 10 could be restated as saying that no guide passes through the root edge of F. Theorem 7 is a special case of the Main Lemma when F is the whole tree.

The base case, when the factor F has one or no nodes, is easy (1 color, going downwards, is sufficient). For the induction step, fix a factor F, and assume that there is a bound n sufficient for any factor of smaller height than F, which includes all subfactors of F. Below, subfactors of F are simply called *subfactors*, without explicitly referring to F.

A tree edge of F that is an external edge of one if its subfactors is called a *border edge*. In particular each external edge of F is a border edge. Special care will be paid in our proof to internal (i.e. not external) border edges, i.e., the root edges of all subfactors except the root edge of F itself.

Claim 3. *If two internal border edges in a factor are comparable by the ancestor relation $\leq$ then they are assigned the same value by the split.*

We do the same case distinction as in the word case, regarding the number of internal border edges on the paths from a source to witness nodes. For a node $x \in X$ and a witness (y_1, y_2) we define two numbers m_1, m_2. Let m_1 be the number of internal border edges on the path between y_1 and x, and let m_2 be the number of internal border edges on the path between x and y_2. For technical convenience, we deliberately choose to not to count external border edges. We divide the set X into three parts:

1) Nodes $x \in X$ that have a witness with $m_2 \leq 1$.
2) Nodes $x \in X$ that have a witness with $m_1 \leq 1$ and $m_2 \geq 2$.
3) Nodes $x \in X$ that have a witness with $m_1, m_2 \geq 2$.

We prove the Main Lemma for each of the three parts separately. Next, we combine the three guidance systems into a single guidance system. Our construction will yield two kinds of guides: the *ancestor guides* pointing to the first witness and thus going up a tree; and *descendant guides* pointing to the second witness, and thus going down the tree. Interestingly, ancestor guides will be only created in case 2. All the guides will satisfy the consistency condition required in Lemma 10.

Nodes $x \in X$ that have a witness with $m_2 \leq 1$.: This case works the same way as for words. Consider a subfactor G of F. In this case, each node $x \in X \cap G$ has a descendant witness y_2 that is either in G, or in a child subfactor of G, or perhaps outside F. Apply the induction assumption to G, producing a guidance system Π_G with at most n colors. Since the Main Lemma requires the guidance system to point to descendants of the factor's root, and $m_2 \leq 1$, we infer that inside F the guides of Π_G can only intersect G and its child subfactors, and no other subfactors (it is possible that the guides leave the factor F, though). Therefore, all the guidance systems Π_G can be combined into a single guidance system with at most $2n$ colors, used alternatingly for even and odd depths.

Nodes $x \in X$ that have a witness with $m_1 \leq 1$ and $m_2 \geq 2$.: Here we need a different argument for trees than for words. In this case, for each node $x \in X$ there is an ancestor witness y_1 that is either in the subfactor of x, in the parent subfactor, or outside F. Note that the latter is possible only when x belongs either to the root subfactor of F, or to some of its child subfactors; denote this set of subfactors by $\mathcal{G}_0$. We will construct the guidance system in a step-by-step manner, for all subfactors, according to the ancestor ordering.

Formally speaking, consider a family $\mathcal{G}$ of subfactors that is closed under ancestors and includes $\mathcal{G}_0$. We provide a guidance system $\Pi_{\mathcal{G}}$ of $4n^2 + 3n$ colors that provides witnesses for all nodes of X belonging to the subfactors in $\mathcal{G}$. The construction of $\Pi_{\mathcal{G}}$ is by induction on the number of subfactors in $\mathcal{G}$.

The induction base is when $\mathcal{G}$ equals $\mathcal{G}_0$. For each subfactor $G \in \mathcal{G}_0$, we apply the Main Lemma, for the smaller height, to G and nodes from X that belong to G, yielding a n-color guidance. We combine these guidance systems into

$\Pi_{\mathcal{G}_0}$ as follows: use one set of n colors for the root subfactor, and another set of n colors for all the child subfactors of the root.

For the induction step, suppose that we have already constructed $\Pi_{\mathcal{G}}$ for $\mathcal{G}$, and that $G \notin \mathcal{G}$ is a subfactor whose parent is in $\mathcal{G}$. Consider the guides of $\Pi_{\mathcal{G}}$ that pass through the root edge of G. We apply two distinctions to these guides. First, we use the name *parent guides*, for the guides that originate in the parent subfactor of G, and the name *far guides* for the other guides. Second, we use the name *ending guides* for the guides whose target is in G and the name *transit guides* for the other guides, which continue into a child subfactor of G, or even exit F. Altogether, there are four possibilities: parent transit guides, far ending guides, etc. We assume additionally that there are at most n parent guides and at most $2n$ far and parent guides altogether, and hence at most $2n$ guides entering G. This additional invariant is satisfied by the induction base, and it will be preserved through the construction.

Apply now the induction assumption of the Main Lemma to G and nodes from X that belong to G, yielding a guidance system Π with n fresh colors. We use the name *starting guides* for the guides of Π. We want to combine $\Pi_{\mathcal{G}}$ with Π in such a way that the resulting guidance system still uses at most $4n^2 + 3n$ colors, like $\Pi_{\mathcal{G}}$, and satisfies the additional invariant. If we were to simply take the two systems together, we might end up with an external leaf edge of G which is traversed both by starting and transit guides, which could exceed the bound $2n$ on guides passing through external edges.

We solve this problem as follows. Consider the external leaf edges of G that are traversed by the far transit guides. There are at most $2n$ such edges by our invariant assumption. We will remove all starting guides that pass any of these edges, and find other witnesses for nodes that use these starting guides. This guarantees that the invariant condition is recovered: at most $2n$ guides passes through any external leaf edge of G, and at most n of them are starting guides. These other witnesses will be ancestors. This explains why the induction starts with $\mathcal{G}$ containing the root subfactor and its children, since these are the subfactors that may have ancestor witnesses outside the whole factor F (recall that passing through the root edge is not counted in m_1). The statement of Main Lemma does not allow guides that pass through the root edge of F.

The removing of starting guides proceeds as follows. Let e be an external leaf edge of G that is traversed by a far transit guide, which has color j in the guidance system $\Pi_{\mathcal{G}}$. Let π be a starting guide, which has color i in Π, that also traverses e, with y_2 its target node. By the consistency property of π, there is some y_1 such that $(y_1, y_2) \in E$ is a witness pair for all source nodes of π. Note that by assumption on $m_1 \leq 1$, the node y_1 is either in the subfactor G or its parent. We create an ancestor guide with a fresh color that connects all the source nodes of π to y_1. The color of this guide, which we call an *ancestor color*, will take into account three parameters: the colors i and j, as well as a parity bit $b \in \{0, 1\}$. The parity bit is 0 if and only if G has an even number of ancestor subfactors. We use the triple (i, j, b) for the color name.

We will show that this new ancestor guide does not conflict with any other ancestor guide with the same color. Each new ancestor guide is contained in G and possibly its parent subfactor H, by assumption on $m_1 \leq 1$. Inside the subfactor G there is at most one ancestor guide of each color, so there are no collisions inside G. One could imagine, though, that a new ancestor guide π with color (i, j, b) collides inside the parent subfactor H with some other ancestor guide π' of the same color. Since the colors of π and π' agree on the parity bit b, we conclude that the guide π' cannot originate in H, which has a different parity than G. Therefore, π' must originate in some other child subfactor of H, call it G', that had been previously added to $\mathcal{G}$. Since the color of π' is also (i, j, b), we conclude that j was the color of a far transit guide in G'. This is impossible, since a far transit guide in G' or G must originate not in H, but in an ancestor of H, and therefore there would be a collision in the root of H.

The ancestor guides created above are the only ancestor guides in our solution. In the subfactors where they are used, the ancestor guides have the target in the parent subfactor.

Let us count the number of colors used. We need $2n$ colors for the transit guides, and n colors for the starting guides. For the ancestor guides, we need $4n^2$ colors. Altogether, we need $4n^2 + 3n$ colors. Note that all the guides satisfy the consistency condition required by Lemma 10.

Nodes $x \in X$ that have a witness with $m_1, m_2 \geq 2$.: In this case, each node $x \in X$ has a witness (y_1, y_2) such that the path from y_1 to x, as well as the path from x to y_2, passes through at least two internal border edges. This case is the only one where we use the forward Ramseyan split.

Consider a source x with a witness (y_1, y_2). The internal border edges split naturally $\mathrm{word}(y_1, x)$ and $\mathrm{word}(x, y_2)$ into m_1+1 and m_2+1 words, respectively:

$$\begin{aligned} \mathrm{word}(y_1, x) &= v_0 \cdot v_1 \cdot \ldots \cdot v_{m_1} \\ \mathrm{word}(x, y_2) &= w_0 \cdot w_1 \cdot \ldots \cdot w_{m_2}. \end{aligned}$$

The first letter of v_0 is the label of y_1. The last letter of v_{m_1} and also the first letter of w_0 is the label of x. The last letter of w_{m_2} is the label of y_2 (recall that we do not count an external border edge, so y_1 or y_2 might be outside F). Furthermore, each two consecutive internal border edges are not only visible, but also neighbouring, by our last claim. Hence, as we have a forward Ramseyan split (cf. (4) in Section IV), the values $\alpha(\mathrm{word}(y_1, x))$ and $\alpha(\mathrm{word}(x, y_2))$

are determined by the first two parts and the last part:

$$\begin{array}{lrcl} \text{(i)} & \alpha(\text{word}(y,x)) & = & \alpha(v_0)\cdot\alpha(v_1)\cdot\alpha(v_{m_y}) \\ \text{(ii)} & \alpha(\text{word}(x,z)) & = & \alpha(w_0)\cdot\alpha(w_1)\cdot\alpha(w_{m_z}). \end{array} \quad (7)$$

Let us fix six values $s_1, \ldots, s_6 \in S$. By splitting the set X into at most $|S|^6$ parts, each requiring a separate guidance system, we can assume that each $x \in X$ has a witness where

$$\begin{array}{lll} s_1 = \alpha(v_0) & s_2 = \alpha(v_1) & s_3 = \alpha(v_{m_1}) \\ s_4 = \alpha(w_0) & s_5 = \alpha(w_1) & s_6 = \alpha(w_{m_2}). \end{array}$$

We will only consider witnesses that satisfy the assumptions above.

We now proceed to create the guidance system. As in the case $m_1 \leq 1$, the guidance system will be defined for a family $\mathcal{G}$ of subfactors that is closed under ancestors. The guidance system will use at most 3 colors, and will have the following additional invariant property: if e is an edge that connects a subfactor G with a child subfactor H, then at most two guides pass through e. Furthermore, if exactly two guides pass through e, then one of the guides has its target in H.

The construction is by induction on the number of subfactors in $\mathcal{G}$. The induction base when $\mathcal{G}$ has no subfactors is obvious. Below we show how to modify a guidance system $\Pi_{\mathcal{G}}$ for $\mathcal{G}$ when adding a new subfactor G.

Consider the (at most two) guides of $\Pi_{\mathcal{G}}$ that pass through the edge connecting G to $\Pi_{\mathcal{G}}$. As in the case $m_1 \leq 1$, we use the term *transit guide* for the guides of $\Pi_{\mathcal{G}}$ that enter G through its root and exit through one of its external leaf edges. By the invariant assumption, there is at most one transit guide.

We now define a guidance system Π for the nodes in G, which we will next combine with $\Pi_{\mathcal{G}}$.

Claim 4. *There is a one color guidance system* Π *defining a witness function for all nodes in* $G \cap X$.

Proof: For each node $x \in G \cap X$, choose the lexicographically first witness $y_2 > x$ that satisfies the assumptions on the six images in the semigroup, call it y_x. Let Y be all these witnesses y_x; this set is an antichain with respect to the descendant relation. For each $y \in Y$, let X_y be the chain of nodes x which are witnessed by y. By the lexicographic assumption, if $y, y' \in Y$ are such that y is lexicographically before y', then no element from X_y has an ancestor in $X_{y'}$. Consequently, if we define π_y to be the guide that connects all X_y to y, then $\Pi = \{\pi_y\}_{y \in Y}$ is a one color guidance system for all nodes in $G \cap X$. ■

The guides of Π we call *starting guides* as usual.

We now need to combine Π and $\Pi_{\mathcal{G}}$. If we simply combine $\Pi_{\mathcal{G}}$ and Π, we might end up with a starting guide going through and external edge of G that is already traversed by two transit guides. To avoid this problem, we need to do an optimisation relying on a simple observation formulated in the claim below.

A descendant guide π is called *live* in a subfactor G if π passes through G, and its target is *not* in a child subfactor of G (i.e., the target is in a proper descendant of some child of G). The idea is that the target of π satisfies the assumption $m_2 \geq 2$ from the 'point of view' of nodes in G. Note that guides live in G may be either transit or starting in G.

Claim 5. *Suppose that two consistent descendant guides* π *and* π' *are live in a subfactor* G *and exit* G *through the same edge. Suppose also that at least one of them is starting in* G*. Then all source nodes of one of* π*,* π' *can be moved to the other.*

Proof: Let (y_1, y_2) be the witness pair corresponding to π and let (y_1', y_2') be the witness pair corresponding to π' – by consistency, not only the second witnesses y_2 and y_2' are determined by π and π', but the whole witness pairs. Note that the source nodes of a descendant guide in G are all situated on one path from the root of G to one of the leaves. If both π and π' are starting in G, assume wlog that π has a source node that is an ancestor of all source nodes of π'; otherwise one of the guides is starting in G, wlog assume it is π'. As only the case of $m_1, m_2 \geq 2$ is considered, and the values $s_1, \ldots, s_6$ are fixed, due to equation (7)(i) the pair (y_1, y_2) is a witness for all source nodes of π' as well. Thus, these nodes may be guided to y_2 instead of y_2'. □ ■

We use the term *live transit guide* for the transit guides that are live in G, and *dead transit guide* for the other transit guides (those that have their target in a child subfactor of G).

Consider an edge e that connects G with a child subfactor H. Suppose first that e is traversed by a starting guide and a live transit guide. Using the claim we merge the starting guide with the transit one. Therefore, we end up satisfying the invariant property: e is passed by at most one live guide, and possibly by one dead transit guide.

□

IX. Applications

This section contains the proofs of Theorem 4 and 5.

Proof of Theorem 4

We now prove Theorem 4, which says that emptiness is decidable for class automata on bipartite data trees. For simplicity, we consider a more restricted definition: we allow only data words, each class has exactly two elements: one in the first half, and the other in the second half. Stated differently, a *bipartite data word* is defined as a data word $(w, \sim)$ with an even number $2n$ of positions, where every class has two positions: one in $\{1, \ldots, n\}$ and the other in $\{n+1, \ldots, 2n\}$. A similar proof works for the general definition.

Consider a class automaton $\mathcal{A}$ where the transducer is $f : \Sigma^* \to \Gamma^*$ and the class condition is a language L over alphabet $\Gamma \times \{0, 1\}$. In a bipartite data word each class has

size exactly two, so we can model the class condition as a binary query φ with input alphabet Γ, which selects a pair $x < y$ in a word w if and only if $w \otimes \{x, y\}$ belongs to L.

We first state a simple lemma. Suppose w is a word and X a set of positions (usually a prefix or suffix). Then $w|X$ is the substring of w which contains only the labels from the positions in X.

Lemma 11. *Let φ be a regular binary query with input alphabet Σ. There is a morphism $\alpha : (\Sigma \times \{0,1\})^* \to S$ and a subset $F \subseteq S^2$ such that for any positions $x \leq z < y$ in a word $w \in \Sigma^*$, $\varphi(w)$ contains (x, y) if and only if*

$$(\alpha(w \otimes \{x\}|\{1, \ldots, z\}), \alpha(w \otimes \{y\}|\{z{+}1, \ldots, |w|\})) \in F.$$

Coming back to the proof of Theorem 4, we will prove a stronger result, namely that there exists an automaton with Presburger conditions that accepts exactly the data erasure of $L(\mathcal{A})$, i.e., the set $K \subseteq \Sigma^*$ of words w such that for some $\sim$, the pair $(w, \sim)$ is a bipartite data word accepted by $\mathcal{A}$. The result follows, since emptiness is decidable for automata with Presburger conditions.

The language K is the inverse image, under the transducer f, of the set $M \subseteq \Gamma^*$ of words that satisfy

> There exists an equivalence relation $\sim$ such that $(w, \sim)$ is a bipartite data word where each class $x < y$ satisfies $(x, y) \in \varphi(w)$.

Since automata with Presburger conditions are closed under images of transducers, it suffices to show that M is recognized by a an automaton with Presburger conditions.

Apply Lemma 11, obtaining a morphism α and a set F. We claim that membership of a word w of length $2n$ in the language M depends only on the following $2|S|$ numbers, which we call the footprint of w.

$$\begin{aligned} i_s &= |\{x : x \leq n \text{ and } \alpha(w \otimes \{x\}|\{1, \ldots, n\}) = s\}| \\ j_s &= |\{y : y > n \text{ and } \alpha(w \otimes \{y\}|\{n+1, \ldots, 2n\}) = s\}|. \end{aligned}$$

More specifically, w belongs to M if and only if the footprint satisfies the following semilinear property $Q \subseteq \mathbb{N}^{2|S|}$: there exist numbers $\{k_{s,t}\}_{(s,t)\in F}$ such that for each $s \in S$

$$\sum_{t:(s,t)\in F} k_{s,t} = i_s \qquad \sum_{t:(t,s)\in F} k_{t,s} = j_s.$$

This completes the proof of Theorem 4, since an automaton with Presburger conditions can compute the footprint and test if it satisfies Q.

Proof of Theorem 5

Towards a contradiction, suppose that ψ is recognized by any class automaton in the generalized version for 2 data values. The document that will confuse ϕ will be a word. Consider the document (over a one letter alphabet) with positions $1, \ldots, 2n$, where the data values are defined

$$\begin{aligned} d_1(i) = d_1(n+i) &= i \quad \text{for } i \in \{1, \ldots, n\} \\ d_2(i) = d_2(n+i) &= -i \quad \text{for } i \in \{1, \ldots, n\} \end{aligned}$$

Since the above document satisfies ψ, it should also be accepted by the automaton. Let the work alphabet of the automaton be Γ, and let $a_1 \cdots a_{2n} \in \Gamma^*$ be the word produced by the automaton in the accepting run. For a data value d, we use the term *class string* of d for the word

$$a_1 \cdots a_{2n} \otimes d_1^{-1}(d) \otimes d_2^{-1}(d).$$

By definition of the way class automata accept documents, each class string should belong to the class condition. Consider a number $i \in \{1, \ldots, n\}$. The class string of i is

$$u_i = \quad a_1 \cdots a_{2n} \otimes \{i, n+i\} \otimes \emptyset$$

Suppose that

$$\alpha : (\Gamma \times \{0,1\} \times \{0,1\})^* \to S$$

is a morphism recognizing the class condition. If n is greater than $|S|^2$, then we can find two data values $i < j \in \{1, \ldots, n\}$ such that

$$\begin{aligned} \alpha(u_i|\{1, \ldots, n\}) &= \alpha(u_j|\{1, \ldots, n\}) \\ \alpha(u_i|\{n+1, \ldots, 2n\}) &= \alpha(u_j|\{n+1, \ldots, 2n\}). \end{aligned}$$

Consider a new document obtained from the previous one by swapping the first, but not second, data value in the positions i and j. This new document violates the property ψ. To get the contradiction, we will construct an accepting run of the class automaton for this new document. The output of the transducer is the same $a_1 \cdots a_{2n}$. The class strings are the same for data values other than i and j, so they are also accepted. For the class strings of i and j, the images under α are the same by assumption on i and j, and hence they are also accepted.

X. The classification theorem

We prove now Theorem 6. Most of the proof is standard, and uses known results from [5] and [3]. There are two original contributions: a) we show that data automata are the same as class automata with a local class condition; b) we show that alternating one register automata are captured by class automata with tail class conditions.

A. $\mathscr{A}_0$ versus $\mathscr{C}_0$

We begin by considering the general case, which involves the powerful but undecidable automata: counter automata and class automata.

Proposition 2. *The following language classes are equal.*

1) *Languages of words without data recognized by counter automata.*
2) *Projections of languages of data words recognized by class automata.*

sketch: We begin with the inclusion of the first class in the second class. Consider a counter automaton with the

set of transitions Δ. Let L be a language over alphabet Δ that contains all data words $(w, \sim)$ such that the following condition holds.

> Let x be a position with label $c := c + 1$. Then there is exactly one other position y in the same class, this position is after x, and has label $c := c - 1$. Furthermore, no position between x and y is labeled by the test $c = 0$.

The language L is recognized by a class automaton. It is not difficult to see that after erasing the data values from L, we get exactly the accepting runs of the counter automaton. Consequently, the language accepted by the counter automaton is obtained from L by erasing data values and projecting each transition onto the letter it reads.

The opposite inclusion, of the second class in the first class, is proved the same way as when translating a data automaton into a counter automaton without zero tests in [3]. The zero tests are used to capture the additional power of class automata. ■

B. $\mathscr{A}_1$ versus $\mathscr{C}_1$

We move to the first of the three decidable restrictions: counter automata without zero tests, and class automata with local class conditions. Case $i = 1$ of Theorem 6 follows now from the two results below, which use data automata [3]. Before we define data automata, we state the two results.

Proposition 3. *The following language classes are equal.*

1) *Languages of words without data recognized by counter automata without zero tests.*
2) *Projections of languages of data words recognized by data automata.*

Proposition 4. *Data automata and class automata with local class conditions recognize the same languages of data words.*

Proposition 3 was shown in [3]. We now define a data automaton and prove Proposition 4.

A *data automaton* is given by a transducer and a class condition. The transducer is the same as in a class automaton, it is a letter-to-letter transducer $f : \Sigma^* \to \Gamma^*$ from the input alphabet Σ to the work alphabet Γ. The difference is in the class condition: in a data automaton the class condition is a language $L \subseteq \Gamma^*$ over the work alphabet Γ, and not over $\Gamma \times \{0, 1\}$. A data word $(w, \sim)$ is accepted if there is an output $v \in f(w)$ of the transducer such that for every class X, the string $v|X$ belongs to L.

Clearly a data automaton is a special case of a class automaton with a local class condition. The interest of Proposition 4 is that each automaton with a local class condition is equivalent to a data automaton. We show this below.

Consider a class automaton, with transducer f and a local class condition L. Let X be a set of positions. The X-successor of x is the first position after x in X. This position may be undefined, written as $\perp$. For $n \in \mathbb{N}$, the n-profile of a position x with respect to a set X is a symbol in

$$\Pi_n = \{\perp, +1, \ldots, +n, \mod 1, \ldots, \mod n\}$$

defined as follows. If x has no X-successor, the symbol is $\perp$. Otherwise, let d be the distance between x and its X-successor; if d is at most n, then the symbol is $+d$; otherwise the symbol is $\mod d'$, where $d' \in \{1, \ldots, n\}$ and $d' \equiv d \mod n$.

The n-profile of data word $(w, \sim)$ is defined to be the function that maps each position of w to its n-profile with respect to its own class. The n-profile can be treated as a word $v \in \Pi_n^*$ of the same length as w.

Proposition 4 follows from the following two lemmas, and closure of data automata under relabelings and intersections. The second lemma follows from [12].

Lemma 12. *Let $L \subseteq (\Gamma \times \{0, 1\})^*$ be a local language. There is some $n \in \mathbb{N}$ and a regular language $K \subseteq (\Gamma \times \Pi_n)^*$ such that for any data word $(w, \sim)$ over Γ:*

$$w \otimes X \in L \quad \textit{iff} \quad (w \otimes v)|X \in K, \quad \textit{for any class } X.$$

sketch: Let $\mathcal{A}$ be a deterministic automaton recognizing L. Since L is local, for any two $a, b \in \Gamma$, the transitions of $\mathcal{A}$ over $(a, 0)$ and $(b, 0)$ may be assumed to be the same. Intuitively, only the number of consecutive positions outside of X matters. Furthermore, as $\mathcal{A}$ is finite, this number can only be measured up to some finite threshold, and then modulo some number. This information is stored in the profile. ■

Lemma 13. *For any $n \in \mathbb{N}$ there is a data automaton, with input alphabet $\Sigma \times \Pi_n$, which accepts a data word $(w \otimes v, \sim)$ if and only if v is the n-profile of $(w, \sim)$.*

$\mathscr{A}_2$ versus $\mathscr{C}_2$.: The relationship between Presburger automata and commutatively local class automata, analogous to Propositions 2 and 3, follows from results of [4] specialized to data words.

C. $\mathscr{A}_3$ versus $\mathscr{C}_3$

We now move to the last of the three decidable restrictions: gainy counter automata, and class automata with tail class conditions. Case $i = 3$ of Theorem 6 will be a consequence of Propositions 5 and 6 below. These propositions talk about alternating one register automata, which we define and discuss in more detail shortly.

Proposition 5 ([5])**.** *The following language classes are equal.*

1) *Languages of words without data recognized by gainy counter automata.*
2) *Projections of languages of data words recognized by alternating one register automata.*

Suppose $\pi : \Sigma \to \Gamma$. If K is a set of data trees over Σ, we write $\pi(K)$ for the set of data trees over Γ obtained from K by keeping the data and replacing the labels using π. Any such $\pi(K)$ we call a relabelling of K. Note that unlike with projections, we do not erase any positions.

Proposition 6. *The following languages classes are equal:*

- *Languages of data trees recognized by class automata with a tail class condition.*
- *Relabelings of languages of data trees recognized by alternating one register automata.*

The two propositions give case $i = 3$ of Theorem 6, since the composition of a relabeling and a projection is a projection.

Now our main goal is to prove Proposition 6. We will prove this proposition for the more general case of data trees. (Under the natural generalization of tail class conditions for trees, where instead of a suffix we talk about a subtree.) We will talk about binary data trees, although the same proofs would work for unranked data trees, using the first child / next sibling encoding.

We begin by defining an alternating one register automaton on data trees. When talking about register automata, it is convenient to represent a data tree in a different way. Instead of the equivalence relation $\sim$ in $(t, \sim)$ in a data tree over Σ, we use a labeling μ of the nodes of t with data values from some fixed domain of data values, call it D. We present this as single tree $t \otimes \mu$ over the alphabet $\Sigma \times D$. The presentation using μ is more convenient for the automaton with a register, since it makes clear what is loaded into the register: an element of D. A data tree in the representation $t \otimes \mu$ corresponds to exactly one data tree in the representation $(t, \sim)$; but a data tree in the representation $(t, \sim)$ corresponds to many data trees in the representation $t \otimes \mu$.

Alternating one register automata

The ingredients of an alternating one register automaton are a state space Q, which is partitioned into *states owned by* $\exists$ and *states owned by* $\forall$; an initial state q_I; an input alphabet and a set of transitions. Each transition is a triple (q, o, p), where q, p are states, and o is one of the operations: "test if the current label is a", "test if current class is in the register", "load class into the register", "move to the left child", or "move to the right child". A configuration of the automaton in a data tree is a triple consisting of a state, a tree node and a data value, called the *content of the register*. The automaton accepts a data tree if player $\exists$ has a strategy to win certain game, played by two players $\exists$ and $\forall$, to be described below.

The game begins in the *initial configuration*, consisting of the initial state, the root node, and the data value of the root node. In a configuration (q, x, d), the player who owns state q chooses the next configuration (p, y, e) in such a way that for some transition (q, o, p) the two configurations are related by the $\to_o$ relation defined in Table II. Player $\exists$ wins the game if a configuration owned by $\forall$ is reached from which no move can be done, otherwise $\exists$ loses (this includes infinite plays, which can happen if the automaton stays in a node forever). Note that due to alternation we don't need to distinguish accepting states. Likewise, we don't need testing whether the current class *is not* in the register, as this can be simulated by the opponent's test whether the current class *is* in the register.

Tail class automata

The *class tail* of a node x, belonging to a class X, in a data tree $(t, \sim)$ is the subtree of $t \otimes X$ rooted in x. A *tail class automaton* with input alphabet Σ and work alphabet Γ is given by a letter-to-letter transducer f with input alphabet Σ and output alphabet Γ, and a regular language over $\Gamma \times \{0, 1\}$, called the *tail condition*. So far, the definition is the same as for normal class automata. The difference is in the semantics: a tail class automaton accepts a data tree $(t, \sim)$ if there is an output $s \in f(t)$ such that all class tails in $(s, \sim)$ are accepted by the tail condition. (In class automata, for each class X, $s \otimes X$ needs to be accepted.)

It is not difficult to see that tail class automata are equivalent to class automata with a tail class condition. Hence it suffices to prove the version of Proposition 6 where "tail automata" is substituted for "class automata with a tail class condition".

Proof of Proposition 6

The inclusion of the first class in the second class in the statement of the Proposition 6 is straightforward. Suppose that we have a tail automaton recognizing a set L of data trees, given by a transducer f from Σ to Γ, and a tail condition M. Consider the set K of data trees

$$t \otimes s \otimes \mu \qquad t \text{ over } \Sigma,\ s \text{ over } \Gamma$$

where: a) $s \in f(t)$; and b) for every node x, the class tail of x belongs to K. Clearly, the language L recognized by the tail automaton is obtained from K by the relabeling which erases Γ. On the other hand, K is recognized by an alternating one register automaton. Condition a) is regular, and condition b) is tested by universally branching over all nodes x, putting node x into the register, and then inspecting the subtree.

The inclusion of the second class in the first class in the statement of Proposition 6 is more difficult. This inclusion follows from the following theorem, as languages recognized by the tail class automata are closed under relabeling.

Theorem 8. *For every alternating one register automaton one can compute an equivalent tail class automaton.*

The rest of this section is devoted to showing the above theorem. Fix an alternating one register automaton $\mathcal{A}$. We will show that there is a tail class automaton equivalent to $\mathcal{A}$.

operation o	condition on $(q,x,d) \to_o (p,y,e)$
test if current label is a	$x=y$, $d=e$, $t(x)=a$
test if current class is in the register	$x=y$, $d=e=\mu(x)$
load current class into the register	$x=y$, $e=\mu(x)$
move to the left/right child	y is the left/right child of x, $d=e$

Table II
DEFINITION OF $\to_o$ GIVEN A DATA TREE $t \otimes \mu$.

Recall the acceptance game described in the definition of alternating one register automata, which we denote by G. This game depends on the automaton $\mathcal{A}$ and on the data tree $t \otimes \mu$, which we hope will be clear from the context. Below, we will also consider games with initial configurations different from the one described above.

A configuration of $\mathcal{A}$ in a data tree is called *local* if its register value is the class of the current position. After performing a "load..." transition, the resulting configuration is always local. Suppose that Γ is a set of local[1] configurations in a data tree. We define the game G_Γ in the same way as the game G in the definition of alternating one register automata, with the difference that when a local configuration is seen, different from the initial configuration of the game, the play is ended. Player $\exists$ wins if the configuration is in Γ and player $\forall$ wins otherwise.

Lemma 14. *In a data tree, the following conditions are equivalent for any set of local configurations Δ.*

1) *Player $\exists$ wins the game G from a local configuration (q,x,d).*
2) *For some set of local configurations Γ containing (q,x,d), player $\exists$ wins the game G_Γ from every configuration in Γ.*

Proof: For the bottom up implication, we create a winning strategy in G by composing the strategies winning in the game G_Γ from the positions in Γ. For the top down implication, we choose Γ to be the set of all local configurations where $\exists$ can win the game. The second implication works because G and G_Γ are positionally determined. ■

In Lemma 15 below, we will show that a tail class automaton can test if a set of local configurations is winning for player $\exists$, in the sense of the above lemma. First, we say how a set of local configurations is coded in the input of a tail class automaton. Suppose that Γ is a set of local configurations in a data tree $t \otimes \mu$. We write $t \otimes \Gamma$ for the tree with the same nodes as t, but where the label of each node x is enriched by the set of states q such that Γ contains the unique local configuration with state q and node x. (The alphabet of $t \otimes \Gamma$ is $\Sigma \times P(Q)$ instead of Σ.)

Lemma 15. *There is a tail class automaton $\mathcal{E}$ such that for any data tree $t \otimes \mu$ and set of local configurations Γ, the automaton $\mathcal{E}$ accepts $t \otimes \Gamma \otimes \mu$ if and only if player $\exists$ wins the game G_Γ from every configuration in Γ.*

Proof: Fix $t \otimes \mu$ and a set Γ of local configuration. What do we need to know about $t \otimes \mu$ to see if player $\exists$ wins G_Γ from the unique local configuration in state q and node x?

The crucial observation is that as far as playing the game G_Γ is concerned, we only need to know which positions are in the class of x, call it X, and we do not care about the precise data values of other positions. This is because the game G_Γ is terminated when the register value is changed. Using this observation, we can write an alternating automaton $\mathcal{B}$ (on trees without data) such that for any data tree $t \otimes \mu$, any set Γ of local configurations, and any class X containing a node $x \in X$, $\mathcal{B}$ accepts the subtree of $t \otimes \Gamma \otimes X$ rooted in x if and only if player $\exists$ wins the game G_Γ from all the local configurations in node x that are in Γ. Using $\mathcal{B}$ for the tail condition, together with the identity transducer, we get a tail class automaton as required. ■

Lemmas 14 and 15 give Theorem 8. The tail class automaton that is equivalent to $\mathcal{A}$ works as follows: it uses the transducer to guess a set of local configurations Γ that contains the initial configuration (the initial configuration is necessarily local). Then it uses the automaton $\mathcal{E}$ from Lemma 15 to test if player $\exists$ can win the game G_Γ from every configuration in Γ. By Lemma 14, the latter condition is equivalent to player $\exists$ winning the game G from the initial configuration, which is the same as $\mathcal{A}$ accepting the tree.

XI. DECIDING EMPTINESS OF TAIL CLASS AUTOMATA

We now prove that emptiness is decidable for tail class automata.

Theorem 9. *Emptiness is decidable for tail class automata.*

Note that this theorem follows from the previous results. To decide emptiness of the language L of a tail class automaton, do the following steps. First, apply Proposition 6 to compute a relabeling π and a language K recognized by an alternating one register automaton, such that $L = \pi(K)$. Next, apply Proposition 5 (more precisely, its tree variant from [6]) to compute a gainy counter automaton (for trees) whose emptiness is equivalent to emptiness of K. Finally, use results on gainy counter automata for trees from [6] to decide emptiness of the gainy counter automaton for trees.

[1] the definition of G_Γ works for any kind of configurations but, we only use local ones.

We give the proof below aiming at a self-contained presentation, as we do not to use any external results to show that emptiness is decidable for alternating one register automata, or for tail class automata. We would like to underline, however, that none of the ideas here are substantially different from the ones in the original papers on alternating one register automata.

Fot the rest of this section fix a tail class automaton $\mathcal{T}$. Let f be the transducer, and let $\mathcal{A}$ be a tree automaton recognizing the tail condition. Let P and Q be the state spaces of f and $\mathcal{A}$, respectively.

A. Profiles

A *profile* is a pair consisting of a *transducer state* $p \in P$ and an *obligation set* S, which is a finite subset of $Q \times D$. The rough idea is that a profile represents run of $\mathcal{T}$ in the moment that it enters a node x from its parent. The transducer state represents the state of the transducer in x, while each pair (q, d) in the obligation set represents a run of the automaton $\mathcal{A}$ that was started for a class tail of an ancestor of x with data value d. A more precise definition is presented below.

In the following we write $\mathcal{A}(q)$ for the automaton$\mathcal{A}$ with initial state changed to q, likewise $f(p)$ for the transducer. A profile (p, S) is called *satisfiable* if there exists a data tree $t \otimes \mu$ and

1) A tree s output by $f(p)$.
2) For each node x in s, a run ρ_x of $\mathcal{A}$ on the class tail of x in $s \otimes \mu$.
3) For each $(q, d) \in S$, a run $\rho_{(q,d)}$ of $\mathcal{A}(q)$ on $s \otimes \mu^{-1}(d)$.

Emptiness of $\mathcal{T}$ is equivalent to satisfiability of the *initial profile*, where the transducer state is the initial state of the transducer, and the obligation set is empty. The rest of Section XI is devoted to an algorithm that decides if a given profile is satisfiable.

B. Well structured transition systems

Consider an enumerable set, which we will call a *domain*. A *transition system* on the domain is a function f that maps each element x of the domain to a nonempty family of subsets of the domain. A *derivation* is a finite tree whose nodes are labeled by elements of the domain, such that for each node with label x, the labels in the children of x form a set from $f(x)$. We write f^* for the set of elements in the domain that have (i.e. appear in the root of) a derivation.

A transition system f is called *computable* if f maps each element to a *finite* family of *finite* subsets of the domain, and f is a computable function.

A binary relation R over the domain is called a *downward simulation* if for any $(y, x) \in R$ and any $X \in f(x)$, there is some $Y \in f(y)$ such that each element of Y is related by R to some element of X: for each $y' \in Y$ there is $x' \in X$ with $(y', x') \in R$.

Lemma 16. *If a downward simulation R contains a pair (y, x) and $x \in f^*$ then $y \in f^*$ and $|y| \leq |x|$.*

A transition system is *monotone* if its domain is equipped with a partial order $\leq$ that is a downward simulation. By Lemma 16 in a monotone transition system the set f^* is downward-closed, and thus its complement $\neg f^*$ is upward closed.

Theorem 10. *Consider an enumerable domain with a computable, well founded order without infinite antichains. For any computable and monotone transition system f, membership in f^* is decidable.*

Proof: For an element x of the domain we run two algorithms in parallel. The first algorithm enumerates all derivations and terminates if it finds one with x in the root. Thus the algorithm terminates if and only if x belongs to f^*.

The second algorithm enumerates all finite antichains in the domain and terminates if it finds an antichain Z that such that x belongs to the upward closure of Z, call it $Z^\uparrow$, and Z satisfies the following property (*): for any element $z \in Z$, every set in $f(z)$ has nonempty intersection with $Z^\uparrow$. We now show that this algorithm terminates if and only if x does not belong to f^*.

Suppose that x does not belong to f^*. We claim that the algorithm succeeds; one possible set Z produced by the algorithm is all minimal elements in the complement of f^*. Since the order has no infinite antichains, Z is finite. By well foundedness, the complement of f^* equals $Z^\uparrow$. Thus Z satisfies property (*) and $x \in Z^\uparrow$.

Suppose that x belongs to f^*. We claim that the upward closure $Z^\uparrow$ of any antichain Z satisfying (*) contains no elements of f^*, and hence the algorithm must fail. For the proof we will use a bit of duality of induction and coinduction.

The set f^* is the smallest set $\mathcal{X}$ with the property that whenever $f(x)$ contains a set X with $X \subseteq \mathcal{X}$ then $x \in \mathcal{X}$. Thus f^* is the smallest (pre-)fixed point of the mapping:

$$\mathcal{F} : \mathcal{X} \mapsto \{x : \text{ some set in } f(x) \text{ is included in } \mathcal{X}\},$$

ie., the smallest $\mathcal{X}$ with $\mathcal{F}(\mathcal{X}) \subseteq \mathcal{X}$. By duality, the complement of f^* is the greatest (post-)fixed point of the mapping $\mathcal{X} \mapsto \neg\mathcal{F}(\neg\mathcal{X})$; i.e., the greatest $\mathcal{X}$ with $\mathcal{X} \subseteq \neg\mathcal{F}(\neg\mathcal{X})$; i.e., the greatest $\mathcal{X}$ such that whenever $x \in \mathcal{X}$ then each set in $f(x)$ has nonempty intersection with $\mathcal{X}$.

Now we are ready to argue that if Z satisfies (*) then $Z^\uparrow$ is included in $\neg f^*$. Indeed, by monotonicity it follows that $Z^\uparrow$ satisfies (*): for every $z \in Z^\uparrow$, every set in $f(z)$ has nonempty intersection with $Z^\uparrow$. Thus $Z^\uparrow$ is a post-fixed point, $Z^\uparrow \subseteq \neg\mathcal{F}(\neg Z^\uparrow)$, and hence necessarily is included in the greatest one. ∎

C. Profiles form a transition system

In this and in the next section we apply the machinery of Theorem 10 to decide which profiles are satisfiable, completing the decision procedure for emptiness of tail class automata. Our aim is first to define an ordered domain and a transition system f, and then its quotient $\widehat{f}$ that satisfies the assumptions of Theorem 10, and such that satisfiable profiles can be computed on the basis of testing membership in $\widehat{f}^*$.

The domain of f will contain all profiles; f itself we define below. The definition of f is designed so that a data tree that makes a profile satisfiable is essentially a derivation of this profile, and vice versa (cf. Lemma 17 below).

A profile (p, S) we call *accepting* if p as well as all states appearing in S are accepting. If a profile $x = (p, S)$ is accepting then let $f(x)$ contain only $\emptyset$. Otherwise, $f(x)$ will contain all the sets $\{(p_0, S_0), (p_1, S_1)\}$, of size at most two each, that arise as an outcome of the following nondeterministic 'procedure':

- Choose an input letter a of f, a transition $p, a \to b, p_0, p_1$ thereof, and a data value $\bar{d}$.
- For each $(q, d) \in S \cup \{(q_{\text{init}}, \bar{d})\}$, where q_{init} is the initial state of $\mathcal{A}$, choose a transition of $\mathcal{A}$

$$\begin{array}{lcll} q, (b, 1) & \to & q_0, q_1 & \text{if } d = \bar{d} \\ q, (b, 0) & \to & q_0, q_1 & \text{if } d \neq \bar{d} \end{array}$$

  and add (q_0, d) to S_0 and (q_1, d) to S_1.

Lemma 17. *Satisfiable profiles are exactly f^*.*

The profiles are ordered coordinatewise, with the first coordinate (states of the transducer) ordered discretely (all states are incomparable), and the second coordinate (obligations) ordered by inclusion.

Lemma 18. *The transition system f is monotonne.*

D. Abstractions of profiles form a well-structured transition system

Now we define the well structured transition system $\widehat{f}$. Elements of the domain will not be profiles, but isomorphism classes of profiles, in the following sense. Two profiles are called *isomorphic* if they have the same transducer state and there is a bijection on data values that maps one obligation set to the other. It is not difficult to see that the isomorphism class of (p, S) is identified by its *abstraction* $(p, [S])$, where the vector

$$[S] \in \mathbb{N}^{P(Q) \setminus \{\emptyset\}} ,$$

called the *abstraction of* S, stores on coordinate $Q' \subseteq Q$ the number of data values in the set

$$\{d \in D : Q' = \{q : (q, d) \in S\}\}.$$

The domain of $\widehat{f}$ will consist of abstractions of profiles; in particular, the domain is enumerable. Since isomorphic profiles are equivalently satisfiable, this abstraction is not a problem: it makes sense to talk about satisfiable abstractions.

The transition function $\widehat{f}$ is defined by projecting the transition function f: if $w = [S]$ is an abstraction of the obligation set S, then $\widehat{f}(p, w)$ contains precisely those sets that are obtained by abstraction of some set X from $f(p, S)$; by abstraction of X we mean here the set of profile abstractions obtained by replacing each profile (p', S') appearing in X with its abstraction $(p', [S'])$.

The following lemma together with Lemma 17 reduces emptiness of tail class automata to testing $\widehat{f}^*$.

Lemma 19. *A profile is in f^* if and only if its abstraction is in $\widehat{f}^*$.*

Proof: For both directions we may use Lemma 16 applied to the disjoint union of f and $\widehat{f}$; as the downward simulation we will use either the relation R containing all pairs (a profile, its abstraction), or the inverse relation R^{-1}, respectively. It remains to show that both R and R^{-1} are downward simulations.

Observe that the isomorphism of profiles satisfies a stronger property, which we call a *2-level bisimulation*: whenever x and y are isomorphic and $X \in f(x)$, there is $Y \in f(y)$ such that X and Y witness the following *1-level bisimulation* condition: for any $x' \in X$ there is $y' \in Y$ isomorphic to x'; and vice versa, for any $y' \in Y$ there is $x' \in X$ isomorphic to y'.

In consequence, the relation R is a 2-level bisimulation too. Thus both R and R^{-1} are downward simulations indeed. ∎

The transition function $\widehat{f}$ maps always an abstraction x to a finite family of finite sets of abstractions, and $\widehat{f}(x)$ is computable, for a given x, by simulating the nondeterministic procedure given in Section XI-C. Thus we have:

Lemma 20. *The transition system $\widehat{f}$ is computable.*

The order on abstractions of obligation sets is defined by projecting the inclusion order on obligation sets: $w \leq w'$ if $w = [S]$, $w' = [S']$ for some obligation sets $S \subseteq S'$. Accordingly we define the order on abstractions of profiles: $(p, w) \leq (p', w')$ if $p = p'$ and $w \leq w'$. Directly by Lemma 18 and by the bisimulation property of the profile isomorphism we get:

Lemma 21. *The transition system $\widehat{f}$ is monotone.*

We complete the check-list of assumptions of Theorem 10 by showing the following:

Lemma 22. *The order $\leq$ is computable, well founded and has no infinite antichains.*

Proof: Since there are finitely many transducer states, it suffices that these properties hold for the order on abstractions of obligation sets.

Recall that abstractions of obligation sets are vectors of numbers indexed by nonempty subsets of Q. Note that the order on abstractions of obligation sets we use here, is not the same as the classical coordinatewise order on vectors of numbers, call it $\preceq$.

To show computability, we give the following alternative definition of $\leq$. For two abstractions v, w of obligation sets, we have $v \leq w$ if and only if v can be reached from w by a finite number of steps of the form: choose a coordinate $R \subseteq Q$ and an element $q \in R$, decrement the vector on coordinate R; and then increment the vector on coordinate $R \setminus \{q\}$ if $R \setminus \{q\}$ is nonempty.

This alternate definition also shows that the order is well founded: it is impossible to do an infinite sequence of such steps.

Finally, we want to show that $\leq$ has no infinite antichains. It is not hard to see that $v \preceq w$ implies $v \leq w$. It is well known that $\preceq$ has no infinite antichains, and hence the same must hold for $\leq$. ■