Two-Variable Logic on Data Trees and XML Reasoning

Mikołaj Bojańczyk
Warsaw University
and
Anca Muscholl
LABRI & Université Bordeaux 1
and
Thomas Schwentick
Universität Dortmund
and
Luc Segoufin
INRIA & LSV

Motivated by reasoning tasks for XML languages, the satisfiability problem of logics on *data trees* is investigated. The nodes of a data tree have a *label* from a finite set and a *data value* from a possibly infinite set. It is shown that satisfiability for two-variable first-order logic is decidable if the tree structure can be accessed only through the *child* and the *next sibling* predicates and the access to data values is restricted to equality tests. From this main result, decidability of satisfiability and containment for a data-aware fragment of XPath and of the implication problem for unary key and inclusion constraints is concluded.

Categories and Subject Descriptors: F.4.1 [**Mathematical logic and formal languages**]: Mathematical logic; H.2.3 [**Database management**]: Languages—*Query languages*; H.2.1 [**Database Management**]: Data models; F.4.3 [**Mathematical Logic and Formal Languages**]: Decision problems; I.7.2 [**Document Preparation**]: Markup languages

General Terms: Algorithms, Design, Languages, Theory

Additional Key Words and Phrases: Integrity Constraints, DTDs, XML, Consistency, Implication

1. INTRODUCTION

Most theoretical work on XML and its query languages models XML documents by labeled ordered unranked trees, where the labels are from a finite set. Attribute values are usually ignored. This has basically two reasons, which are not independent. First, the modeling allows to apply automata based techniques, as automata

Work supported by the French-German cooperation programme PROCOPE, the EU-TMR network GAMES, the Polish government grant no. N206 008 32/0810 and DFG grant SCHW 678/3-1.

operate on trees of this kind. Second, extending the model by attribute values (data values) quickly leads to languages with undecidable static analysis (see, for instance [Alon et al. 2003; Neven and Schwentick 2003; Benedikt et al. 2005; Geerts and Fan 2005]).

Nevertheless, there are examples of decidable static reasoning tasks involving attribute values [Arenas et al. 2005; Buneman et al. 2003]. The motivation for our work was to find a logical approach for such tasks.

It is immediately clear that full first-order logic is far too powerful for this purpose. Satisfiability for first-order logic with a predicate for data values equality is undecidable already on strings [Bojańczyk et al. 2006]. There are several possible candidates for more appropriate logics, including temporal logics or fragments of first-order logic. In this work, we concentrate on a (classical) fragment of first-order logic, two-variable logic. There are several good reasons to consider this fragment. It is known that on many kinds of structures this fragment is decidable [Grädel and Otto 1999]. On ordered, unranked trees, it corresponds in a natural way to the navigational behavior of XPath[1], and it can express many interesting integrity constraints.

Before we describe the technical contributions of the paper, we first discuss the connections with XML processing in more detail.

Core-XPath, the fragment of XPath capturing its navigational behavior introduced by Gottlob et al. [Gottlob et al. 2002], is by now well understood. In particular, it corresponds to $\mathrm{FO}^2(<,+1)$ on unranked ordered trees [Marx 2005]. Here, $\mathrm{FO}^2(<,+1)$ is the two-variable fragment of first-order logic that uses the order $<$ and successor $+1$ relations, along with the labels of the nodes. The labels are encoded by unary relations, one for each of the (finitely many) possible labels. The symbol "$<$" refers to two binary predicates: one for comparing the descendant/ancestor relationship of two nodes and one for the preceding/following relationship of two siblings. Similarly, "$+1$" also refers to two binary predicates: one for comparing the parent/child relationship of two nodes and one for comparing the next/previous sibling relationship of two siblings. Core-XPath is decidable even in the presence of DTDs and the complexity of many of its fragments has been studied in the literature. We refer to [Benedikt et al. 2005; Geerts and Fan 2005] and the references therein for a comprehensive survey.

In the presence of data values a simple extension of Core-XPath is to allow (in)equalities of the form $p/@A \ \mathtt{op} \ q/@B$, $\mathtt{op} \in \{=, \neq\}$, inside qualifiers, meaning that the value of the A attribute of some node accessible by a path matching p equals the B attribute value of some node accessible by a path matching q. We denote this fragment by Core-Data-XPath, but it has also been called FOXPath in [Benedikt and Koch 2009]. As shown in [Geerts and Fan 2005], Core-Data-XPath is undecidable and both [Benedikt et al. 2005; Geerts and Fan 2005] studied fragments of Core-Data-XPath. Given the correspondence between Core-XPath and $\mathrm{FO}^2(<,+1)$ it seems natural to investigate the logic $\mathrm{FO}^2(\sim,<,+1)$, the extension of $\mathrm{FO}^2(<,+1)$ with a binary predicate $\sim$ that checks data value equality of two nodes, as a possible logical foundation for Core-Data-XPath. It is easy to verify (see Section 6) that $\mathrm{FO}^2(\sim,<,+1)$ is contained in Core-Data-XPath, and that this inclusion is strict

[1]Here and in the rest of the article we refer to XPath 1.0.

(for instance, $\mathrm{FO}^2(\sim, <, +1)$ cannot express the test $\texttt{Self/@A} = \texttt{Self//b//c/@A}$). Therefore it is natural to wonder whether $\mathrm{FO}^2(\sim, <, +1)$ is decidable.

We do not solve the question whether $\mathrm{FO}^2(\sim, <, +1)$ is decidable here. Nevertheless, we show that the decidability of $\mathrm{FO}^2(\sim, <, +1)$ is a difficult problem as it would imply deciding multicounter automata on trees and the linear logic MELL, both of which are known as open issues in their respective fields (see [de Groote et al. 2004] and the references therein).

Our main result shows that the logic $\mathrm{FO}^2(\sim, +1)$ is decidable. An additional contribution of our paper is a unified framework for some decidability questions for XML that involve data values, and that were studied separately in the past. We give some examples of such applications next.

—Common reasoning tasks for XML are the consistency and the implication problems for integrity constraints. Given a finite set S of constraints and a further constraint φ, one asks whether each document satisfying all constraints in S also satisfies φ. This boils down to testing if there is a document that satisfies all constraints in S but not φ, a satisfiability question. The most common family of integrity constraints are key and inclusion constraints. Many of them involve only one attribute. It is easy to see that such constraints can be expressed in $\mathrm{FO}^2(\sim, +1)$. Our main result implies the decidability of the implication problem for such constraints. This was already known from [Buneman et al. 2003], which shows that the complexity of implication, without schemas, is polynomial.

—An advantage of the logical approach is that reasoning problems can be relativized to documents satisfying schemas. Schemas are usually captured by regular tree languages (where only the labels and not the data values are used). It is known that regular tree languages can be characterized by $\mathrm{EMSO}^2(+1)$ sentences, i.e., sentences where an $\mathrm{FO}^2(+1)$ formula is preceded by a block of existential quantifiers ranging over sets of nodes. When satisfiability is concerned, it is straightforward that decidability of $\mathrm{FO}^2(\sim, +1)$ implies decidability of $\mathrm{EMSO}^2(\sim, +1)$. Thus, by combining formulas in a suitable way, it follows easily that the implication problem for unary key and inclusion constraints (and thus also for foreign key constraints) is decidable also relative to a schema given by a regular tree language. This result was already known from [Fan and Libkin 2002], where it was actually shown that the satisfiability problem is complete for NPTime. Our more generic approach yields a 3NExpTime upper bound.

—As tree automata can be used to assign *types* to nodes of a document, integrity constraints can refer to such types. Therefore, as these types can also be expressed by $\mathrm{EMSO}^2(\sim, +1)$ formulas, the implication problem for more involved integrity constraints is still decidable.

—Another application of the logical result considers the containment problem for XPath *with attribute equalities*. We present a fragment of XPath which allows equalities and inequalities on attribute values for which the containment problem can be reduced to satisfiability of $\mathrm{FO}^2(\sim, +1)$. By combining techniques, we obtain decidability of the containment for this XPath fragment even relative to a schema consisting of a regular tree language and unary constraints.

Overview The paper consists of two main parts. In the first, we introduce the

logic $\mathrm{FO}^2(\sim, +1)$ and prove the main technical result: satisfiability is decidable for sentences of $\mathrm{FO}^2(\sim, +1)$ over data trees. We also discuss possible extensions of this logic and show some lower bounds. In Section 4 we give strong evidence that decidability of $\mathrm{FO}^2(\sim, <, +1)$ on unranked trees is a difficult problem by reducing the non-emptiness problem for vector addition tree automata to it. In the second part of the paper, we apply the main result to XML reasoning tasks. In Section 5, we show that satisfiability and implication for unary key and inclusion constraints are decidable. Section 6 establishes decidability of the containment problem for an XPath fragment with attribute equalities.

Related work Closely related to our work are the papers [Benedikt et al. 2005; Geerts and Fan 2005] and the references therein. Most of the fragments they consider are inside Core-XPath (i.e., without data values). However, several other fragments are in Core-Data-XPath. In [Benedikt et al. 2005] the decidable fragments of Core-Data-XPath that have data equality tests either don't have negation or cannot access arbitrarily deep nodes in the tree (in contrast, $\mathrm{FO}^2(\sim, +1)$ has both). They also don't have horizontal navigation and therefore miss an important aspect of XML navigation features. The paper [Geerts and Fan 2005] extends the results of [Benedikt et al. 2005] by including the horizontal axis, but the only decidable fragment presented in that paper does not have negation. Finally, the focus of [Benedikt et al. 2005; Geerts and Fan 2005] was to have the precise complexity for the decision procedure of the fragment considered, while the precise complexity of $\mathrm{FO}^2(\sim, +1)$ is still an open issue. An inspection of the current proof of Theorem 3.1 gives an upper bound of 3NExpTime, and the best lower bound we currently have is NExpTime-hardness.

Another logical approach was considered in [Jurdziński and Lazić 2007], which extends from data words to data trees the temporal logic approach of [Demri et al. 2005; Demri and Lazić 2006]. The main contribution of [Jurdziński and Lazić 2007] is an alternating automaton over data trees, which uses registers to inspect data values. The decidability results are for one-way alternating automata with one register, with a non primitive recursive lower bound on the complexity. Various fragments of XPath can be encoded into the alternating automata. In general, the set of properties that can be described in our approach, and the set of properties that can be described using the alternating automata of [Jurdziński and Lazić 2007], are incomparable. For instance, "every data value appears twice" can be expressed in our logic but not by the alternating automata, while the converse separation is witnessed by "every data value appears at most once on each path".

The logic considered in [Alon et al. 2003] in order to solve the type inference problem is incomparable to $\mathrm{FO}^2(\sim, +1)$. It uses patterns with variables for the data values together with equality and inequality constraints on the variables in order to extract the relevant pieces of data. It can use arbitrarily many variables in the patterns, something $\mathrm{FO}^2(\sim, +1)$ cannot do, but it can only inspect the tree up to a given constant depth.

As we have already mentioned, restricting FO to its two-variable fragment is a classical idea when looking for decidability [Grädel and Otto 1999]. Over graphs or over any relational structures, FO is undecidable, while its two-variable fragment is decidable [Mortimer 1975]. This does not imply anything on the decidability of

$\mathrm{FO}^2(\sim, +1)$, since the equivalence relation and the two tree successor relations cannot be axiomatized in FO^2. A recent paper [Kieroński and Otto 2005] generalized the result of [Mortimer 1975] in the presence of one or two equivalence relations. Again this does not apply to our context as we also have two successor relations. However [Kieroński and Otto 2005] also showed that the two-variable fragment of FO with *three* equivalence relations, without any other structure, is undecidable. This implies that $\mathrm{FO}^2(\sim_1, \sim_2, \sim_3, +1)$ is undecidable.

On the other hand, two equivalence relations plus the extension of the successor relation to $+1, +2$ is already undecidable on words (see long version of [Bojańczyk et al. 2006]). Therefore, manipulating more than two different attributes at the same time quickly leads to undecidability.

Note that this does not imply much for XPath, as already in the presence of two equivalence relations the logic $\mathrm{FO}^2(\sim_1, \sim_2, +1)$ seems to be no longer included in XPath. For instance, we conjecture that XPath cannot check that a node x has the following property: $\exists y\ (y \neq x \wedge x \sim_1 y \wedge x \sim_2 y)$ expressing that there exists another node with the same two attribute values as x.

Results on consistency of integrity constraints in the presence of DTDs were given in [Arenas et al. 2005]. All of the results were obtained for DTDs where the tag names and the types are coupled (all tag names have the same type), and the extension to decoupled DTDs (also known as *extended DTDs*) was left open. In particular, it was shown that it is decidable whether a set of unary keys and foreign keys is consistent with a DTD. The decidability of $\mathrm{FO}^2(\sim, +1)$ implies that it is decidable whether any set of integrity constraints definable in $\mathrm{FO}^2(\sim, +1)$ is consistent with an extended DTD. In particular, as (absolute) unary keys and foreign keys are definable in $\mathrm{FO}^2(\sim, +1)$ this extends one of the results of [Arenas et al. 2005].

Another related line of research is to consider logics and automata on words with data values. In [Bouyer et al. 2003; Kaminski and Francez 1994; Neven et al. 2004] automata and logics over words with data values were considered. The automata of [Kaminski and Francez 1994; Bouyer et al. 2003] had very limited expressive power, while the logics and automata of [Neven et al. 2004] are undecidable. In [Bojańczyk et al. 2006] it is shown that $\mathrm{FO}^2(\sim, <, +1)$ is decidable on words with data values.

2. NOTATIONS AND PRELIMINARIES

In this paper we consider unranked, ordered, labeled trees with data values. A **data tree t over Σ** has a set of **nodes**, where every node v has a **label** $v.l \in \Sigma$ and a **data value**[2] $v.\mathrm{data} \in \mathbb{N}$.

A data tree can be seen as a relational first-order structure. The universe of this structure is the set of nodes of the tree, moreover, the predicates are as follows:

—For each possible label $a \in \Sigma$, there is a unary predicate $a(x)$, which is true for all nodes that have the label a.

—The binary predicate $x \sim y$ holds for two nodes if they have the same data value.

[2]We could choose any other infinite set instead of $\mathbb{N}$, as formulas can compare values only with respect to equality.

—The binary predicate $E_{\rightarrow}(x, y)$ holds for two nodes if x and y have the same parent node and y is the immediate successor of x in the order of children of that node.

—The binary predicate $E_{\downarrow}(x, y)$ holds if y is a child of x.

—The binary predicates $E_{\Rightarrow}$ and $E_{\Downarrow}$ are the transitive closures of $E_{\rightarrow}$ and $E_{\downarrow}$, respectively.

We write $\mathrm{FO}^2(\sim, <, +1)$ for two-variable first-order logic with all these predicates and $\mathrm{FO}^2(\sim, +1)$ for the logic without $E_{\Rightarrow}$ and $E_{\Downarrow}$. By $\mathrm{FO}^2(<, +1)$ and $\mathrm{FO}^2(+1)$ we denote the respective logics without $\sim$. Whenever we want to stress that a formula does not have free variables we call it a **sentence**.

Abusing notation, we allow ourselves to use these predicates outside of logical formulas, e.g., we write $v \sim w$ for two nodes v, w that have the same data value. We also write $v \sim d$ if the data value of the node v is d.

A set of nodes of a data tree is called **connected** if it is connected in the graph induced by $E_{\rightarrow}$ and $E_{\downarrow}$.

For a data value d, the ***d*-class** of a data tree is the set of all nodes with data value d. A **class** is a d-class, for some d. A **zone** is a maximal connected set of nodes with the same data value. Two zones are **adjacent** if their union is connected. Zones are illustrated in Figure 1.

A **node profile** is a triple (l, p, r) of elements from $\{=, \neq, \perp\}$. The profile of a node v indicates whether v has the same data value as its left sibling, parent, and right sibling, respectively, where $\perp$ is used in the case that there is no respective node. We will often use formulas like $\mathrm{root}(x)$ and $\mathrm{lmchild}(x)$ expressing that a node has no parent (resp., no left sibling, i.e., it is a leftmost child). These formulas are straightforward Boolean combinations of profile atoms.

Let Pro denote the set of the 19 possible node profiles (if a node has no parent, then it has no siblings, so 8 profiles are not legal). For a data tree t over Σ, the **profiled tree** associated with t is the data tree over $\Sigma \times \mathrm{Pro}$ obtained by adding to each node of t its profile.

For a data tree t over Σ, the **data erasure** of t is the tree over Σ obtained from t by ignoring the data value v.data of each node. The data erasure is a finitely labeled tree.

3. DECIDABILITY OF $\mathbf{FO}^2(\sim, +1)$ ON TREES

In this section, we present the main result of the paper.

THEOREM 3.1. *It is decidable whether an* $\mathrm{FO}^2(\sim, +1)$ *sentence is satisfied in some finite unranked ordered data tree.*

We actually prove a slightly more general result.

The first generalization is that we consider an extended logic $\mathrm{EMSO}^2(\sim, +1)$. This logic consists of $\mathrm{FO}^2(\sim, +1)$ formulas that are preceded by a prefix of existential set quantifiers, i.e. sentences ψ of the form

$$\exists R_1 \ldots \exists R_n \varphi \ ,$$

where $R_1, \ldots, R_n$ are unary predicates and φ is a formula of $\mathrm{FO}^2(\sim, +1)$. We call the formula φ the *core*, and $\exists R_1 \ldots \exists R_n$ the *second-order prefix*. The formula φ

can be seen as a formula on data trees with an extended alphabet Σ', in which each symbol consists of a symbol from the original alphabet Σ and a bit vector indicating membership in the sets $R_1, \ldots, R_n$.

Clearly, ψ has a model if and only if the sentence φ has a model and therefore, the decidability of $\mathrm{EMSO}^2(\sim, +1)$ and $\mathrm{FO}^2(\sim, +1)$ are equivalent.

Apart from being more expressive, the extended logic is technically more convenient. The idea is that a lot of the complexity can be pushed from the core into the second-order prefix. From now on, we will be working with sentences of $\mathrm{EMSO}^2(\sim, +1)$.

As a second generalization, we assume that for each possible profile p, the signature contains a unary predicate $p(x)$ that is true in nodes of profile p. Clearly, the profile predicates can be removed from formulas, since they are definable in $\mathrm{FO}^2(\sim, +1)$. However, the availability of the profile predicates simplifies the logical part of the proof, especially the normal forms.

We now outline the structure of the proof. Essentially the idea is classical: first we compile formulas into automata, and then we test the automata for emptiness. However, due to the complexity added by data values, some more intermediate steps are required. These steps are: first, a simplification of the formulas and then, a simplification of the models considered. For simplified formulas over simplified models we then proceed with the standard automata-theoretic approach. The simplified formulas are defined in Definition 3.2. The simplified models require that few zones have large outdegree, see Definition 3.3.

In the following definition, α and β denote **atomic types** of nodes, i.e., conjunctions of unary predicates or their negations (the unary predicates are the labels, the profile predicates, and the predicates from the existential second-order quantification). Note that *true* and *false* are atomic types, too, corresponding, respectively, to the empty conjunction and, e.g., $a(x) \wedge \neg a(x)$. Note that an atomic type does not need to specify values for all predicates, we will use the term label for such an atomic type.

Definition 3.2. A sentence of $\mathrm{EMSO}^2(\sim, +1)$ is in **data normal form** if its core is a conjunction of **simple formulas**, i.e. one of the following three kinds:

(a) **Data-blind** properties, i.e. sentences of $\mathrm{FO}^2(+1)$.

(b) "Each class contains at most one node of type α":

$$\forall x \forall y \; (x \sim y \wedge \alpha(x) \wedge \alpha(y)) \rightarrow x = y \; .$$

(c) "Each class with a node of type α also has a node of type β":

$$\forall x \exists y \; \alpha(x) \rightarrow (x \sim y \wedge \beta(y)) \; .$$

Definition 3.3. The *outdegree* of a data zone is the number of its adjacent data zones.

We now present the three main steps of the proof in slightly more detail:

—First, in Section 3.2 we show that each sentence of $\mathrm{EMSO}^2(\sim, +1)$ can be rewritten into one in data normal form, that is equivalent w.r.t. satisfiability.

—Next, in Section 3.3, we show that for each sentence in φ in data normal form, one can calculate M, N such that if φ has a (finite) model, then it has one where at most M data zones have outdegree greater than N.

—Finally, in Section 3.4, we decide satisfiability of sentences over models where at most M data zones have outdegree greater than N. This is done by constructing a certain kind of (extended) tree automaton with decidable non-emptiness problem.

We begin by stating the following simple lemma. This is mainly to illustrate how complexity can be pushed from the core into the second-order prefix:

LEMMA 3.4. *Sentences in data normal form are closed under conjunction and disjunction.*

PROOF . Closure under conjunctions follows immediately from the definition. For a disjunction $\varphi \vee \psi$, we add a new unary predicate R to the second-order prefix, along with a data-blind sentence enforcing that R holds either everywhere or nowhere. The idea is that R tells us which of the disjuncts holds. We then add R to all the types occurring in φ and add $\neg R$ to all the types occurring in ψ. □

3.1 Data-free regular languages for free

In this section, we recall that any regular property of trees without data can be encoded in $\mathrm{EMSO}^2(+1)$. The basic idea is that the second-order prefix along with the successor relation in the core can be used to express automata on trees without data.

There are many ways of defining regular languages of unranked ordered trees without data (we call these *data-free regular languages* from now on). We use automata over unranked trees similar to the ones defined in [Brüggemann-Klein et al. 2001], also known as hedge automata. Our automata are close to the deterministic version of unranked tree automata considered by [Cristau et al. 2005; Martens and Niehren 2005] in the setting of minimization. They are easily translatable into $\mathrm{EMSO}^2(+1)$ and can be extended in a straightforward manner by constraints in Subsection 3.4.

A **nondeterministic automaton over unranked trees** consists of a finite set Q of states, subsets I and J of Q, along with relations

$$\delta_h, \delta_v \subseteq Q \times \Sigma \times Q \ ,$$

which are called the **horizontal** and **vertical** transition relations respectively. A **run** of such an automaton over a Σ-tree t is a labeling $\rho : V \to Q$ of nodes by states such that every node v with label a satisfies all conditions below:

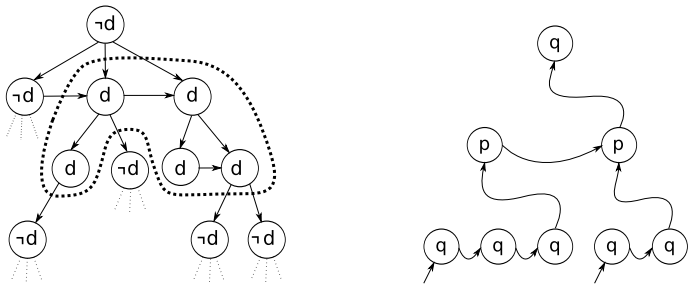
—If v is a leaf then $\rho(v) \in I$.

—If v has no horizontal predecessor then $\rho(v) \in J$.

—If v has a horizontal successor w, then the triple $(\rho(v), a, \rho(w))$ belongs to the horizontal transition relation δ_h.

—If v has no horizontal successor and its parent is w, then the triple $(\rho(v), a, \rho(w))$ belongs to the vertical transition relation δ_v.

A run is **accepting** when the state and label of the root belong to the designated **accepting** set $\mathcal{F} \subseteq Q \times \Sigma$. A tree is **accepted** if it admits an accepting run.



A zone A run

Fig. 1. Illustration of zones (each node is represented with its data value) and runs (each node is represented with the state given by the run).

The equivalence between $\mathrm{EMSO}^2(+1)$ and automata over unranked trees works along the same lines as for ranked trees (see [Neven and Schwentick 2002; Carme et al. 2004] for similar results):

THEOREM 3.5. *Nondeterministic automata over unranked trees accept exactly the languages definable in* $\mathrm{EMSO}^2(+1)$.

Furthermore, runs of nondeterministic automata can be captured by data-blind formulas, plus additional predicates. An **automaton formula** is a sentence

$$\exists R_1 \dots R_n \forall x \forall y \quad \begin{cases} (E_{\rightarrow}(x,y) \;\rightarrow\; \varphi_1) \wedge \\ ((E_{\downarrow}(x,y) \wedge \mathrm{rmchild}(y)) \;\rightarrow\; \varphi_2) \wedge \\ (\mathrm{root}(x) \;\rightarrow\; \varphi_3) \wedge \\ (\mathrm{leaf}(x) \;\rightarrow\; \varphi_4) \wedge \\ (\mathrm{lmchild}(x) \;\rightarrow\; \varphi_5) \end{cases}$$

where the formulas φ_1, φ_2 are boolean combinations of types for x and y, the formulas $\varphi_3, \varphi_4, \varphi_5$ are boolean combinations of types for x, $\mathrm{root}(x)$ says x is the root, $\mathrm{leaf}(x)$ says x is a leaf, and $\mathrm{lmchild}(x)$ and $\mathrm{rmchild}(x)$ say x has no left (resp., right) sibling.

FACT 3.6. *Automaton formulas capture exactly the data-free regular languages.*

PROOF. The sets $R_1, \dots, R_n$ correspond to the states used in the run. The core formula checks the consistency of the run. □

Since regular languages are closed under complementation, the above fact implies:

FACT 3.7. *Properties defined by automaton formulas are closed under negation.*

3.2 Reduction to data normal form

In this subsection, we show how sentences of $\mathrm{EMSO}^2(\sim, +1)$ can be rewritten into data normal form. A similar result for data strings is shown in [Bojańczyk et al. 2006].

For complexity reasons that will be apparent in the next section we will actually use a stronger normal form, called automata data normal form. A sentence φ of $\mathrm{EMSO}^2(\sim, +1)$ is in *automata data normal form* if it is in data normal form

and its data-blind formulas are automaton formulas. In the rest of this section all data-blind formulas will be implicitly automaton formulas.

PROPOSITION 3.8. *Every sentence of* $\mathrm{EMSO}^2(\sim, +1)$ *over profiled trees can be rewritten in exponential time into an equivalent one in automata data normal form (of exponential size).*

We begin our proof of Proposition 3.8 with some examples. Consider first the property "every class contains at least one node of type α", which is not in data normal form. We can express it easily in data normal form, using a simple sentence of kind (c): "every class with a node of type *true* also has a node of type α". As a second example, consider a sentence stating that "each class contains either zero or at least two nodes of type α". This can be transformed[3] into data normal form as follows:

> Add a unary predicate R such that each class with at least one node of type $\alpha \wedge R$ also has a node with $\alpha \wedge \neg R$, and vice versa (simple formulas of kind (c)).

The second example shows how simple formulas can be used alongside existentially quantified predicates, to express non-simple properties. This sort of reasoning will be heavily used later on in this section.

3.2.1 *Intermediate Normal Form.* As a first step to prove Proposition 3.8, we show that every sentence of $\mathrm{EMSO}^2(\sim, +1)$ can be effectively transformed into an equivalent sentence in "intermediate normal form" of exponential size, with exponentially many new unary predicates R_i. Then, in Section 3.2.2, we will show how to transform the intermediate normal form into automata data normal form.

Essentially, a sentence of intermediate normal form is an $\mathrm{EMSO}^2(\sim, +1)$ sentence where the quantifier depth of the core is 2, and the quantifier-free subformulas are slightly simplified. More precisely, a sentence of $\mathrm{EMSO}^2(\sim, +1)$ is said to be in **intermediate normal form** if its core is a conjunction of formulas of the following two kinds:

(1) $\forall x \forall y \quad \big[\alpha(x) \wedge \beta(y) \wedge \delta(x,y)\big] \rightarrow \mathrm{dist}_{\leq 1}(x,y)$
(2) $\forall x \exists y \quad \alpha(x) \rightarrow \big[\beta(y) \wedge \delta(x,y) \wedge \epsilon(x,y)\big]$

where

—α, β are types,
—$\mathrm{dist}_{\leq 1}(x,y)$ is the disjunction

$$E_{\downarrow}(x,y) \vee E_{\downarrow}(y,x) \vee E_{\rightarrow}(x,y) \vee E_{\rightarrow}(y,x) \vee x = y$$

—$\delta(x,y)$ is either $x \sim y$ or $x \not\sim y$, and
—$\epsilon(x,y)$ is one of $E_{\downarrow}(x,y)$, $E_{\downarrow}(y,x)$, $E_{\rightarrow}(x,y)$, $E_{\rightarrow}(y,x)$, $x = y$, or $\neg\mathrm{dist}_{\leq 1}(x,y)$.

We note that this normal form is quite similar to the one obtained in [Bojańczyk et al. 2006] for data words.

[3]Of course, this property can also be expressed as the negation of "at least one node" and "at most one node".

LEMMA 3.9. *Every sentence of* $\mathrm{EMSO}^2(\sim,+1)$ *can be transformed in exponential time into an equivalent sentence in intermediate normal form (of exponential size, with at most exponentially many new unary predicates).*

PROOF. The overall strategy is not new: we reduce the quantifier depth of the core to 2, then we add unary predicates that 'color' certain distinguished positions, resp. classes containing distinguished positions. These additional colors are then used to simplify the formulas.

The formal proof proceeds in three steps.

Step 1: Quantifier depth 2.

The first step uses the well-known *Scott Normal Form* (see for example [Grädel and Otto 1999]). Each $\mathrm{EMSO}^2(\sim,+1)$ sentence is equivalent (with a linear blowup) to one where the core is of the form

$$\forall x\forall y\ \chi \wedge \bigwedge_i \forall x\exists y\ \chi_i,$$

where χ and each χ_i are quantifier-free. Intuitively, the additional relations state which subformulas of φ are satisfied at a given position.

Step 2: Dealing with $\forall x\forall y\ \chi$.
In the second step we show that the formula $\forall x\forall y\ \chi$ can be replaced by a formula

$$\exists R_1 \cdots \exists R_m \qquad \bigwedge_i \theta_i \ \wedge \bigwedge_{i=1,2,3} \forall x\exists y\ \xi_i$$

where the ξ_i are again quantifier-free and each θ_i is of the form (1) in the intermediate normal form. Moreover, the number of θ_i is at most exponential.

To this end, we first rewrite $\forall x\forall y\ \chi$ into the following form:

$$\begin{array}{ll}\forall x\forall y & (\mathrm{dist}_{>1}(x,y) \to \psi_{>1}(x,y)\wedge\\ & (E_{\to}(x,y) \to \psi_{\to}(x,y))\wedge\\ & (x=y \to \psi_{=}(x,y))\wedge\\ & (E_{\downarrow}(y,x) \to \psi_{\downarrow}(y,x)))\end{array}$$

where $\mathrm{dist}_{>1}(x,y)$ is $\neg\mathrm{dist}_{\leq 1}(x,y)$ and the ψ formulas are quantifier-free and use only $\sim$ and the unary predicates. Their size is bounded by the size of χ.

Over unranked trees this is logically equivalent to:

$$\begin{array}{lll} & \forall x\forall y\ \big(\mathrm{dist}_{>1}(x,y) \to \psi_{>1}(x,y)\big) & \\ \wedge & \forall x\exists y\ \big(\mathrm{root}(x) \vee (E_{\downarrow}(y,x)\ \wedge \psi_{\downarrow}(x,y))\big) & (\xi_1)\\ \wedge & \forall x\exists y\ \big(\mathrm{rmchild}(x) \vee (E_{\to}(x,y)\ \wedge \psi_{\to}(x,y))\big) & (\xi_2)\\ \wedge & \forall x\exists y\ \big((x=y\ \wedge \psi_{=}(x,y))\big)\ . & (\xi_3)\end{array}$$

Here $\mathrm{root}(x)$ and $\mathrm{rmchild}(x)$ are testing the profile of x. They express the fact that x has no parent or no right sibling, respectively.

We denote the latter three conjuncts by ξ_1,ξ_2,ξ_3 and we will take care of them in Step 3. It only remains to deal with the first conjunct

$$\psi = \forall x\forall y\ \big(\mathrm{dist}_{>1}(x,y) \to \psi_{>1}(x,y)\big)\ .$$

By turning the $\neg\psi_{>1}$ into DNF (with an exponential blowup), we can rewrite $\psi_{>1}$ into a conjunction of formulas of the form

$$\neg(\alpha(x) \wedge \beta(y) \wedge \delta(x,y)) \ ,$$

where α, β are types and δ is either $x \sim y$ or $x \not\sim y$. Since conjunction distributes over implication and universal quantification, ψ becomes a conjunction of formulas of the form

$$\forall x \forall y \ \mathrm{dist}_{>1}(x,y) \to \neg(\alpha(x) \wedge \beta(y) \wedge \delta(x,y)) \ ,$$

which is equivalent to a formula of kind (1).

Step 3: Dealing with $\bigwedge_i \forall x \exists y \ \chi_i$.

In the last step, we show that each formula $\forall x \exists y \ \chi$ can be translated into an equivalent formula $\exists R'_1 \cdots \exists R'_n \bigwedge_i \theta_i$ in which each θ_i is of the kind (2) above. Moreover, the number of θ_i and the number n of additional predicates are both at most exponential.

First, χ can be written as a disjunction of an exponential number of formulas φ_j of the form

$$\varphi_j = \alpha_j(x) \wedge \beta_j(y) \wedge \delta_j(x,y) \wedge \epsilon_j(x,y) \ ,$$

where the $\alpha_j, \beta_j, \delta_j, \epsilon_j$ are of corresponding forms as in (2). It only remains to eliminate the disjunction. To this end, we add for each disjunct a new unary predicate $R_{\chi,j}$ with the intended meaning that if $R_{\chi,j}$ holds at a position x then there is a y such that $\varphi_j(x,y)$ holds.

More formally, we existentially quantify over the $R_{\chi,j}$ predicates and then require:

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

Moreover, we need to enforce that for each x and χ at least one $R_{\chi,j}$ holds. This is done using a formula of kind (2): we write that no position has all $R_{\chi,j}$ formulas false at the same time. Formally, we write the formula $\forall x \exists y \ (\bigwedge_j \neg R_{\chi,j}(x)) \to \mathit{false}$.

By putting together the obtained formulas we get a sentence in intermediate normal form. □

3.2.2 *Data normal form.* In this section we rewrite sentences from intermediate normal form into automata data normal form. More precisely, we show that every conjunct $\theta \in \{\theta_1, \ldots, \theta_n\}$ of a formula

$$\varphi = \exists R_1 \cdots R_m \ \ (\theta_1 \wedge \ldots \wedge \theta_n)$$

in intermediate normal form can be transformed into automata data normal form with a linear blowup. Since formulas in automata data normal form are closed under conjunction without any additional cost, we obtain a linear translation from intermediate normal form to automata data normal form. This completes the proof of Proposition 3.8.

Let then $\theta \in \{\theta_1, \ldots, \theta_n\}$ be a conjunct as above. Recall that θ can be of two kinds, (1) or (2), each of which talks about types α, β. Assume first that the type α is a single predicate P, and the type β is a single predicate Q. Under

this assumption, there are finitely many possible formulas θ; we will show that in each case one can find an equivalent formula in automata data normal form. In particular, the translation of θ into automata data normal form has constant cost. In the more general case, where the types α, β use more than one predicate, we existentially quantify over new predicates P, Q and add a data-blind formula "P holds in exactly the nodes with type α, and Q holds in exactly the nodes with type β". This formula is the reason for the linear cost of translating θ into automata data normal form.

The idea is always the same: we add new existentially quantified predicates which are used to mark nodes with certain properties and which then allow to express the property at hand with simple formulas. Below, we denote as P-node a node where predicate P holds. A P-class (P-zone, resp.) denotes a class (a zone, resp) containing an P-node.

We begin with some auxiliary properties:

(i) "No class contains both R-nodes and S-nodes." We add a new predicate P. Using a simple formula of kind (b), we require that no class contains two P-nodes. Then, using two simple formulas of kind (c), we check that each R-class contains a node with $P \wedge R \wedge \neg S$, and each S-class contains a node with $P \wedge S \wedge \neg R$.

(ii) "Predicate R holds in all nodes that belong to an S-class." This is checked by a formula of kind (i): "no class contains both R and $\neg R$"; one formula of kind (c): "each R-class has an S-node"; and one data-blind formula: "each S-node is an R-node".

(iii) "Predicate R marks the nodes of exactly one class." We add a new predicate S. We use (ii) to say that R holds in exactly the nodes that belong to an S-class. Finally, we add the data-blind property: "There is exactly one S-node".

(iv) "Predicate R holds in the nodes that belong to an S-zone". We verify that each zone has only R-nodes, or only $\neg R$-nodes; moreover, a zone is an R-zone if and only if it has an S-node. Both properties can be checked by automata formulas using the profiles.

(v) "Each class has at most one S-zone". First, we use (iv) to mark by a new predicate R each node from an S-zone. Note that each zone has exactly one node for which neither the parent nor the right sibling is in the same class. Such a node, called here the zone root, can be identified by its profile. Therefore the property in this item is equivalent to saying that each class contains at most one zone root with predicate R, which is a simple formula of kind (b).

The translation of θ into automata data normal form is by case analysis. By the definition of the intermediate normal form, θ may be in one of the two forms (1) or (2).

Case (1) $\theta = \forall x \forall y \quad \big[P(x) \ \wedge \ Q(y) \ \wedge \ \delta(x,y)\big] \ \rightarrow \ \mathrm{dist}_{\leq 1}(x,y)$:
Case (1.1): $\delta(x,y)$ is $x \sim y$. In this case, θ says that every two occurrences of P and Q in the same class must be adjacent or identical. Clearly, θ implies that, if a class contains both P-nodes and Q-nodes, then all

such nodes must be in the same zone. We add a new predicate R. Using (ii), we ensure that R marks all nodes from $P \wedge Q$-classes. We then use (v) to express that each R-class has at most one zone with P or Q. An automaton formula can check that in every R-zone all P- and Q-nodes are adjacent or identical.

Case (1.2): $\delta(x,y)$ is $x \not\sim y$. In this case, θ says that every occurrences of P and Q in different classes must be adjacent. The crucial observation is as follows: if v, w are two nodes in a tree, then at most two nodes are adjacent to both v, w (the extreme case is when we can connect v to w via a sibling and the parent). Therefore, if there are two P-classes then there are at most two Q-nodes outside these two classes. In particular there are at most four classes containing either P- or Q-nodes. Combining (i), (ii) and (iii) above, we may assume that these are marked by predicates $R_1, \ldots, R_4$. In the presence of these predicates, θ boils down to an automaton formula. Otherwise, if there is only one P-class, then we mark it using an extra predicate R as in (iii) and an automaton formula checks that every $R \wedge P$-node and $\neg R \wedge Q$-node are adjacent.

Case (2) $\theta = \forall x \exists y \quad P(x) \rightarrow \big[Q(y) \ \wedge \ \delta(x,y) \ \wedge \ \epsilon(x,y)\big]$:

If $\epsilon(x,y)$ implies $\text{dist}_{\leq 1}(x,y)$, then θ can be checked by an automaton formula, which uses profiles to check if adjacent nodes have the same data value. Thus, we assume that $\epsilon(x,y)$ is $\text{dist}_{>1}(x,y)$.

Case (2.1): $\delta(x,y)$ is $x \sim y$. This means that each P-node needs a non-adjacent Q-node in the same class. First, we have to require that each class with a P-node also has a Q-node, this is a simple formula of kind (c). Using (iv), we can mark each Q-zone with a predicate R. Using (v), we can mark each class with at most one Q-zone by a predicate S. If a P-class has two Q-zones (which is equivalent to not having S) then θ holds for all nodes in the class. The remaining case to consider is when a class has one Q-zone; in this case all P-nodes in this zone must have a non-adjacent Q-node in the same zone. Putting all this together, the property θ boils down to: "each $P \wedge R \wedge S$-node has a non-adjacent Q-node in the same zone". This is can be verified by an automaton formula that uses the profiles for checking the zones.

Case (2.2): $\delta(x,y)$ is $x \not\sim y$. This means that each P-node needs a non-adjacent Q-node in a different class. Using an additional predicate, it can be checked whether Q occurs in exactly one class (properties (ii) and (iii)). If this is the case, then θ translates to an automaton formula, plus a formula saying that no class has both P- and Q-nodes (property (i)). Otherwise, let v, w be Q-nodes in different classes. As we noted before, there are at most two nodes x, y adjacent to both v, w. All P-nodes in classes other than those of x, y already satisfy θ. The classes of x, y can be marked with two additional predicates and an automaton formula can be used to check θ for the P-nodes belonging to the classes of x, y.

3.3 A Small Model Property

We now proceed with the second part of the proof of Theorem 3.1. In this part, we show:

PROPOSITION 3.10. *For every sentence φ in automata data normal form, one can compute constants M, N such that φ is satisfied in a data tree only if it is satisfied in one where at most M data zones have outdegree more than N.*

This proposition is the most involved and technical part of our proof of Theorem 3.1. We fix a satisfiable sentence φ in automata data normal form for the rest of Section 3.3. We begin with a model of φ. We view this model as a first order structure where all the unary monadic predicates that are existentially quantified in φ are part of the signature. Our goal is to transform the model into one with only a few data zones of unbounded outdegree.

3.3.1 *A few definitions.* We use the following terminology.

—A **label** is a type that includes every unary predicate in the signature (or its negation). For simplicity, we denote this set of labels by Σ, too. So from now on, labels contain all the information about the unary predicates (in particular, the profiles and also the states of automaton formulas). Note also that Σ is exponential in the size of the signature, and that all labels are mutually exclusive.

—The set of all children of some node is called[4] a **sibling group**. (Two nodes are called **siblings** if they have the same parent node.) A contiguous sequence of siblings is called an **interval**. For two siblings v and w, where w is to the right of v, we write $L[v, w]$ for the interval with left end v and right end w. We write $L(v, w)$, $L(v, w]$ and $L[v, w)$ for the same interval without v and w, without v and without w, respectively.

—The **depth** of a node, a sibling group, or an interval is its distance from the root.

—The **induced forest** of an interval I is the set of descendants of all nodes in I (including I).

—A **twin-pair** (v, v') is a pair of two consecutive siblings, i.e., v' is the horizontal successor of v. A twin-pair is **pure** if $v \sim v'$. The twin-pair (v, v') is called a ***σ*-twin-pair** if the label of v' is σ.

—The **left interface** of an interval I is the twin-pair (v, v') consisting of the leftmost node v' of I and its left neighbor $v \notin I$. If there is no such left neighbor then we set $v = \perp$. Correspondingly, the **right interface**, contains the rightmost node of I and its right neighbor.
A non-pure interface (resp. non-pure twin-pair), i.e., an interface (v, v') with $v \not\sim v'$, is called a **border**. As above, a border (v, v') is called ***σ*-border** if the label of v' is σ.

—An interval is ***σ*-looping** if both its left and right interfaces are σ-twin-pairs.

—A node with data value d is called a ***d*-node**. An interval consisting of d-nodes only is called ***d*-pure**. If the exact value d of the data does not matter, we simply call it **pure**.

[4]Of course, *group* is not meant in an algebraic sense here.

—An interval is called **complete** if both its interfaces are borders.

—A node that has the same data value as its parent is called **attached**. An interval with an attached node is also called attached.

—If the parent of an interval (node, sibling group) has value d we call it a ***d*-parent** interval (node, sibling group).

—The **frontier** $\mathrm{Front}(Z)$ of a set of nodes Z is the set of nodes outside Z that are connected to Z by a (horizontal or vertical) edge.

—For a data value d, a ***d*-path** is a set of d-nodes connected by the *vertical* successor relation. A **data path** is a d-path for some d. By $\mathrm{Sub}(v, w)$ we denote the set of nodes that are in the subtree of v but not in the subtree of w (including v and excluding w).

Figure 2 illustrates some of these terms.

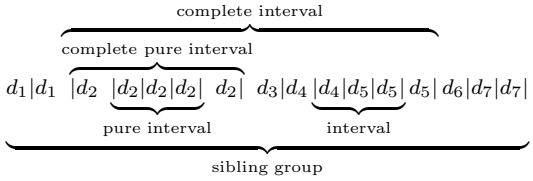


Fig. 2. Different types of intervals

3.3.2 *Proof strategy.* In a nutshell, the proof of Proposition 3.10 is a long sequence of steps combining cut-and-paste and counting arguments; in each one we modify an existing model into one where the data zone outdegree is progressively smaller. More precisely, we will show that there exists a model for φ that satisfies the following properties:

—At most M_1 complete pure intervals have more than N_1 nodes (Proposition 3.20).

—At most M_2 sibling groups contain more than N_2 complete pure intervals (Proposition 3.21).

—At most M_3 zones contain data paths with more than N_3 nodes (Proposition 3.25).

All the constants used above depend only on the size of the sentence φ, see Table I for their asymptotic values[5]. Once we have obtained such a model, one can easily see that at most

$$M = M_1 + M_2 + M_3$$

"large" zones contain more than

$$(N_1 \cdot N_2)^{N_3+1}$$

[5]The values of the constants mentioned in Table I are explicitely given in the proofs and the hidden numbers are not very big

nodes. Moreover, nodes in the zones that are not "large" have at most $N_2 + 3$ neighbors with a different data value. (N_2 is for the children, +2 is for the left and right siblings and +1 is for the parent.) Altogether, the zones that are not "large" have outdegree at most

$$N = (N_1 \cdot N_2)^{N_3+1}(N_2 + 3)$$

Thus, Propositions 3.20, 3.21 and 3.25 imply Proposition 3.10 with $M = |\Sigma|^{O(|\Sigma|^2)}$, $N = |\Sigma|^{|\Sigma|^3}$ (recall that Σ is the set of labels, which is exponential in the size of the signature of φ).

For the proofs of Propositions 3.20, 3.21 and 3.25 we first present, in Subsection 3.3.3, "local" versions. They show that there exists a model for φ where each data value d satisfies:

—there are at most K_1 complete d-pure intervals with more than L_1 nodes (Proposition 3.12),

—there are at most K_2 d-parent sibling groups with more than L_2 complete pure intervals (Proposition 3.13), and

—there are at most K_3 zones with data value d containing a data path with more than L_3 nodes (Proposition 3.17).

During the "local pruning", we will never change the data value of any node, we will only move nodes around the tree. The cut-and-paste techniques that we use are provided in Lemmas 3.11, 3.16, 3.19 and 3.24. Table I gives a summary of the constants used in the the mentioned propositions and lemmas.

Result	Const.	Value	Const.	Value
Prop. 3.12	K_1	$O(\|\Sigma\|)$	L_1	$O(\|\Sigma\|)$
Lemma 3.14	K'_1	$O(\|\Sigma\|)$	L'_1	$O(1)$
Lemma 3.15	K'_2	$O(\|\Sigma\|)$	L'_2	$\|\Sigma\|^{O(\|\Sigma\|)}$
Prop. 3.13	K_2	$O(\|\Sigma\|)$	L_2	$\|\Sigma\|^{O(\|\Sigma\|)}$
Prop. 3.17	K_3	$O(\|\Sigma\|)$	L_3	$O(\|\Sigma\|)$
	K	$O(\|\Sigma\|)$	L	$\|\Sigma\|^{O(\|\Sigma\|)}$
Prop. 3.20	M_1	$\|\Sigma\|^{O(\|\Sigma\|^2)}$	N_1	$O(\|\Sigma\|^2)$
Lemma 3.22	M'_1	$\|\Sigma\|^{O(\|\Sigma\|)}$	N'_1	$\|\Sigma\|^{O(\|\Sigma\|)}$
Lemma 3.23	M'_2	$\|\Sigma\|^{O(\|\Sigma\|^2)}$	N'_2	$O(\|\Sigma\|^3)$
Prop. 3.21	M_2	$\|\Sigma\|^{O(\|\Sigma\|^2)}$	N_2	$\|\Sigma\|^{O(\|\Sigma\|)}$
Prop. 3.25	M_3	$\|\Sigma\|^{O(\|\Sigma\|^2)}$	N_3	$O(\|\Sigma\|^2)$

Table I. The constants used for pruning.

We use basically two kinds of transformation steps. The first kind, which we call **horizontal transfer** involves an interval I and a twin-pair (u, u'). Under suitable conditions, I together with its induced forest can be removed from its current location and inserted between u and u'. The second kind is called **vertical transfer** and it incorporates two nodes v, w, where w is a descendant of v and two nodes u, u', where u' is the parent of u. Again under suitable conditions, the part of the tree between v and w can be inserted between u and u'.

Let I be a d_1-parent interval with left interface (v, v') and right interface (w, w'). We are interested in sufficient conditions for placing the induced forest of I between

the two nodes of a d_2-parent twin-pair (u, u') such that the resulting tree is still a model of our fixed formula φ. To this end, we call an interval I **compatible** with a twin-pair (u, u') if the following conditions hold:

(i) The labels of u', v', w' are the same.

(ii) If $u \sim u'$ then $u \sim v \sim w$.

(iii) If $u \not\sim u'$ then $u \not\sim v'$, $w \not\sim u'$ and $v \not\sim w'$.

(iv) If d_1 or d_2 occur in I then $d_1 = d_2$.

We recall that the label of a node implies its profile. In particular, in case (ii) also $w \sim w'$ and $v \sim v'$ hold.

3.3.3 *Pruning locally.* In this subsection we show that there is a model, where for each data value d, there are few large zones with value d. The idea behind the proofs is to shift intervals along with their induced forests around the tree. For this purpose we will use the following lemma, which is illustrated in Figure 3.

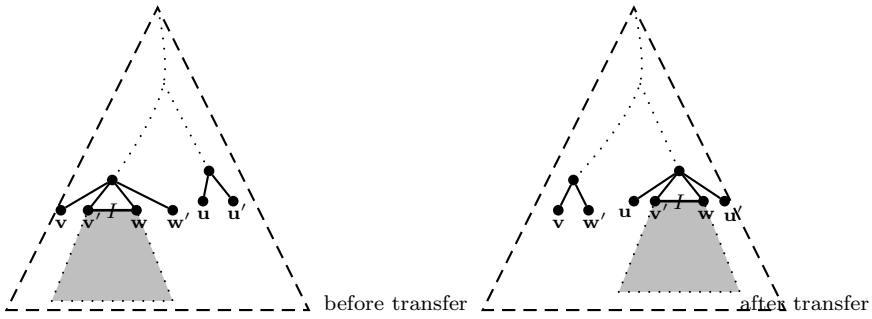


Fig. 3. Illustration of the Local Horizontal Transfer Lemma.

LEMMA 3.11. *(Local Horizontal Transfer Lemma).*
Let t be a model of φ containing a d_1-parent interval I, and a d_2-parent twin-pair (u, u') which does not belong to the induced forest of I.
If I is compatible with (u, u') then the tree t' obtained from t by transferring the induced forest of I between u and u' is a model, too.

PROOF. Clearly, the local transfer does not change the truth value of any subformula of φ of the kind (b) or (c) in the automata data normal form. It remains to consider the automata subformulas of φ. Compatibility ensures in particular that the profile of each transferred node does not change (items (ii) and (iii) in the definition take care of the left/right sibling, and item (iv) takes care of the parent). Let (v, v') and (w, w') be the left and right interfaces of I. Item (i) of compatibility says that u', v' and w' have the same labels. Therefore (v, w'), (u, v') and (w, u') have valid horizontal transitions for the automaton formulas of φ because (w, w'), (v, v') and (u, u') did. Therefore t' satisfies all automaton formulas of φ. □

Small complete pure intervals. First we show that there is a model with few large d-pure intervals, for every data value d. Recall that the asymptotic values of the constants K_1 and L_1 from the lemma below may be found in Table I together with the constants used in the other propositions and lemmas.

PROPOSITION 3.12. *There is a model of φ with at most K_1 complete d-pure intervals of size more than L_1, for every data value d.*

PROOF. Let d be a fixed data value. We will show that parts of pure intervals with data value d can be transferred to a bounded number of pure intervals until the property stated in the proposition is satisfied.

For every possible node label σ, we fix a node u_σ of minimal depth that has label σ and data value d (this node may be undefined). We will use the node u_σ as the target of a Local Horizontal Transfer. The minimality of the depth guarantees that the Local Horizontal Transfer is applied with a target node which is not a descendant of the transferred interval. Let U denote the set of these nodes. Clearly, $|U| \leq |\Sigma|$.

Let I be a d-pure interval not containing any node of U. If I has more than $|\Sigma|$ nodes there must be two nodes v and v' in I with the same label, say $\sigma \in \Sigma$. Recall that the label σ induces the same profile for the nodes v, v' and u_σ. Thus, $L[v, v')$ is compatible with (u, u_σ), where u is the left neighbor of u_σ. Then, by Lemma 3.11, $L[v, v')$ and its induced forest can be moved between u and u_σ, and the resulting tree is again a model of φ.

We iterate this step until there are no more d-pure intervals of size more than $|\Sigma|$ besides those containing the nodes of U. This proves the statement of the lemma with $K_1 = |\Sigma|$ and $L_1 = |\Sigma|$. □

Few complete pure intervals. We now proceed to show that for any given data value d, there are few d-parent sibling groups with many complete pure intervals. This part of the local pruning argument is the most involved one.

PROPOSITION 3.13. *There is a model of φ with at most K_2 d-parent sibling groups with more than L_2 complete pure intervals, for every data value d.*

Before we proceed with the proof, we note that we will only be cutting and pasting complete intervals. In particular, we will never change the length of any pure interval, and therefore the model obtained will also satisfy the properties that have been enforced in Proposition 3.12. More generally, in each step of Subsection 3.3, the modified models will keep the properties from the previous steps.

The proof of Proposition 3.13 uses Lemmas 3.14 and 3.15 which we prove first.

We call two twin-pairs **data-equivalent** if they agree in the data values of their left and their right components. That is, (v, v) and (w, w') are data-equivalent, if $v \sim w$ and $v' \sim w'$.

LEMMA 3.14. *There is a model of φ in which, for every data value d, there are at most K_1' d-parent sibling groups with more than L_1' σ-borders that are pairwise non data equivalent for some $\sigma \in \Sigma$.*

PROOF. Let the data value d be fixed.

CLAIM. There is a model having at most $|\Sigma|$ d-parent sibling groups with at least 5 σ-borders (v, v') with pairwise distinct left data values, for some $\sigma \in \Sigma$.

For each label σ, let B_σ be a topmost d-parent sibling group containing at least 5 σ-borders with distinct left data values. If no such B_σ exists, B_σ is undefined. The sibling groups B_σ will be used as targets of Local Horizontal Transfers. The minimality of the depth of B_σ will guarantee that the Local Horizontal Transfer is applied with target nodes that are not descendants of the transferred interval.

Let $\mathcal{B}$ be the set of all sibling groups B_σ. We show that from any other d-parent sibling group C containing more than 5 σ-borders with pairwise different left values (for some σ) one can transfer an interval into $\mathcal{B}$. Thereby we diminish the size of C. By iterating this step, and repeating the process for all labels, we end with a model which satisfies the statement of the claim.

Let C be such a d-parent sibling group. Therefore $C \notin \mathcal{B}$ and C has at least 5 different left values in σ-borders for some σ.

Let $(w_1, w_1'), \ldots, (w_5, w_5')$ be σ-borders in C with distinct left values, ordered by their position in the sibling group from left to right. Let $(u_1, u_1'), \ldots, (u_5, u_5')$ be σ-borders in B_σ with distinct left data values, also ordered from left to right.

We are going to find some indices $i < j$ and k such that the interval $L[w_i', w_j)$ can be transferred from C and placed between u_k and u_k'. The indices i, j, k are chosen carefully to ensure condition (iii) in the definition of compatibility (as all other conditions are already fulfilled). Thus we need: $w_i \not\sim w_j'$ and $u_k \not\sim w_i'$ and $w_j \not\sim u_k'$. Figure 4 illustrates this situation.

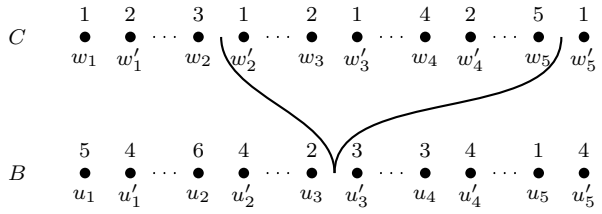


Fig. 4. Illustration of the proof of the claim in Lemma 3.14. Here, $k = 3$, $i = 2$, $j = 5$. The reader should verify that it is not possible to choose $i = 1$ (with any combination of j and k).

We now proceed to show that such indices i, j, k can be found.

—Since the values of $u_1, \ldots, u_5$ are all different, there must be some $k \in \{1, \ldots, 5\}$ such that at most one $i \in \{1, \ldots, 5\}$ satisfies $u_k \sim w_i'$. Let us fix such a k and let i_0 be the only index such that $u_k \sim w_{i_0}'$ (this index may be undefined). As the i we choose below is different from i_0, we will have $u_k \not\sim w_i'$.

—We choose j maximal such that $w_j \not\sim u_k'$, and $i \neq i_0$ minimal so that $w_i \not\sim w_j'$. Since $j \geq 4$ and $i \leq 3$ we immediately get $i < j$.

This concludes the proof of the claim.

Next, in a symmetric fashion we obtain a model in which, there are at most $|\Sigma|$ d-parent sibling groups containing at least 5 σ-borders with pairwise distinct right data values.

The remaining d-parent sibling groups have at most 4 σ-borders with different left values and at most 4 σ-borders with different right values, thus at most 16 σ-

borders which are pairwise non data-equivalent, for every label σ. Thus we proved the lemma with $L'_1 = 16$ and $K'_1 = 2|\Sigma|$.

Thus, the proof of Lemma 3.14 is completed. □

The following lemma not only disallows many σ-borders with the same data value but many σ-borders at all (relative to each data value).

LEMMA 3.15. *There is a model of φ in which for every data value d, there are at most K'_2 d-parent sibling groups with more than L'_2 σ-borders for some $\sigma \in \Sigma$.*

PROOF. We fix a data value d. First we apply Lemma 3.14 and obtain a model with at most K'_1 d-parent sibling groups with more than L'_1 non data-equivalent σ-borders, for every label σ. Let $\mathcal{B}$ denote the set of these sibling groups.

We fix some order on the set of labels, say $\Sigma = \{\sigma_1, \ldots, \sigma_s\}$. We will iteratively modify the model for each label. At each stage $j = 0, \ldots, s$ we will define constants k_j, l_j so that the model will satisfy the following invariant:

> There is a set $\mathcal{B}_j$ of at most k_j sibling groups such that in each d-parent sibling group outside $\mathcal{B}_j \cup \mathcal{B}$ the total number of σ-borders with $\sigma \in \{\sigma_1, \ldots, \sigma_j\}$ is at most l_j.

For the basis of the induction we set $\mathcal{B}_0 = \emptyset$, $k_0 = 0$ and $l_0 = 0$.

Assume that we already constructed $\mathcal{B}_{j-1}$, for $j \geq 1$, and we want to construct $\mathcal{B}_j$.

For two sibling groups B, C let $B \leq C$ if either B has a larger depth than C or their depth is the same and B has at most as many σ_j-borders as C. In the proof, we will be moving intervals from smaller to larger sibling groups (with respect to $\leq$).

Assume first that there are at most three d-parent sibling groups outside $\mathcal{B} \cup \mathcal{B}_{j-1}$, with more than

$$l_j := l_{j-1} + L'_1(l_{j-1} + 1)$$

σ_j-borders. In this case we add these sibling groups to $\mathcal{B}_j$, set k_j to be $k_{j-1} + 3$, and we are done.

Otherwise, we find four d-parent sibling groups $C_1 \leq C_2 \leq C_3 \leq C_4$ outside $\mathcal{B} \cup \mathcal{B}_{j-1}$, each of them with more than l_j σ_j-borders. We show below that in this case we can find two sibling groups among $\{C_1, C_2, C_3, C_4\}$, say B and C, with $B \leq C$ and use the Local Horizontal Transfer Lemma to transfer an interval with two σ_j-borders from B to C. Moreover, we can do this so that the transferred interval will not contain any σ_i-border with $i < j$.

To show that iterating the above transfer step terminates we will use the following order argument. With each data tree we associate two vectors W_1, W_2 which store information on the depths of nodes and numbers of borders of d-parent sibling groups, respectively. More specifically, the m-th entry of W_1 is the number of nodes at depth m in the tree, the m-th entry of W_2 is the number of d-parent sibling groups with m σ_j-borders for $m > l_j$, and zero otherwise. We order these vectors lexicographically, with the lowest entries considered first for W_1-vectors and the highest entries considered first for W_2-vectors. Eventually, we consider the lexicographic order on (W_1, W_2).

Notice that moving an interval with two σ_j-borders from B to C where $B \leq C$ either strictly decreases the depth of all moved nodes and thus increases W_1, or it does not change W_1 and strictly increases W_2. Since both the set of nodes and the number of sibling groups do not change, this step cannot be iterated indefinitely. At some point, we must get a model with at most three d-parent sibling groups outside $\mathcal{B} \cup \mathcal{B}_{j-1}$, and then $\mathcal{B}_j$ can be defined as above.

Because $C_1 \notin \mathcal{B} \cup \mathcal{B}_{j-1}$, it contains at most L'_1 pairwise non data-equivalent σ_j-borders and at most l_{j-1} σ_i-borders for $i = 1, \ldots, j-1$. By definition of l_j, the sibling group C_1 must contain two data equivalent σ_j-borders $(v_1, v'_1), (v_2, v'_2)$ (in that order) such that the interval $L(v_1, v'_2)$ does not contain any of the borders $\sigma_1, \ldots, \sigma_{j-1}$. Let d_1 and d_2 be the data values of v_1 and v'_1, respectively. Note that $d_1 \neq d_2$ by definition of borders.

Let (u, u') be a σ_j-border in C_k, with $k \in \{2, 3, 4\}$. If the data value of u is not d_2 and the data value of u' is not d_1 then the Local Horizontal Transfer Lemma allows us to move the interval $L[v'_1, v_2]$ (and its descendants) between u and u' and we have found our B and C with $B = C_1$ and $C = C_k$. (Note that the assumption $B \leq C$ guarantees that C is not in the induced forest of B.)

If this didn't work, then for every $k \in \{2, 3, 4\}$ and every σ_j-border (u, u') in C_k, either the data value of u is d_2 or the data value of u' is d_1. However, by construction each of the sibling groups C_2, C_3, C_4 has more than l_j σ_j-borders and less than l_{j-1} σ_i-borders, $i = 1, \ldots, j-1$. Therefore, since $L'_1 > 2$, in each of these sibling groups there must be either at least two σ_j-borders with left value d_2 or at least two σ_j-borders with right value d_1, such that there is no σ_i-border between them, for any $i < j$.

We conclude that there must be $l < m$ from $\{2, 3, 4\}$ and σ_j-borders (v_1, v'_1), (v_2, v'_2) in C_l and (u, u') in C_m, respectively, such that v_1, v_2 and u all have data value d_2, or that v'_1, v'_2 and u' all have data value d_1. In either case we can move $L[v'_1, v_2]$ from C_l between u and u' in C_m. So $B = C_l$ and $C = C_m$ in this case.

After performing the transfer from B to C, we might have increased the number of pairwise non data-equivalent σ_k-borders in C, for $k \geq j$, which we bounded in Lemma 3.14. If, for all $k \geq j$, C contains no more than L'_1 pairwise non data equivalent σ_k-borders, then we are done with the construction of $\mathcal{B}_j$ and we proceed with the next value for j. If this is not the case, we need to reapply Lemma 3.14, but we have to be careful in order to guarantee the convergence of the process. Recall the proof of Lemma 3.14. It identifies, for each label σ, one sibling group B_σ such that for all sibling groups containing more than L'_1 σ-borders with pairwise distinct left data values one of the σ-border can be transferred into B_σ (and similarly for right data values). The set $\mathcal{B}$ resulting from applying Lemma 3.14 is the union of all the B_σ. Assume C contains for some $k \geq j$ more than L'_1 σ_k-borders with distinct left data value. If C is not above B_{σ_k} then we can proceed with the proof of Lemma 3.14 and transfer some of these σ_k-border to B_{σ_k}. The problem arises when C is above B_{σ_k}. Then we need to replace B_{σ_k} with C in $\mathcal{B}$. Notice that when doing this the sibling groups in $\mathcal{B}$ will only get higher in the tree, thefore at some point, this case will never occur. Once we have the new set $\mathcal{B}$ we need to consider the number of σ-borders in B_{σ_k}, even for $\sigma \leq j$. From this point we process again all values for j starting from 1 as explained above. Note again that the process

terminates because $\mathcal{B}$ is modified only a finite number of time.

Finally, we obtain $L_2' = (L_1'|\Sigma|)^{(|\Sigma|+1)}$ and $K_2' = K_1' + 3|\Sigma|$. □

Proof of Proposition 3.13. Let $K_2 = K_2'$ and $L_2 = |\Sigma|L_2' + 1$. Whenever a sibling group contains more than L_2 complete pure intervals, there must be more than L_2' σ-borders, for some label σ. Hence, the sibling group must be one of the K_2' sibling groups with more than L_2' σ-borders. The statement of the proposition follows. □

Shallow zones. Thanks to Propositions 3.12 and 3.13, we know that for each d, almost all d-parent sibling groups contain few pure intervals, and almost all pure intervals with data value d contain few nodes. The only remaining reason for a zone with data value d to contain many nodes is a long path.

Long paths are eliminated using the following lemma. An illustration is given in Figure 5.

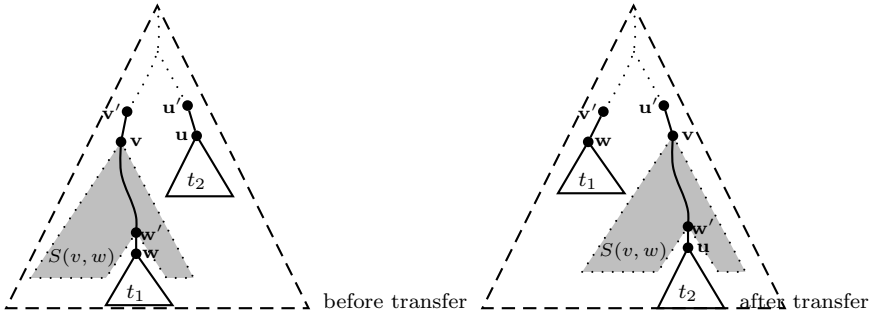


Fig. 5. Illustration of the Local Vertical Transfer Lemma.

Lemma 3.16 Local Vertical Transfer Lemma. *Let d be a data value and let u, v, w be d-nodes in a model t with the same label. If w is a descendant of v through a d-path and u is not in $Sub(v, w)$ then the tree obtained from t by placing $Sub(v, w)$ between u and its parent is also a model. (In the new tree, v becomes a child of the parent of u, u a child of the parent of w, and w a child of the parent of v.)*

Proof. It is easy to see that the only critical positions in this step are between v, w, u and their respective parents. The new tree t' is again a model of φ since

—all consistency conditions on labels of neighboring nodes are preserved (these are the conditions from the automaton subformulas of φ); recall also that node labels encode in particular the states of automaton formulas.

—classes are not changed, therefore all subformulas of kinds (b) and (c) of φ are preserved;

—t' is profiled consistently with the labels (labels imply profiles).

□

The following proposition shows that we can have only few long paths, for each data value. Together with Propositions 3.12 and 3.13, the proposition implies that each data value contains few large zones. If each data value contains few large nodes, then in particular few pure intervals are large, few sibling groups contain many complete intervals, and few paths are long. However, in the subsequent "global" constructions, we will need to refer separately to the three properties: small pure intervals, few complete intervals, and few long paths.

PROPOSITION 3.17. *There is a model of φ where for every data value d, there are at most K_3 d-zones containing a path with more than L_3 nodes.*

PROOF. Let d be fixed. For every label σ which implies that a node is attached, let v_σ be an arbitrarily chosen node of minimal depth with data value d and label σ. If such a node does not exist, v_σ is undefined. If v_σ is defined, then we initially define P_σ to be the singleton d-path consisting of v_σ. Let $\mathcal{P}_d$ be the set of paths P_σ, for all $\sigma \in \Sigma$; this set contains at most $K_3 = |\Sigma|$ paths. We will iteratively move parts of long d-paths into the paths from $\mathcal{P}_d$ until the statement of the proposition is satisfied.

To this end, we consider a d-path p that is disjoint with $\bigcup \mathcal{P}_d$ and has more than $L_3 = |\Sigma|+1$ nodes. Clearly there must be two nodes v, w on the path with the same label σ. As the lower of these nodes is an attached node, we can conclude that the label σ implies that a node is attached, and thus the upper node is an attached node as well. By Lemma 3.16, we can now move $\mathrm{Sub}(v, w)$ between some suitable node u in P_σ and its parent. Note that because u has minimal depth among the d-nodes of label σ, u is not in $\mathrm{Sub}(v, w)$.

It is easy to verify that the new model still satisfies the properties of Proposition 3.12 and Proposition 3.13. □

3.3.4 *Pruning globally.* For the rest of this section, we will only consider models of φ satisfying the local properties stated in Propositions 3.12, 3.13 and 3.17. In this subsection we prove their global counterparts, where the restriction "for every data value d" is lifted (and doing this we ensure that the local properties hold throughout the section). For this purpose, we need more involved transfer lemmas. Unlike the previous section, where nodes were only moved around, in this section we will actually be changing the data values of some nodes.

We first introduce some more notation. The **spill** of a node v is the part of the zone of v which is contained in the subtree of v (including v itself). The spill of an interval I is the union of the spills of its nodes. The ***d*-spill** of an interval I is the union of the spills of the d-valued nodes of I.

In the subsequent constructions, we will sometimes want to take nodes from a set Z and assign them a different data value d. This change, however, may violate the conditions stated in our formula, especially conditions (b) - "Each class contains at most one node of type α" - and (c) -"Each class with a node of type α also has a node of type β". Recall that the label of a node is a complete type containing each predicate or its negation. In order to avoid this problem we say that a set of nodes Z is **safe** if for each node of Z there is a node with the same data value and the same label outside of Z. From now on we will only change the data values of safe zones.

The following lemma shows that every set of nodes with the same data value (e.g. a zone or pure interval) can be made safe by removing at most $|\Sigma|$ elements:

LEMMA 3.18. *Let Z be a subset of a class in a model. There is a subset $Y \subseteq Z$ with at most $|\Sigma|$ elements such that $Z \setminus Y$ is safe.*

PROOF. For each label we select a node of that label in Z (if such a node exists) and put it into Y. It is now immediate that $Z \setminus Y$ is safe. □

We start with a global horizontal transfer lemma which is illustrated in Figure 6.

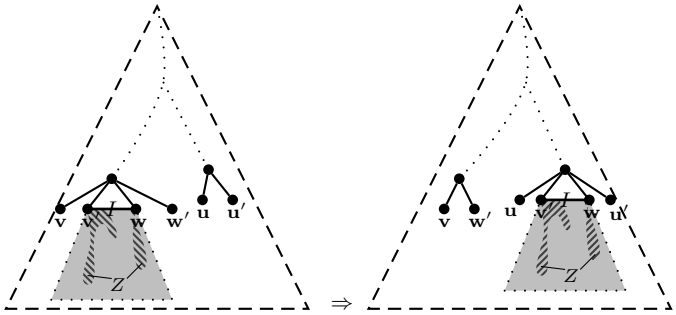


Fig. 6. Global Horizontal Transfer.

LEMMA 3.19. *(Global Horizontal Transfer Lemma).*
Let I be a d_1-parent interval of t and let (u, u') be a d_2-parent twin-pair which is not in the induced forest of I. Let (v, v') and (w, w') be the left and right interface of I, respectively. Assume that v' and w' have the same data value e_1. Furthermore, let e_2 be the data value of u' and let Z be the e_1-spill of I.

If t is a model of φ then the tree t' obtained by changing the data value of Z from e_1 to e_2 and moving the induced forest of I between u and u' is a model of φ if the following conditions hold.

(i) The labels of v', w', u' are the same.
(ii) If $d_2 \neq e_1$ then d_2 does not occur in I.
(iii) If $d_1 \neq e_1$ then d_1 does not occur in I.
(iv) $\mathrm{Front}(Z) \cup \{w\}$ does not contain any node with data value e_2.
(v) The set Z is safe.
(vi) All labels occurring in Z also occur in the class of e_2.

PROOF. As in the proof of Lemma 3.11, item (i) ensures the consistency of neighboring labels in t', therefore t' satisfies all automata subformulas, as t did.

Conditions (v) and (vi) on the labels in classes e_1 and e_2 ensure that simple conditions of kind (b) and (c) from φ are not violated by the transfer. Consider first a simple condition (b): "each class contains at most one node of type α". The transfer could violate this condition by adding a second α to the class of e_2. This is impossible by safety of Z, since every node in Z with type α appears at least twice

in e_1, and therefore a type α that appears in Z cannot be involved in a simple condition (b). Consider now a simple condition (c), which says that "each class with a node of type α also has a node of type β". This condition could be violated in one of two ways: by removing all β's from the class of e_1, or by adding an α without a β to the class of e_2. The first violation cannot hold by assumption on safety of Z, while the second violation cannot hold by assumption (vi), which says that all labels in Z also occur in the class of e_2.

Items (i)-(iv), plus the equal values of v', w' guarantee that t' is a consistently profiled data tree. To see this, note first that the data (in)equalities for (v, w'), (u, v') and (w, u') are trivially preserved (by item (i)). Let x, y be two nodes in the induced forest of I, and assume first that y is a left/right sibling or a child of x. Item (iv) ensures that if x changes its value, the data (in)equality for (x, y) is preserved. If y is the parent of x, and x belongs to I, then there are several cases to consider. If x has value different from e_1, d_1, then item (ii) applies. If x has value $d_1 \neq e_1$, then item (iii) applies. If x has value e_1, then item (i) applies, since the label of v', u' implies the profile. □

To be able to find a target twin-pair fulfilling item (iv) of Lemma 3.19 it is critical that Front(Z) is small and thus contains only a limited number of data values. The following definition gives a sufficient condition for Front(Z) being small. A node is called **admissible** if its spill

—contains only complete pure intervals with at most L_1 nodes,

—contains only nodes whose children sibling group has at most L_2 complete pure intervals, and

—has depth at most L_3.

Clearly, if a node is admissible, then its spill Z satisfies

$$|\text{Front}(Z)| \leq L \qquad \text{where } L = (L_1 L_2)^{L_3+1} .$$

Furthermore, from Propositions 3.12, 3.13 and 3.17 it follows that in each pure interval I, there are at most

$$K = K_1 + K_2 + K_3 \tag{$*$}$$

non-admissible nodes. More generally, each antichain (nodes pairwise incomparable with respect to the descendant relation) contains at most K non-admissible d-nodes, for any d.

Small intervals. Again, we prove first that it can be assumed that almost all pure intervals are small.

PROPOSITION 3.20. *There is a model of φ in which at most M_1 complete pure intervals have more than N_1 nodes.*

PROOF. We define the following constants (where K is the constant defined in

Eq. (∗)):

$$\begin{aligned} K' &= |\Sigma|L + 1 \ , \\ M_1' &= K'K_1|\Sigma| \ , \\ M_1 &= 2^{|\Sigma|}M_1' \ , \\ N_1 &= K|\Sigma| + (|\Sigma| + 1)^2 \ . \end{aligned}$$

In the remainder of the proof we call any complete pure interval of length more than N_1 **long**. For a set $\Gamma \subseteq \Sigma$ of labels, a **Γ-class** is a class in which exactly the labels from Γ occur. A **Γ-interval** is an interval from a Γ-class (which is not the same thing as an interval with exactly the labels from Γ).

The goal is to limit the number of long intervals to at most M_1. To this end, we construct a model in which, for each $\Gamma \subseteq \Sigma$ there are at most M_1' long intervals from Γ-classes. The transformation of the model is done separately, for each set Γ. Let thus Γ be fixed in the following.

The general idea is as follows. We start with a model t that satisfies all local properties from Section 3.3.3. For each label σ, we will fix a set $\mathcal{I}_\sigma$ with a bounded number of long Γ-intervals. We will then reduce the size of all other pure intervals by using the Global Horizontal Transfer Lemma to move safe subintervals of large pure intervals into long intervals from the sets $\mathcal{I}_\sigma$. Notice that safety guarantees that the initial pure interval remains a Γ-interval (no label has disappeared in the corresponding class) while the resulting target interval remains a Γ-interval by construction (no new label has been added to the corresponding class).

We begin by defining the sets $\mathcal{I}_\sigma$. Two cases need be considered.

(1) If σ occurs in at most K' long Γ-intervals, we put all the long intervals that contain the label σ into $\mathcal{I}_\sigma$.
(2) Otherwise, σ occurs in at least K' long Γ-intervals. We set $\mathcal{I}_\sigma$ to contain K' long Γ-intervals with distinct data values, each containing an occurrence of σ. These pure intervals will be the destination of horizontal transfers. They are chosen by traversing the tree in a root-to-leaf, left-to-right manner, and putting into $\mathcal{I}_\sigma$ the first K' long Γ-intervals with label σ and distinct data values. In particular, for each long Γ-interval $I \notin \mathcal{I}_\sigma$ that contains σ we have either a) the induced forest of I does not contain any intervals from $\mathcal{I}_\sigma$; or b) there exists some $J \in \mathcal{I}_\sigma$ with the same value as I, which does not belong to the induced forest of I.

Let $\mathcal{I}$ be the union of the sets $\mathcal{I}_\sigma$, for all labels σ. Clearly, $\mathcal{I}$ contains at most

$$M_1' = K'K_1|\Sigma|$$

pure intervals. We claim that if J is a long Γ-interval outside $\mathcal{I}$, then the Global Horizontal Transfer Lemma can be used to transfer some subinterval $I \subseteq J$ into $\mathcal{I}$. By iterating this process, we eventually obtain a model where at most M_1' pure Γ-intervals are long.

Let then J be a long Γ-interval outside $\mathcal{I}$. Since J is long, it contains a σ-looping subinterval, for some label σ. If $\mathcal{I}_\sigma$ contains a long interval I with the same data value as J, then we can use the Local Horizontal Transfer Lemma to move the σ-looping subinterval of J into $\mathcal{I}$. Otherwise, for any label σ that occurs twice

in J, the set $\mathcal{I}_\sigma$ does not contain any interval with the same data value as J. In particular, by the last sentence in case (2), the induced forest of J contains no element of $\mathcal{I}_\sigma$.

Like any pure interval, J contains at most K non-admissible nodes. Since J is long,

$$|J| > N_1 = K|\Sigma| + (|\Sigma| + 1)^2 \ ,$$

it must contain at least $|\Sigma| + 1$ disjoint intervals of size exactly $|\Sigma| + 1$ that contain only admissible nodes. We apply Lemma 3.18 to the spill of J and conclude that at least one of these $|\Sigma| + 1$ many disjoint intervals must have a safe spill. Within this interval, we find two occurrences of some label, say σ, and therefore a σ-looping subinterval. Summing up: we have found a pure interval $I \subseteq J$ that is σ-looping, and contains at most $|\Sigma|$ nodes, all of which are admissible. Furthermore, the spill Z of I is safe. We will now use the Global Horizontal Transfer Lemma to transfer I into $\mathcal{I}$.

We need to find a twin-pair (u, u') in $\mathcal{I}$ that can be used as a target for the transfer. This twin-pair must be chosen so that requirements (i) - (iv) in the Global Horizontal Transfer Lemma are satisfied. The requirement (v) is already satisfied, by assumption on Z being safe. Finally, (vi) is satisfied as the source and the target class both have the same label set Γ.

The first requirement (i) is that the node u' has label σ. Since the label σ was found in a long interval outside $\mathcal{I}$, case (2) must have been used in the construction of $\mathcal{I}_\sigma$. Therefore, we have not only one, but K' pure twin-pairs (u, u') in $\mathcal{I}$ that satisfy the first requirement, each with a different left data value. Furthermore, none of these twin-pairs is in the induced forest of J.

In our case, the value e_1 in the statement of the Global Horizontal Transfer Lemma is simultaneously the data value of all nodes in I. In particular, requirements (ii) and (iii) are trivially satisfied.

The last remaining requirement is (iv), which says that the frontier of Z does not contain any node with the same data value as the left data value e_2 of the target twin-pair (u, u') and that $w \not\sim e_2$. Note that we have K' candidates for e_2 inside $\mathcal{I}_\sigma$. We will show that the frontier of Z is smaller than K', and therefore one of these candidates can be used. Indeed, the reader will recall that the spill of an admissible node has a frontier of size at most L. Since I has at most $|\Sigma|$ nodes, and all are admissible, its spill Z satisfies

$$|\mathrm{Front}(Z)| \leq |\Sigma|L < |\Sigma|L + 1 = K' \ .$$

Having thus satisfied all requirements, we can apply the Global Horizontal Transfer Lemma and move the pure interval I into $\mathcal{I}$. It remains to slightly modify this model so that all the local properties of Section 3.3.3 are satisfied.

During the process above, we only increase the length of one pure interval in $\mathcal{I}$. But this interval was already long by the definition of $\mathcal{I}$, so, for each data value d there are still at most K_1 complete d-pure intervals of length more than L_1. The number of complete pure intervals in each of the sibling groups of the model has not changed during the modification. The only difficulty is with long paths. When we were transferring I between u and u', we may have created new long e_2-paths, by concatenating a path in Z with an e_2-path arriving at the parent of u. In order

to reduce this number we reapply Proposition 3.17 to the class of e_2. We then get the desired model of φ, as this last transformation does not affect the other local properties and also does not introduce new long intervals. □

Few intervals

PROPOSITION 3.21. *There is a model with at most M_2 sibling groups containing more than N_2 complete pure intervals.*

Similar to an attached node, a twin-pair (v, v') with parent u is called **right-attached** if $v' \sim u$ and **left-attached** if $v \sim u$. It is called **attached** if it is left- or right-attached. An interval[6] is said to be **free** if it contains no attached nodes, otherwise it is attached. Free intervals are nice, because they are easy to move around in the model.

The proof of the proposition is shown in two steps.

—Lemma 3.22 gives a model where only a bounded number of sibling groups have a free interval with many complete pure subintervals.

—Lemma 3.23 gives a model where only a bounded number of sibling groups have many attached complete pure intervals.

The statement of the proposition then follows easily.

LEMMA 3.22. *There is a model in which at most M_1' sibling groups contain free intervals with more than N_1' complete pure subintervals.*

PROOF. Let us define the following constants:

$$\begin{aligned} N_1' &= \max\{L_2, 17|\Sigma| + 1\}\,, \\ M_3' &= (N_1' + 2)|\Sigma| + 1\,, \\ M_4' &= M_3' + |\Sigma|(3N_1' + 3) + 1\,, \\ M_1' &= M_4' K_2\ . \end{aligned}$$

We start with a model that satisfies all local properties of Section 3.3.3, and also the small pure interval property assured by Proposition 3.20.

We call a sibling group **bad** if it contains a free interval with more than N_1' complete pure subintervals. We want to limit the number of bad sibling groups to at most M_1'. We will ensure this by transforming the tree so that at most M_4' different data values occur at parents of bad sibling groups. Since we only consider models satisfying all the local properties of Section 3.3.3, we will then have at most $M_4' K_2 = M_1'$ sibling groups with more than $L_2 \leq N_1'$ complete pure intervals at all, which shows the claim.

We order bad sibling groups similarly as in Lemma 3.15, by letting $B \leq C$ if either the depth of B is larger than that of C, or they have the same depth and B has at most as many borders as C. We show below that if there are more than M_4' data values used for parents of bad sibling groups, then we can use Lemma 3.11 to move some interval J with borders (together with its descendants) from a bad sibling group B to a bad one C with $B \leq C$. (Note that we use the Local, and not

[6]Note that in this part of the section intervals are not always pure.

Global, Horizontal Transfer Lemma here.) For termination we argue in a similar way as for Lemma 3.15. We use again vectors W_1, W_2, where the m-th entry of W_1 is the number of nodes at depth m in the tree, and the m-th entry of W_2 is the number of bad sibling groups with m borders. Again, by each transformation step either W_1 increases or W_1 is unchanged and W_2 increases. Therefore at some point we arrive at a tree where at most M_4' data values are used for parents of bad sibling groups.

So assume that the tree contains at least M_4' bad sibling groups $B_1, \ldots, B_n$ with *distinct parent data values*; these are ordered so that $B_i \leq B_j$ for all $i < j$. In each B_i we fix a free interval I_i with exactly N_1' complete pure subintervals. As $N_1' \geq 17|\Sigma|$, in each I_i we can find some σ-border that occurs at least 17 times; we will refer to this σ as σ_i.

We will consider 3 cases, depending on the number of intervals I_i in which σ_i-borders occur frequently with pairwise distinct left (right, resp.) data values.

Assume first that more than M_3' of the intervals I_i are such that they contain at least 5 σ_i-borders with pairwise distinct left data values.

As $M_3' > (N_1' + 2)|\Sigma|$, we can fix a label σ such that more than $N_1' + 2$ of the intervals I_i have at least 5 σ-borders with distinct left data values. Among these intervals we can find some $i < j$ such that the parent data value of I_j does not occur in I_i. We can find such i, j by taking i minimal (with $\sigma_i = \sigma$), and using the fact that I_i contains exactly N_1' complete pure intervals, and that parent values are pairwise different. Together with I_i being a free interval, we conclude that each subinterval J of I_i and each σ-border (u, u') of I_j satisfy condition (iv) of Lemma 3.11. If J is chosen to be σ-looping, then condition (i) is also satisfied. Furthermore, since σ is a border, we have $u \not\sim u'$ and therefore condition (ii) is satisfied, too. It remains to chose J and (u, u') so that condition (iii) is satisfied. For this, we use the assumption on σ occurring 5 times with pairwise distinct left data values, and argue as in the proof of Lemma 3.14. Thus, the Local Horizontal Transfer Lemma can be applied to J and (u, u').

The second case is the symmetric case where more than M_3' of the intervals I_i contain at least 5 σ_i-borders occurs with pairwise distinct right data values.

The remaining case is where in all but M_3' intervals I_i, σ_i-borders occur with at most 4 distinct left data values and at most 4 distinct right data values. As each of the intervals I_i has at least 17 σ_i-borders, each one must have two data-equivalent σ_i-borders. As $M_4' > M_3' + |\Sigma|(3N_1' + 3)$, there is a label σ such that more than $3N_1' + 3$ of the intervals satisfy $\sigma_i = \sigma$. We can thus find $i_1 < i_2 < i_3 < i_4$ such that

—each of $I_{i_1}, \ldots, I_{i_4}$ has 2 data-equivalent σ-borders, and
—for each $j < k$, the parent value of I_{i_k} does not occur in I_{i_j}.

We can argue as in the proof of Lemma 3.15 and find an subinterval J in I_{i_j} and a σ-border (u, u') in I_{i_k}, for some $j < k$, such that the Local Horizontal Transfer Lemma can be applied. Note that condition (iv) is ensured by the second item above, together with I_{i_j} being free.

In all cases we did not change any data value, did not modify any complete pure interval, and we only increased the size of such sibling groups whose number of complete pure intervals was already bigger than L_2. Furthermore, we only trans-

ferred free intervals, so no new pure vertical paths were created. Thus all the local properties obtained in Section 3.3.3 and the property obtained in Proposition 3.21 still hold. □

LEMMA 3.23. *There is a model with at most M_2' sibling groups that contain more than N_2' attached complete pure intervals.*

PROOF. The proof is very similar to the proof of Proposition 3.20.

Let $\mathcal{B}_0$ be the set of sibling groups containing either 1) pure intervals of length more than N_1, or 2) free intervals with more than N_1' complete pure subintervals. Thanks to Proposition 3.20 and Lemma 3.22, we may assume that $\mathcal{B}_0$ contains at most $M_1 + M_1'$ sibling groups. In the proof, we will not modify sibling groups from $\mathcal{B}_0$. This will allow us to assume that all pure intervals are small and all free intervals have few complete pure subintervals.

We define the following constants (where K is defined in Eq. $(*)$):

$$\begin{aligned} L' &= |\Sigma| N_1' \ , \\ K' &= N_1 L' L + 1 \ , \\ M_2'' &= M_1 + M_1' + K' K_2 |\Sigma| \ , \\ M_2' &= 2^{|\Sigma|} M_2'' \ , \\ N_2' &= |\Sigma|(|\Sigma|+1)(|\Sigma|+K) \ . \end{aligned}$$

In the following, we say a sibling group is **bad**, if it contains more than N_2' attached complete pure intervals. The lemma will be established once we limit the number of bad sibling group to M_2'.

We say a label σ is **attached** if it implies that a node has the same value as its parent. A **Γ-value** is a data value whose class contains exactly the labels from Γ. Similarly as in the proof of Proposition 3.20, we proceed separately, for each set $\Gamma \subseteq \Sigma$. Let thus Γ be fixed.

For each attached label σ, we will again construct a set $\mathcal{I}_\sigma$ which will be used as targets for transfers. As before, we consider two cases, depending on σ:

(1) Assume first that at most K' Γ-values occur in parents of bad sibling groups with label σ. In this case, we put all bad sibling groups that contain label σ into $\mathcal{I}_\sigma$. Since we only consider models satisfying the local properties of Section 3.3.3, there can be at most $K'K_2$ such sibling groups.

(2) Otherwise, we choose a set $\mathcal{I}_\sigma$ of K' bad sibling groups that contain nodes with label σ, with K' different parent Γ-values. These are chosen by traversing the tree in a root-to-leaf, left-to-right manner, and putting into $\mathcal{I}_\sigma$ the first K' bad sibling groups that contain label σ and have distinct parent Γ-values. In particular, for each bad sibling group $B \notin \mathcal{I}_\sigma$ with parent Γ-value and that contains σ, we have that either a) the induced forest of I does not contain any sibling groups from $\mathcal{I}_\sigma$; or b) there exists some $B' \in \mathcal{I}_\sigma$ with the same parent value as B, which does not belong to the induced forest of B.

Let $\mathcal{I}$ be the union of the sets $\mathcal{I}_\sigma$, for all labels σ. Clearly $\mathcal{I}$ contains at most $K'K_2|\Sigma|$ sibling groups, and therefore $\mathcal{I} \cup \mathcal{B}_0$ contains at most M_2'' sibling groups. We will show that if B is a bad sibling group outside $\mathcal{I} \cup \mathcal{B}_0$, then part of B can be

transferred into $\mathcal{I}$, possibly with a change of data value. By iterating this process, we eventually obtain a model satisfying the statement of the lemma.

Let thus B be a bad sibling group outside $\mathcal{I} \cup \mathcal{B}_0$ with a Γ-value d in the parent of B. This is also the data value of all nodes in B with an attached label. Since B is bad, there must be some attached label σ, such that B contains more than $(|\Sigma| + K)$ complete σ-looping intervals. At least one of these intervals, which we will denote by I, has a safe d-spill (by Lemma 3.18) and contains only d-admissible nodes (by Eq. $(*)$). We may well assume that I contains at most $|\Sigma|$ complete pure attached intervals, since otherwise we could find a different looping complete interval inside I. Furthermore, by assumption on $B \notin \mathcal{B}_0$, between every two attached intervals in B we can find at most N_1' consecutive free complete pure intervals. In particular, I contains at most $L' = |\Sigma|N_1'$ complete pure intervals altogether, free or attached.

Summing up: inside the sibling group B we have found a complete σ-looping interval I with at most L' complete pure intervals and a safe d-spill. Furthermore, all nodes in I that have data value d are admissible. Finally, by assumption on $B \notin \mathcal{B}_0$, each pure interval in I has length at most N_1.

We will now apply the Global Horizontal Transfer Lemma (Lemma 3.19) and transfer I into a twin pair (u, u') inside $\mathcal{I}$. Note that, as σ is attached, the right elements of the interfaces carry the same data value as required by the Global Horizontal Transfer Lemma. We need to choose (u, u') so that that all requirements (i) - (v) of the lemma are satisfied. By assumption on the d-spill of I being safe, the requirement (v) is already satisfied. As the source and target class are Γ-classes, (vi) is also guaranteed.

The first requirement (i) is that the node u has label σ. Since the label σ was found in a sibling group outside $\mathcal{I}$, case (2) must have been used in the construction of $\mathcal{I}_\sigma$. Therefore, we have not only one, but K' twin-pairs (u, u') in $\mathcal{I}$ that satisfy the first requirement, each with a different left data value.

If at least one of these twin-pairs (u, u') is in the induced forest of I, then we know that there is some $B' \in \mathcal{I}_\sigma$ whose parent data value is d and which does not belong to the induced forest of I. We can therefore use the Local Horizontal Transfer Lemma to move the interval I into B'. Therefore, from now on we assume that the K' twin-pairs (u, u') in $\mathcal{I}$ that satisfy the first requirement are all outside the induced forest of I. We will try to pick one among these candidates that satisfies the remaining requirements (ii), (iii) and (iv).

The requirement (iii) is satisfied, since by assumption on σ being an attached label, both d_1 and e_1 (as defined in the Global Horizontal Transfer Lemma) are the same data value d.

It remains to satisfy the requirements (ii) and (iv). Note that in our case, each candidate (u, u') for the target twin-pair is right-attached – since σ is an attached label – and therefore both d_2 and e_2 (as defined in the lemma) are the same data value. The requirements (ii) and (iv) will therefore be satisfied if we can find a candidate for the data value $d_2 = e_2$ that does not occur in the interval I itself, nor in the frontier of the d-spill of I. By our assumptions, I contains at most $N_1 L'$ nodes, and therefore at most as many data values. Furthermore, recall that the spill of an admissible node has a frontier of size at most L. In particular, since all

nodes in I with data value d are admissible, the frontier of the d-spill of I contains at most $N_1L'L$ nodes. Summing up, there are at most $N_1L'L$ data values that need to be avoided by the twin pair (u, u'). Since $\mathcal{I}$ contains at least

$$K' = N_1L'L + 1$$

candidates, at least one of them is suitable, and the Global Horizontal Transfer Lemma can be applied to reduce the size of B.

It might be necessary to modify this model slightly in order to maintain all the local properties of Section 3.3.3 and the property of Proposition 3.20.

Let d' be the right value $d_2 = e_2$ of the target twin pair (u, u'). During the process above, the class of d' has been modified. It certainly does not have more d'-parent sibling groups with more than L_2 complete pure intervals, as only sibling groups containing many complete pure intervals have been modified. But some long d'-intervals and d'-paths may have been created while changing the data value d into d'. We first apply Proposition 3.12 and Proposition 3.17 to the data value d'. This does not affect d'-parent sibling groups with many pure subintervals, and therefore the new model satisfies all the properties of Section 3.3.3. But this may have introduced globally new long pure intervals. We can then apply again Proposition 3.20 and we get the desired model as Proposition 3.20 does not affect the number of complete pure intervals inside a sibling group. □

Proof of Proposition 3.21. We set $M_2 = M_1' + M_2'$ and $N_2 = N_2'(N_1' + 1)$ and conclude the proposition directly from Lemmas 3.22 and 3.23. □

Shallow zones. Analogously to the local case, we first give a vertical transfer lemma.

The **innerspill** of a path that goes from a node v to its descendant w is defined to be the spill of v without the spill of w.

Lemma 3.24 Global Vertical Transfer Lemma.
Let $d_1 \neq d_2$ be data values and let v, w be two nodes such that w is a descendant of v through a d_1-path p and let u be a node not in $Sub(v, w)$ with data value d_2. Let Z be the innerspill of p.

Then the tree obtained from t by changing the data value of Z from d_1 to d_2 and moving $Sub(v, w)$ between u and its parent is also a model if the following conditions hold.

(i) The labels of v, w, u are the same.
(ii) $\mathrm{Front}(Z)$ does not contain any node with data value d_2.
(iii) Z is safe.
(iv) All labels occurring in Z also occur in the class of d_2.

Proof. As in the proof of Lemma 3.16 it is easy to see that the consistency of neighboring labels is preserved. Condition (iii) ensures that simple conditions of kind (b) and (c) from the automata data normal form are not violated. Finally condition (ii) ensures that the new data tree is consistently profiled. □

Proposition 3.25. *There is a model where at most M_3 zones contain paths with more than N_3 nodes.*

PROOF. We start with a model t that satisfies the conditions of Propositions 3.20 and 3.21.

We define the following constants (where K is defined by Eq. $(*)$):

$$\begin{aligned} K' &= L|\Sigma| + 1 \ , \\ N_3 &= \max\{(K + |\Sigma| + 1)(|\Sigma| + 1), L_3\} \ , \\ M_3' &= |\Sigma|K'K_3 \ , \\ M_3 &= 2^{|\Sigma|}M_3' \ . \end{aligned}$$

The general idea is that if a data path is too long we can find a subpath in it to which we can apply the Global Vertical Transfer Lemma.

As before, we transform the tree in several rounds, one for each $\Gamma \subseteq \Sigma$. We fix Γ. For each label σ, we define a set $\mathcal{P}_\sigma$ of nodes with label σ, which will be used as a target for transfers. As before, we consider two cases:

(1) Consider first the case when at most K' different Γ-classes contain σ. In this case, we define $\mathcal{P}_\sigma$ to be the set of all nodes with label σ that are on some data path of length more than N_3. Since $N_3 \geq L_3$, by Proposition 3.17 we know that $\mathcal{P}_\sigma$ contains nodes from at most $K'K_3$ zones.

(2) Otherwise, we set $\mathcal{P}_\sigma$ to be any set of K' nodes that have label σ and pairwise different Γ-values.

Let $\mathcal{Z}$ be the set of all zones that contain a node from one of the sets $\mathcal{P}_\sigma$. Clearly the size of $\mathcal{Z}$ is at most $M_3' = |\Sigma|K'K_3$. We will now show that if a data path p has more than N_3 nodes, then some fragment of p can be transferred into a zone from $\mathcal{Z}$. By iterating this process, we arrive at the statement of the lemma.

Let thus p be a data path of a Γ-value d_1, disjoint from $\mathcal{Z}$ that contains more than N_3 nodes. Recall that there are at most K non-admissible nodes in any antichain of the class of d_1 (cf. Eq. $(*)$). Since p contains at least

$$N_3 \geq (K + |\Sigma| + 1)(|\Sigma| + 1)$$

nodes, there must be two nodes in v, w in p that have the same label, say σ, and such that innerspill Z between v and w is safe and contains only admissible nodes (except maybe for the nodes of p). Without loss of generality we may assume that the path from v to w contains at most $|\Sigma|$ nodes.

By assumptions on v, w concerning admissible nodes and the length of the path from v to w we have:

$$\text{Front}(Z) \leq L|\Sigma| < K'$$

Note that since the label σ was found outside $\mathcal{Z}$, case (2) must have been used when constructing $\mathcal{P}_\sigma$. In particular, $\mathcal{P}_\sigma$ must contain at least one node u with label σ and a Γ-value d_2 not occurring in $\text{Front}(Z)$.

We can now apply the Global Vertical Transfer Lemma and transfer the subtree $\text{Sub}(v, w)$ between u and its parent, changing the data value d_1 into d_2 in the innerspill Z.

Note that after the transfer, we augmented neither the number of complete pure intervals, nor the number complete pure intervals in any sibling group, thus preserving Propositions 3.20 and 3.21. □

3.4 Satisfiability for Small Models

In this section we show the following proposition:

PROPOSITION 3.26. *Given M, N, and a sentence φ in automata data normal form, it is decidable whether φ has a model in which at most M data zones have outdegree more than N.*

The proof is by reduction to the non-emptiness of linear constraint tree automata. In Subsection 3.4.1, we define linear constraint tree automata and show that they have a decidable emptiness problem. Then, in Section 3.4.2, we show that the (data erasure of) solutions can be recognized by a linear constraint tree automaton.

3.4.1 *Linear constraint tree automata.* A **linear inequality over variable set** X is an expression of the form

$$\sum_{x \in X} k_x \cdot x \quad \geq \quad 0 \qquad k_x \in \mathbb{Z} \; .$$

A **linear constraint** over X is a Boolean combination of linear inequalities. A valuation $\nu : X \to \mathbb{N}$ satisfying a linear constraint is defined in the usual way. We call such a satisfying valuation a **solution**.

Definition 3.27. A **linear constraint tree automaton (LCTA)** is a nondeterministic unranked tree automaton $\mathcal{A}$ with state space Q, together with a linear constraint over Q. The LCTA accepts a tree if the tree admits a run $\rho : V \to Q$ of $\mathcal{A}$ on t, which accepts in the usual sense, and which moreover satisfies the linear constraint wrt. its Parikh image $(|\rho^{-1}(q)|)_{q \in Q}$.

The following result can be obtained by using [Neeraj Verma et al. 2005], where it is shown how to compute in linear time an existential Presburger formula for the Parikh image of a context-free language described by a grammar (a similar result was also obtained in [Fan and Libkin 2002]). The proof in [Neeraj Verma et al. 2005] can be directly adapted to extended context-free grammars, i.e., grammars with rules $A \to L_A$, where $L_A \subseteq \Sigma^*$ is a regular language giving the possible right-hand sides of the rule with left-hand side A.

THEOREM 3.28. *Non-emptiness of LCTA is in* NPTIME.

PROOF. The general idea is that a tree is similar to a derivation of an extended context-free grammar, therefore emptiness of LCTA can be resolved by calculating the Parikh image of an extended context-free grammar.

Let $\mathcal{A}$ be a LCTA with state space Q, initial state sets I and J, transition relations δ_v, δ_h, acceptance condition $\mathcal{F} \subseteq Q \times \Sigma$ and a linear constraint E. We transform $\mathcal{A}$ into an extended context-free grammar G of polynomial size such that $L(\mathcal{A})$ is non-empty if and only if the Parikh image of $L(G)$ has non-empty intersection with the linear constraint E. The claim then follows using [Neeraj Verma et al. 2005; Papadimitriou 1981], since the Parikh image of $L(G)$ can be expressed by a linear-size existential Presburger formula, hence also the intersection with E.

The grammar G describes accepting runs of the underlying tree automaton of $\mathcal{A}$ as follows. The nonterminals are states Q of the LCTA, while the terminals are a disjoint copy $\bar{Q} = \{\bar{q} \mid q \in Q\}$ of the states. For each nonterminal q, G allows a

derivation step $q \to \bar{q}q_1 \cdots q_n$ if: 1) the state q_1 belongs to J; and 2) for each $i < n$ there is some label $a \in \Sigma$ such that (q_i, a, q_{i+1}) belongs to δ_h; and 3) there is some label $a \in \Sigma$ such that (q_n, a, q) belongs to δ_v. Finally, if q belongs to I, we add a rule $q \to \bar{q}$. The starting nonterminals are those states $q \in Q$ such that (q, a) is final for some label $a \in \Sigma$. It is fairly straightforward that the Parikh image of $L(G)$ is exactly the Parikh image of the set of accepting runs of the underlying tree automaton of $\mathcal{A}$. □

It is easy to see that tree languages recognized by LCTA enjoy the following closure properties:

LEMMA 3.29. *The class of unranked tree languages accepted by LCTA is closed under conjunction, disjunction and renaming*[7].

Note that it is important here that the linear constraints speak of states and not of letters of the input. Even over words, an automaton with linear constraints over letters in the input cannot recognize the language $\{b^*a^nb^n \mid n \in \mathbb{N}\}$.

3.4.2 *Satisfiability via linear constraint tree automata.* The following proposition shows that when restricted to data trees with small outdegree of data zones, LCTA can recognize the data erasure of $\mathrm{FO}^2(\sim, +1)$ sentences. The difficult step of the proof of this proposition is transforming an accepting run of the LCTA into a data tree which is a model of the sentence. For this, as it will be made clear below, we need the assumption that only few zones have large outdegree.

PROPOSITION 3.30. *Let φ be a sentence in automata data normal form. Let $M, N \in \mathbb{N}$. There is a LCTA that recognizes the data erasures of those models of φ where at most M zones have outdegree more than N.*

The proof of this proposition is given in this subsection.

Let us fix a sentence φ in automata data normal form. As in the previous section, Σ is the set of labels (types that determine the value of every unary predicate).

We distinguish two kinds of labels σ depending of φ. Those that can appear at most once in a class are called **dogs**, and those that can appear arbitrarily many times in a class are called **sheep**. Whether or not a label is a dog or sheep is determined by the simple formulas of kind (b) in Def. 3.2 that appear in φ. By making use of one more relation R_D in φ we can ensure that each label β arising from a formula $\forall x \exists y\ \alpha(x) \to (x \sim y \wedge \beta(y))$ of type (c) in Def. 3.2 is a dog label. Notice that adding the predicate R_D does not affect the normal form.

Consider a class in a data tree satisfying φ. The **pre-class-type** of the class is a pair (D, S) where D is the set of dog labels occurring in the class and S is the set of sheep labels occurring in class. All possible pre-class-types can be computed from φ using the constraints of type (b) and (c).

It will be important in the construction of the LCTA below to know how the dog labels of each class are spread among the zones of the class. For this we define the **class-type** of a class as the triple (π, D, S) where (D, S) is a pre-class-type and π is a partition of D that separates two dog labels if and only if they appear in different zones of the class.

[7] A renaming function is one of the form $h : \Sigma \to \Sigma'$.

The number of class-types is doubly exponential in n, where n is the number of unary predicates in φ. Indeed $|\Sigma| = 2^n$, and thus there are at most $2^{O(2^n)}$ pairs (D, S). For each such pair there are $2^{O(|D|)} = 2^{O(2^n)}$ ways of partitioning D.

Let $k = M + N|\Sigma| + 4$.

Let ν be a function that assigns to every class-type τ a number from $\{0, \ldots, k\}$. For each such function ν, we define $L(\varphi, \nu)$ to be the set of models t of φ with at most M zones of outdegree greater than N, where for each class-type τ the following holds:

—If $\nu(\tau) < k$, then there are exactly $\nu(\tau)$ classes in t of class-type τ.

—If $\nu(\tau) = k$, then there are at least k classes in t of class-type τ.

Proposition 3.30 follows immediately from the following lemma, by taking a disjunction of automata over all possible functions ν.

LEMMA 3.31. *For every $M, N \in \mathbb{N}$, for every sentence φ in automata data normal form, and every function ν, there is an LCTA $\mathcal{A}_{\varphi,\nu}$ that recognizes the data erasure of $L(\varphi, \nu)$. The size of $\mathcal{A}_{\varphi,\nu}$ is doubly exponential in $|\varphi|$ and polynomial in M and N.*

The rest of this subsection is devoted to showing Lemma 3.31.

We fix M, N, φ and ν for the rest of the subsection. Let t be a data tree in $L(\varphi, \nu)$. A class (data value) in t is called **special** if it has a zone with outdegree more than N, or if its class-type τ is such that $\nu(\tau) < k$. Note that the total number of special data values is at most $M + mk$, where m is the number of class-types. We assume without loss of generality that these data values are $\{1, \ldots, M + mk\}$.

We begin by describing the idea behind the LCTA $\mathcal{A}_{\varphi,\nu}$. It works on a profiled tree without data and tries to verify if the tree can be expanded with data values so that it belongs to $L(\varphi, \nu)$. The tree already contains most of the information: the node labels (that include the profiles) and the division into pseudo-zones. Here, a **pseudo-zone** is a maximal (connected) set of nodes whose labels indicate that the nodes should have the same data value. The whole point of $\mathcal{A}_{\varphi,\nu}$ is to verify if the pseudo-zones can be grouped into classes that are consistent with the possible class types allowed by φ.

It should be noted that $\mathcal{A}_{\varphi,\nu}$ can easily check whether a profiled tree without data is locally consistent, i.e., whether the profile conditions can be satisfied: e.g., if u is the left sibling of v and the profiles of u and v say that both have the same data value as their parent, then the profile of u must say that it has the same data value as its right sibling.

The LCTA $\mathcal{A}_{\varphi,\nu}$ is defined as follows. For each node, it guesses the class-type of the class of this node and whether this class will have a special data value, and if so, which one. We call this value the **prevaluation** of a node, which is either one of $\{1, \ldots, M + mk\}$ or undetermined. We call the values in $\{1, \ldots, M + mk\}$ **special**.

Naturally, all the guesses are done consistently with the pseudo-zones, i.e. all nodes in the same pseudo-zone get the same special value, if any. Then the LCTA $\mathcal{A}_{\varphi,\nu}$ verifies that:

(1) Nodes in the same zone get the same class-type.

(2) The labels in the tree satisfy the neighboring label conditions given by the automaton formulas of φ.

(3) No two adjacent pseudo-zones have the same determined prevaluation. (They may both have undetermined data values.)

(4) All pseudo-zones of outdegree more than N have a special value. Likewise for nodes whose class-type τ satisfies $\nu(\tau) < k$.

(5) The set of dog labels in every pseudo-zone of class-type τ is consistent with the partition π induced by τ.

(6) For each τ, any two labels a, b in the set D_τ of dog labels of τ occur the same number of times in nodes of class type τ.

Note that the linear constraints are only used in the last property. Clearly the automaton $\mathcal{A}_{\varphi,\nu}$ accepts all trees in $L(\varphi, \nu)$.

We now proceed to show the converse, i.e., that for any profiled tree t without data values accepted by the LCTA $\mathcal{A}_{\varphi,\nu}$ there is a valuation of t, i.e., a mapping from nodes to data values such that the resulting data tree t' is in $L(\varphi, \nu)$. The difficulty is to find data values for the nodes with undetermined prevaluation so that any two adjacent pseudo-zones get different data values. We prove that this is indeed possible by taking advantage of the fact that there are at most M zones of outdegree larger than N.

Let us fix a run ρ of $\mathcal{A}_{\varphi,\nu}$ (which tells us the prevaluation λ). A **consistent valuation** μ of t using ρ is a labeling of its nodes with data values, that is consistent with λ and ρ in the following ways:

(A) Every pseudo-zone has exactly one data value.

(B) If $\lambda(v)$ is special then $\mu(v) = \lambda(v)$.

(C) For every data value d, the class-type of d is the one guessed in ρ.

(D) Adjacent pseudo-zones have different values.

For a tree t that is accepted by $\mathcal{A}_{\varphi,\nu}$ it is straightforward to find a valuation for t that fulfills conditions (A)-(C). One only needs to assign data values to nodes where ρ did not supply a guessed data value. This can be done while satisfying (C) thanks to the linear constraints in $\mathcal{A}_{\varphi,\nu}$.

Therefore, it only remains to prove that condition (D) can be fulfilled as well.

To this end, we start with a valuation μ_0 of t fulfilling (A)-(C) and then modify the valuation, if necessary, in a top-down fashion. We will thereby ensure that condition (D) is satisfied for progressively more pseudo-zones. For this purpose, we call a set V of nodes of t **initial**, if it is a union of pseudo-zones and it is closed under the ancestor relation. We say that a valuation of t is **consistent for V** if it fulfills (A)-(C) and there are no two adjacent pseudo-zones in V that have the same data value. We will prove the following claim:

Claim. For every initial set V there is a consistent valuation for V.

The proof is by induction on the size of V. Clearly, μ_0 is a consistent valuation for V if V consists only of the topmost pseudo-zone.

Now we consider an initial set V with at least one pseudo-zone. Let Z be a pseudo-zone outside V, chosen so that its root is of minimal depth. This ensures

that Z has at most 3 adjacent pseudo-zones in V, say Z_1, Z_2, Z_3. Let $W = V \cup Z$. Note that W is again initial.

Let us fix a valuation μ of t that is consistent for W, obtained by induction. If μ is also consistent for V we are done. Otherwise, the pseudo-zone Z has the same data value as one of Z_1, Z_2, or Z_3. We will remove this conflict by swapping the data value of Z with some other data value.

Clearly in this case, the value of Z is not special. In particular, the class type τ of Z satisfies $\nu(\tau) = k$, since all other class types get special values. For the same reason, no pseudo-zone with the data value of Z has outdegree more than N.

Let Δ denote all the data values used in μ that are not special and are used for nodes of class type τ. Since $\nu(\tau) = k$, we have $|\Delta| \geq k - M$.

Consider first the case when the pseudo-zone Z only has sheep labels. In this case, we can assign to the nodes of Z any of the data values from Δ and still have a valuation which is consistent for W. Since $|\Delta| > 3$, this can be done so that Z has a different data value than Z_1, Z_2 and Z_3. We then get a valuation consistent for V.

The difficult case is when Z contains dog labels. Let $\mathcal{Z}$ be the set of pseudo-zones in t that have the same dog labels as Z and a data value in Δ. Since dog labels are those that occur precisely once in a class, we have that $|\mathcal{Z}| = |\Delta|$. We want to exchange the data value in Z with the data value of some pseudo-zone in $\mathcal{Z}$.

Consider first a pseudo-zone Y with the same data value as Z that contains no dogs. This pseudo-zone Y is not special, and therefore has at most N neighbors in t. Because $|\Delta| > N$, there is a data value in Δ which is distinct from all the data values of the neighbors of Y. We can then assign this data value to Y and still get a valuation consistent for W. Therefore, we can assume that there are at most $|\Sigma|$ pseudo-zones with the data value of Z (there are at most $|\Sigma|$ dog-labels). Each such zone has at most N neighbors and therefore there are at most $N|\Sigma|$ pseudo-zones adjacent to a zone having the same data value as Z. Since

$$|\mathcal{Z}| \geq k - M = N|\Sigma| + 4 \ ,$$

there is at least one pseudo-zone $Z' \in \mathcal{Z}$ that has a data value distinct from Z_1, Z_2 and Z_3 and that is not adjacent to any pseudo-zone with the same data value as Z. We can thus pick such a pseudo-zone Z' and exchange the data values of Z and Z'. We obtain thus a valuation that is consistent for V, completing the proof of the claim.

We now comment on the size of $\mathcal{A}_{\varphi,\nu}$. The automaton can easily verify the properties (1)-(3), and (5), with a number of states polynomial in the number of class-types[8] M and N. Property (6) is the linear constraint. It remains to show that property (4) can be tested while keeping the size polynomial. In particular, we need to check whether a pseudo-zone has outdegree larger than N. To do this, consider a node x in a pseudo-zone Z. It is easy to count the number $A(x)$ of different pseudo-zones $Z' \neq Z$ that contain some child y of x, by inspecting the children of x from left to right and incrementing when the profiles tell so. Actually we only need to count up to N (if the counter exceeds N, then Z has

[8]The reader should be aware though, that the number of class-types is doubly-exponential in the size of $|\varphi|$).

already outdegree larger than N and we can stop). Using $A(x)$, we can compute the number $B(x)$ of pseudo-zones that are adjacent to Z and with topmost level below x. Namely, $B(x)$ equals $A(x)$, plus $A(x')$ for all children x' of x that belong to Z, too (note that no pseudo-zone is counted twice). To finish, we consider the nodes of Z that belong to the topmost level. These nodes form an interval I, and we need to add to the sum of the $B(x)$ with $x \in I$, 0, 1 or 2, depending on the left/right neighbor of I and their profiles towards their parent.

This completes the proof of Lemma 3.31 and thus of Proposition 3.30. Furthermore, it completes the proof of Theorem 3.1.

4. A LOWER BOUND FOR $\mathbf{FO}^2(\sim, <, +1)$

We have shown that satisfiability is decidable for $\mathrm{FO}^2(\sim, +1)$. What happens when we also add the descendant order $<$?

In this section we show that satisfiability of $\mathrm{FO}^2(\sim, <, +1)$ on (even binary) trees is at least as hard as checking non-emptiness for vector addition tree automata. The decidability of the latter has been an open problem for many years and is, in turn, equivalent to a notorious open problem in linear logic, the decidability of MELL (Multiplicative Exponential Linear Logic) (see [de Groote et al. 2004] and the references therein). Therefore proving decidability of $\mathrm{FO}^2(\sim, <, +1)$ on trees seems to be quite challenging.

A vector addition tree automaton over binary trees is a bottom-up tree automaton that is additionally equipped with a finite number of counters. Each of these counters carries a non-negative number. A run of the automaton assigns to every node, besides a state, a counter assignment, which is a vector of non-negative integers. The run has to be consistent with transitions of the automaton, which are described below.

Each transition of the automaton is parametrized by three vectors $\vec{a}, \vec{b}, \vec{c}$ (of dimension k, where k is the number of counters), three states p, q, r and a label σ. Assume that v is a node with label σ, and its children v_0, v_1 have been assigned states p, q and counter values $\vec{x}, \vec{y}$ respectively. If the counter values satisfy $\vec{x} \geq \vec{a}$ and $\vec{y} \geq \vec{b}$ then the transition can be applied, and the node v gets state r and the counter assignment

$$(\vec{x} - \vec{a}) + (\vec{y} - \vec{b}) + \vec{c}\,.$$

In other words, the automaton first decrements the counters in the left (resp. right) child by $\vec{a}$ and $\vec{b}$, resp. If both decrements are successful (i.e. the counters are still non-negative), then the counter values are added and further incremented by $\vec{c}$.

More formally a **vector addition tree automaton** A is a tuple

$$(Q, \Sigma, k, F, \delta_0, \delta)\ ,$$

where Q is a finite set of states, Σ is the finite alphabet, k is the number of counters of the automaton, $F \subseteq Q$ is a set of accepting states,

$$\delta \subseteq (Q \times \mathbb{N}^k)^2 \times Q \times \Sigma \times \mathbb{N}^k$$

is the finite set of transitions and $\delta_0 \subseteq Q \times \Sigma \times \mathbb{N}^k$ is the finite set of initial transitions. (The initial transitions are used to give counter assignments to the leaves of the tree, depending on their labels.)

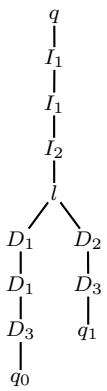


Fig. 7. Coding the transition $(q_0, (2,0,1), q_1, (0,1,1), q, l, (2,1,0))$.

A tree is accepted if it admits a run where the root carries an accepting state and all counters are 0.

Note that the automaton *cannot* test whether a counter is equal to zero, otherwise the model would be immediately undecidable.

THEOREM 4.1. *For any vector addition tree automaton $\mathcal{A}$, a sentence $\varphi_{\mathcal{A}} \in \mathrm{FO}^2(\sim, <, +1)$ can be computed such that $L(\mathcal{A}) \neq \emptyset$ iff $\varphi_{\mathcal{A}}$ has a model.*

PROOF. Let k be the number of counters of $\mathcal{A}$ and Q be its set of states.

The models of $\varphi_{\mathcal{A}}$ are going to represent accepting runs of the automaton $\mathcal{A}$. These runs will be represented by trees, where each node is labeled by either: a letter from the input alphabet Σ, a state Q, or an element of I_i, D_i for $i \in \{1, \ldots, k\}$. The intended meaning of letter I_i is that counter i is *incremented* by one, while D_i means that counter i is *decremented* by one. Nodes in models of $\varphi_{\mathcal{A}}$ will have either zero, one or two children. In order to guarantee that we only decrease a counter wich is not nul, each occurrence of D_i is coupled with some occurrence of I_i, by having the same data value on the 2 nodes. That is, the data value of a node with label I_i will occur exactly one further time, at a node with label D_i (and vice-versa). Data values of nodes labeled by a state or some input letter from Σ will occur exactly once.

In a model of $\varphi_{\mathcal{A}}$, a transition

$$\delta(p, (a_1, \ldots, a_k), q, (b_1, \ldots, b_k), r, \sigma, (c_1, \ldots, c_k))$$

is represented by a fragment of the tree, where, for each $i = 1, \ldots, k$, there are c_i copies of I_i in its top branch, a_i copies of D_i in the left branch and b_i copies of D_i in the right branch, as depicted in Figure 7. The leaf conditions are handled in the same fashion. Transition are combined in the obvious way and that a tree consists of such patterns only, can be easily described in $\mathrm{FO}^2(+1)$.

It is easy to express that each data value occurs either only once (at a node labeled by a state or by Σ) or twice, at a node labeled by some D_i and a descendant of this node, labeled I_i. A formula stating the second property says that no two nodes of label D_i can have the same data value, no two nodes of label I_i can have the same data value, for each node of label D_i there exists a descendant of label I_i with the

same data value and, for each node of label I_i there exists an ancestor of label D_i with the same data value.

Thus, the overall number of decrements of each counter is equal to the number of increments, therefore all counters have value zero at the root. Moreover each decrement is preceded by an increment (below), therefore the value of each counter is always non-negative. This concludes the proof. □

5. INTEGRITY CONSTRAINTS

In this and the following section we show how our main result, Theorem 3.1, can serve as a tool to answer decidability questions for XML in the context of data values. We show how decidability results can be obtained via a reduction to $\mathrm{FO}^2(\sim, +1)$ satisfiability. One of the advantages of this logic-based approach to decidability is the compositionality of logic. This holds especially for $\mathrm{FO}^2(\sim, +1)$, which is closed under all Boolean operations and, as far as satisfiability is concerned even under existential set quantification. It permits to relativize all reasoning problems to documents satisfying schemas. In this section we consider reasoning with XML schemas. XML documents usually come with a specification, often stated in XML Schema, which describes the set of valid documents. It contains a structural part which includes a mechanism for assigning types to nodes of the tree and a set of integrity constraints such as *key constraints* and *inclusion constraints*.

It is natural to ask whether a specification is consistent and whether a set of integrity constraints is minimal or not (*implication problem*).

We will see that it follows quite directly from Theorem 3.1 that the consistency and the implication problem for unary keys and inclusion constraints are decidable, even relative to structural constraints given by a regular tree language.

Note that the decidable results presented here were previously shown in [Fan and Libkin 2002], with a much better complexity, but in the presence of a weaker typing mechanism. The advantage of our technique is the genericity: it immediately applies to any integrity constraint definable to $\mathrm{FO}^2(\sim, +1)$ and in the presence of any typing mechanism induced by regular tree languages.

We first deal with regular tree languages and types.

Recall that $\mathrm{EMSO}^2(\sim, +1)$ is the extension of $\mathrm{FO}^2(\sim, +1)$, where a prefix of existential quantifiers over unary predicates (i.e., set variables) is allowed before an $\mathrm{FO}^2(\sim, +1)$ formula. Similarly, $\mathrm{EMSO}^2(+1)$ is obtained from $\mathrm{FO}^2(+1)$ by prefixing existential existential set quantification.

Basically, the two standard XML schema languages, DTD and XML Schema, are able to define only sets of documents that are regular tree languages (but not all regular tree languages!). In the following, we thus assume that the allowed set of documents is described by a tree automaton A. The **type** of a node v is the state of A on v in an accepting run. In XML Schema it is basically required that A has a unique accepting run, thus the type of each node is uniquely determined. Thus, in the following we only consider unambiguous tree automata. Recall from Fact 3.6 that every regular tree language is expressible in $\mathrm{EMSO}^2(+1)$.

A **key constraint** is an expression of the form $\tau[X] \to \tau$ where τ is a type of a node and X a set of attributes of that node. It says that the X-attributes of a node of type τ uniquely determine the node. Stated in other terms, for each combination

```
<schedule>
 <course ID="5">
   <lecturer faculty="12">  </lecturer>
   <building nr="1"> </building>
 </course>
</schedule>
```

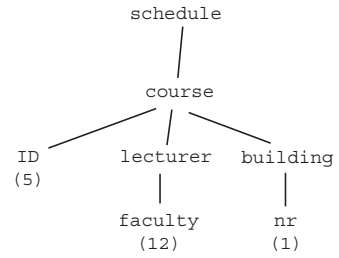


Fig. 8. An XML document and its data tree encoding. In the encoding, data values are in parentheses. Data values for non-attribute nodes are not used.

of attribute values there is at most one node of type τ having these values.

An **inclusion constraint** is an expression of the form $\tau[X] \subseteq \tau'[Y]$ where τ and τ' are two node types and X and Y are sequences of attributes of the same cardinality. It says that for each node u of type τ there is a node v of type τ' such that the X-attributes of u have the same (corresponding) values as the Y-attributes of v. Key and inclusion constraints are said to be **unary** if $|X| = |Y| = 1$.

The **consistency problem** for unary keys and unary inclusion constraints relative to a regular tree language is as follows. Given an (unambiguous) tree automaton A and a set K of unary key and inclusion constraints[9], it asks whether there is a tree t which is accepted by A and fulfills K. The **implication problem** asks, given A and sets K_1, K_2 of constraints, whether each tree accepted by A which fulfills K_1 also fulfills K_2.

PROPOSITION 5.1. *The consistency and implication problems for unary keys and unary inclusion constraints relative to a regular tree language are decidable.*

PROOF. We only consider the more general, implication problem. We encode XML documents as trees in a way which closely corresponds to the XPath data model [W3C 1999], i.e., the attributes of a node v are represented by attribute nodes (labeled by the attribute name) which are children of v. I.e., the B-attribute value of a node v is given by the value of its (unique) child labeled with B. An example of this encoding is presented in Figure 8.

Let A and K_1, K_2 be given. Let $\tau_1, \ldots, \tau_n$ be the types assigned by A. Let $R_1, \ldots, R_n$ be unary predicates, corresponding to the types. As in Fact 3.6, we can write a formula φ_A of $\mathrm{FO}^2(+1)$ that verifies if a given partition of tree nodes among the predicates $R_1, \ldots, R_n$ encodes an accepting run of the automaton A. (In particular, $\exists R_1, \ldots, R_n \varphi_A$ holds in a tree t if and only if t is accepted by A; and a tree node satisfies the predicate R_i if and only if the node has type τ_i). Thus,

[9]Recall that the types used in these constraints are states of A.

a unary key constraint $U : \tau_i[B] \to \tau_i$ can be expressed by the $\mathrm{FO}^2(\sim, +1)$ sentence $\varphi_U =$

$$\forall x \forall y \left(\begin{array}{c} (B(x) \wedge \exists y R_i(y) \wedge E_\downarrow(x,y)) \wedge \\ (B(y) \wedge \exists x R_i(x) \wedge E_\downarrow(y,x)) \wedge \\ x \sim y \end{array} \right) \to x = y \ .$$

An inclusion constraint $U : \tau_i[B_i] \subseteq \tau_j[B_j]$ can be expressed by the $\mathrm{FO}^2(\sim, +1)$ sentence $\varphi_U =$

$$\begin{array}{l} \forall x \ (B_i(x) \wedge \exists y (R_i(y) \wedge E_\downarrow(x,y))) \to \\ \quad \exists y \, (x \sim y \wedge B_j(y) \wedge \exists x (R_j(x) \wedge E_\downarrow(y,x))). \end{array}$$

Thus, the implication problem reduces to (un)satisfiability of the following sentence of $\mathrm{EMSO}^2(\sim, +1)$:

$$\exists R_1, \ldots, R_n (\varphi_A \wedge \bigwedge_{U \in K_1} \varphi_U \wedge \neg \bigwedge_{U \in K_2} \varphi_U).$$

□

Note, that by combining key and inclusion constraints also foreign key constraints can be covered. In [Fan and Libkin 2002] a special case of Proposition 5.1 was proved: the consistency problem for unary keys and foreign keys is NP-complete relative to DTD types. The extension to XML Schema's typing system (and to any regular tree language) was left as an open question. Note also that we do not know yet the precise complexity of the implication problem, we only have the 3NExpTime upper-bound given by the analysis of the proof of Theorem 3.1.

Finally we would like to stress that our method does not cover all known decidable cases of constraints. For instance it is shown in [Arenas et al. 2005] that consistency of unary foreign key constraints together with primary key constraints (a special case of key constraints) of **arbitrary arity** is decidable relative to DTD types. We couldn't find a way to express key constraints of arbitrary arity within our logical framework.

6. XPATH CONTAINMENT

In this section we provide another scenario for which decidability can be obtained using reductions to satisfiability of $\mathrm{FO}^2(\sim, +1)$. We define a fragment of XPath, which we call *LocalDataXPath,* for which static analysis tasks can be decided by using $\mathrm{FO}^2(\sim, +1)$. More precisely, satisfiability and containment test for unary queries expressed in these fragments, possibly in the presence of integrity constraints and schemas can be reduced to satisfiability of $\mathrm{FO}^2(\sim, +1)$.

For most fragments of XPath where the containment problem has been studied and established to be decidable, references to attribute values are not allowed. Notable exceptions are [Benedikt et al. 2005; Geerts and Fan 2005] where the satisfiability problem for various fragments of Core-Data-XPath was studied. However, the expressive power of LocalDataXPath is incomparable to the decidable fragments found in [Benedikt et al. 2005; Geerts and Fan 2005]. On the one hand we can only deal with the successor axis (because we need to encode each axis into $\mathrm{FO}^2(\sim, +1)$) while some fragments of [Benedikt et al. 2005; Geerts and Fan 2005] can handle

descendant axis. On the other hand our fragment is closed under negation - this was not the case for the decidable fragments with the descendant relation presented in [Benedikt et al. 2005; Geerts and Fan 2005]. In particular, in our case decidability of satisfiability immediately yields decidability for the inclusion problem. Finally we note that, again due to the compositionality of the logic, our decidability result still holds in the presence of integrity constraints and schemas.

As mentioned in the introduction, $\mathrm{FO}^2(\sim, <, +1)$ is a fragment of Core-Data-XPath. The proof of this statement follows the same lines as the proof for the inclusion of $\mathrm{FO}^2(<, +1)$ into unary-TL over words [Etessami et al. 2002]. The idea is to transform, by induction on the quantifier depth, each formula $\varphi(x)$ into a Core-Data-XPath expression that evaluates to true for the same set of nodes as $\varphi(x)$. In the inductive step one considers subformulas of the form $\exists y.\ (\tau \wedge \psi(y))$, where τ is an "order-type" formula, stating the complete information about the order between x and y. In our setting, τ also states the information about the data values associated with x, y. For instance, τ could be of the form $x < y \wedge y \neq x + 1 \wedge \alpha(x) \wedge \beta(y) \wedge x \sim y$, where α, β are labels. Each such formula τ can be translated into Core-Data-XPath syntax. For example, the formula τ above translates to $\texttt{Self} :: \alpha/@\texttt{A} = \texttt{Child} :: *\texttt{/Descendant} :: \beta/@\texttt{A}$. The size of the expression we obtain is exponential in the size of φ.

The language LocalDataXPath allows the comparison of attribute values, but compared with Core-Data-XPath it has two restrictions: (1) navigation is not allowed along the "transitive" axes as `Descendant` and `FollowingSibling` and (2) in an equality on attribute values either one of the location paths has to be absolute (i.e., starting from the root), or both (relative) location paths are strongly limited.

In LocalDataXPath only the following axes are allowed:

```
Axis := Child | Parent | NextSibling |
        PreviousSibling | Self | ElseWhere
```

Every axis corresponds to a binary relation on tree nodes. For instance, the `Child` axis is true for node pairs (v, w) where w is a vertical successor of w. The other axes are defined analogously. The new `ElseWhere` axis corresponds to the relation of pairs (v, w) of nodes, where $v \neq w$. It is added in order to allow at least some kind of global navigation.

We define the syntax of LocalDataXPath next. For the purpose of this article it is given in a simplified form as to compared with XPath.

$$
\begin{aligned}
\texttt{LocPath} &:= \texttt{RelLocPath} \mid \texttt{AbsLocPath}\\
\texttt{AbsLocPath} &:= \texttt{/}\,\texttt{RelLocPath}\\
\texttt{RelLocPath} &:= \texttt{Step} \mid \texttt{RelLocPath}\,\texttt{/}\,\texttt{Step}\\
\texttt{Step} &:= \texttt{Axis} :: \texttt{NameTest}\ \ \texttt{Predicate}^*\\
\texttt{SelfStep} &:= \texttt{Self} :: \texttt{NameTest}\ \ \texttt{Predicate}^*\\
\texttt{NameTest} &:= \texttt{Name}\\
\texttt{Predicate} &:= [\texttt{PredExpr}]\\
\texttt{PredExpr} &:= \texttt{LocPath} \mid\\
&\quad \texttt{LocPath}\,\texttt{/}\,\texttt{Attr}\ \texttt{EqOp}\ \texttt{AbsLocPath}\,\texttt{/}\,\texttt{Attr} \mid\\
&\quad \texttt{SelfStep}\,\texttt{/}\,\texttt{Attr}\ \texttt{EqOp}\ \texttt{Step}\,\texttt{/}\,\texttt{Attr} \mid\\
&\quad \texttt{PredExpr and PredExpr} \mid\\
&\quad \texttt{PredExpr or PredExpr} \mid \texttt{not PredExpr}\\
\texttt{Attr} &:= \texttt{@Name}\\
\texttt{EqOp} &:= = \ \mid\ !=
\end{aligned}
$$

In the above, the special symbols $/, =, !=, [,]$ and @ occurring on the right side of := are all terminals in the grammar. One could also add a wildcard in `NameTest` that would fit all labels; however this wildcard can be simulated using disjunction.

Furthermore, there is a syntactic restriction, called *safety*, on the way attributes can be used in the relative comparisons :

$$\texttt{SelfStep}\,\texttt{/}\,\texttt{Attr}\ \texttt{EqOp}\ \texttt{Step}\,\texttt{/}\,\texttt{Attr}\ .$$

In general terms, the restriction says that the attributes must be uniquely determined by the label. More formally, we say that an attribute name `B` is **associated** to a label `a` in a relative comparison as above, if the comparison contains a pattern of the form:

$$\texttt{Axis} :: \texttt{a}\ \ \texttt{Predicate}^*/\texttt{@B}\ .$$

(Note that on the left hand side of `EqOp`, the pattern will be preceded by a `Self` axis, while on the right it will be preceded by some other axis.) A set of expressions is **safe** if each label is associated to at most one attribute name. Somewhat ahead of time, we note that the point of safety is to store the value of the associated attribute in the node itself, and not in the attribute child.

Example 6.1. The following (safe) expression selects a node v with label `a`, if it agrees on attribute `B` with all its children labelled by `b`:

$$\neg(\texttt{Self} :: \texttt{a}/\texttt{@B}\ \ != \ \ \texttt{Child} :: \texttt{b}/\texttt{@B}).$$

The following expression is also safe:

$$\texttt{Self} :: \texttt{a}/\texttt{@B}\ \ = \texttt{Child} :: \texttt{b}/\texttt{@C}\ .$$

Although each of the two above expressions is safe on its own, they are no longer safe as a set, since `b` is associated to both `B` and `C`.

THEOREM 6.2. *Satisfiability and Containment for (unary or binary) Local-DataXPath safe expressions is decidable. This holds even relative to a schema consisting of a regular tree language and unary key and inclusion constraints.*

PROOF. (sketch) The proof is, of course, by translating the expressions into $\mathrm{FO}^2(\sim, +1)$ formulas. We encode XML documents as in the proof of Proposition 5.1 (using the XPath data model) with a small extension that we will introduce later. As long as expressions do not compare attribute values, there is no need to restrict the location paths: We can just use the standard transformation of Core-XPath into $\mathrm{FO}^2(<, +1)$ of [Marx 2005].

This easily extends to equality expressions with at most one relative location path by, intuitively, first simulating the relative path, then jumping to a node with the same data value and checking that this node satisfies its absolute path constraint by simulating the path backwards to the root. Note that it seems crucial here that the second path is absolute and thus does not start at the current node, as the two variables are needed for the navigation and therefore the current node can not be remembered. As an example the expression

$$\texttt{Child} :: \texttt{a}/\texttt{Child} ::\texttt{b}/@\texttt{B} = \\ /\texttt{Child} :: \texttt{c}/\texttt{NextSibling} :: \texttt{d}/@\texttt{C}$$

is translated into the following equivalent formula $\varphi(x)$:

$$\begin{aligned} &\exists y E_{\downarrow}(x,y) \wedge a(y) \wedge \\ &\quad \exists x E_{\downarrow}(y,x) \wedge b(x) \wedge \\ &\quad\quad \exists y E_{\downarrow}(x,y) \wedge B(y) \wedge \\ &\quad\quad\quad \exists x\ x \sim y \wedge C(x) \wedge \\ &\quad\quad\quad\quad \exists y E_{\downarrow}(y,x) \wedge d(y) \wedge \\ &\quad\quad\quad\quad\quad \exists x E_{\rightarrow}(x,y) \wedge c(x) \wedge \\ &\quad\quad\quad\quad\quad\quad \exists y E_{\downarrow}(y,x) \wedge \neg \exists x E_{\downarrow}(x,y). \end{aligned}$$

It only remains to explain how we can deal with relative (in-)equalities. To this end, we exploit the fact that the encoding of XML documents used so far only needs data values in attribute nodes. Thus, we can use the data values of element nodes for this purpose. Note, that the safety restriction on relative (in-)equalities ensures that for each element only one attribute is used in relative (in-)equalities. Therefore, we use data trees in which this attribute value (if any) is stored directly in the data value of the element node (on top of the normal storage in the corresponding attribute node). Note, that an additional $\mathrm{FO}^2(\sim, +1)$ formula can check that the data values in element nodes are consistent with those in the attribute nodes.

As an example, if ($\texttt{Self} :: \texttt{a}/@\texttt{C} \ = \ \texttt{Child} :: \texttt{b}/@\texttt{B}$) is a subexpression of our XPath expression at hand, then we consider data trees in which the data value of a-nodes is interpreted as the C-attribute and the data value of b-nodes as the B-attribute. By our assumption on safety, we do not need to put two attributes in the same node. Thus, the expression is equivalent to the formula $a(x) \wedge \exists y E_{\downarrow}(x,y) \wedge b(y) \wedge x \sim y$.

It is now straightforward to combine the techniques described so far with those of Section 5 to obtain the second statement of the theorem.

Containment for binary queries can be handled by having two distinguished nodes in each tree which correspond to a pair in the query result. □

It should be noted that satisfiability of a similar fragment of XPath with all axes besides `Following` and `Preceding` can be reduced to satisfiability of $\mathrm{FO}^2(\sim, <, +1)$. Unfortunately, we do not know if satisfiability of $\mathrm{FO}^2(\sim, <, +1)$ is decidable.

7. CONCLUSION

An interesting aspect of this work is to present in a unified framework decidability results that were studied separately in the past: consistency of integrity constraints and satisfiability of queries. In the future we hope to be able to also include related problems into the picture like the type inference problem [Alon et al. 2003].

Our main technical result is the decidability of $\mathrm{FO}^2(\sim, +1)$, which can be seen as a non trivial decidable fragment of Core-Data-XPath. A close inspection of the proof of Theorem 3.1 gives an upper bound of 3NExpTime for the decision procedure. The NExpTime lower bound follows from [Etessami et al. 2002]. It would be interesting to know the precise complexity of the problem.

Another obvious question is whether this decidability result can be extended to more expressive signatures. We have already mentioned the open and challenging problem of the decidability of $\mathrm{FO}^2(\sim, <, +1)$.

We conjecture that $\mathrm{FO}^2(\sim, +\omega)$ is decidable. This logic can use predicates of the form $E^k_{\downarrow}$ and $E^k_{\rightarrow}$ testing whether two nodes are at distance exactly k (downwards or rightwards). This is a proper extension, since $\mathrm{FO}^2(\sim, +1)$ even cannot express the fact that a node x has the same data value as its grandfather. However this feature would be useful in practice in order to express tree pattern queries which do not only depend on the labels of the nodes but also how their data values compare. It would also be useful in order to express more integrity constraints, in particular, *relative keys* and *relative inclusion constraints*, as investigated in [Arenas et al. 2005] in the presence of DTDs. We leave the decidability of $\mathrm{FO}^2(\sim, +\omega)$ as an open problem.

Another interesting issue is to find an algebraic form of the considered logics. In particular, we would like to find a decidable model of tree automata that can manipulate data values and express at least all of $\mathrm{FO}^2(\sim, +1)$. Unfortunately two-way automata using registers or pebbles for comparing data values are undecidable even with only one register or pebble [David 2004].

REFERENCES

N. Alon, T. Milo, F. Neven, D. Suciu and V. Vianu. XML with Data Values: Typechecking Revisited In *J. Comp. and Syst. Sci.*, 66(4):688-727 (2003).

M. Arenas, W. Fan and L. Libkin. Consistency of XML specifications. In *Inconsistency Tolerance*, LNCS 3300, pages 15-41, 2005.

M. Benedikt and C. Koch. XPath Leashed. To appear in *ACM Comp. Surveys*, 2009.

M. Benedikt, W. Fan, and F. Geerts. XPath Satisfiability in the Presence of DTDs. In *PODS'05*, pages 25-36, ACM 2005.

M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on words with data. In *LICS'06*, pages 7-16, IEEE 2006.

M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-Variable Logic on Data Trees and XML Reasoning. In *PODS'06*, pages 10-19, ACM 2006.

A. Brüggemann-Klein, D. Wood, and M. Murata. Regular tree and regular hedge languages over unranked alphabets. Technical report, April 2001. `http://citeseer.ist.psu.edu/451005.html`.

P. Bouyer, A. Petit and D. Thérien. An algebraic approach to data languages and timed languages. *Inf. and Comput.*, 182(2):137-162 (2003).

P. Buneman, S. B. Davidson, W. Fan, C. S. Hara and W. C. Tan. Reasoning about keys for XML. *Inf. Syst.*, 28(8):1037-1063 (2003).

J. Carme, J. Niehren and M. Tommasi. Querying Unranked Trees with Stepwise Tree Automata. In *RTA'04*, LNCS 3091, pages 105-118, 2004.

J. Cristau, C. Löding, W. Thomas. Deterministic Automata on Unranked Trees. In *FCT'05*, LNCS 3623, pages 68-79, 2005.

C. David. Mots et données infinis. Master thesis, Université Paris 7, LIAFA, 2004.

P. de Groote, B. Guillaume, and S. Salvati. Vector Addition Tree Automata. In *LICS'04*, pages 64-73, IEEE 2004.

S. Demri, R. Lazić, D. Nowak. On the Freeze Quantifier in Constraint LTL: Decidability and Complexity. In *TIME'05*, pages 113-121, IEEE 2005.

S. Demri and R. Lazić. LTL with the Freeze Quantifier and Register Automata. In *LICS'06*, pages 17-26, IEEE 2006.

K. Etessami, M.Y. Vardi, and Th. Wilke. First-Order Logic with Two Variables and Unary Temporal Logic. *Inf. and Comput.*, 179(2):279-295 (2002).

W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. In *J. of the ACM*, 49:368-406 (2002).

F. Geerts and W. Fan. Satisfiability of XPath Queries with Sibling Axes. In *DBPL'05*, LNCS 3774, pages 122-137, 2005.

G. Gottlob, C. Koch, and R. Pichler. Efficient Algorithms for Processing XPath Queries. In *VLDB'02*, pages 95-106, Morgan Kaufmann, 2002.

E. Grädel and M. Otto. On Logics with Two Variables. *Theor. Comp. Sci.*, 224:73-113 (1999).

M. Jurdziński and R. Lazić. Alternation-Free Modal Mu-Calculus for Data Trees. In *LICS'07*, pages 131-140, IEEE 2007.

M. Kaminski and N. Francez. Finite Memory Automata. *Theor. Comp. Sci.*, 134(2):329-363 (1994).

E. Kieroński and M. Otto. Small Substructures and Decidability Issues for First-Order Logic with Two Variables. In *LICS'05*, pages 448-457, IEEE 2005.

W. Martens, J. Niehren. Minimizing Tree Automata for Unranked Trees. In *Int. Symp. on Database Programming Languages*, LNCS 3774, pages 232-246, 2005.

M. Marx. First order paths in ordered trees. In *ICDT'05*, LNCS 3363, pages 114-128, 2005.

M. Mortimer. On languages with two variables. *Zeitschr. f. math. Logik u. Grundlagen d. Math.*, 21:135-140 (1975).

F. Neven and T. Schwentick. XPath Containment in the Presence of Disjunction, DTDs, and Variables. In *ICDT'03*, LNCS 2572, pages 315-329, 2003.

F. Neven and T. Schwentick. Query automata. *Theor. Comp. Sci.*, 275(1-2):633-674 (2002).

K. Neeraj Verma, H. Seidl, T. Schwentick. On the Complexity of Equational Horn Clauses. In *CADE'05*, LNCS 3632, pages 337-352, 2005.

F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 15(3):403-435 (2004).

Christos H. Papadimitriou. On the complexity of integer programming. *J. of the ACM*, 28(4):765–768 (1981).

XML Path Language (XPath), W3C Recommendation 16 November 1999. Available at `http://www.w3.org/TR/xpath`.