Reasoning about integrity constraints for tree-structured data

Wojciech Czerwiński, Claire David, Filip Murlak, Pawel Parys

Noname manuscript No.
(will be inserted by the editor)

Reasoning about integrity constraints for tree-structured data

Wojciech Czerwiński · Claire David · Filip Murlak · Paweł Parys

Received: date / Accepted: date

Abstract We study a class of integrity constraints for tree-structured data modelled as data trees, whose nodes have a label from a finite alphabet and store a data value from an infinite data domain. The constraints require each tuple of nodes selected by a conjunctive query (using navigational axes and labels) to satisfy a positive combination of equalities and a positive combination of inequalities over the stored data values. Such constraints are instances of the general framework of XML-to-relational constraints proposed recently by Niewerth and Schwentick. They cover some common classes of constraints, including W3C XML Schema key and unique constraints, as well as domain restrictions and denial constraints, but cannot express inclusion constraints, such as reference keys. Our main result is that consistency of such integrity constraints with respect to a given schema (modelled as a tree automaton) is decidable. An easy extension gives decidability for the entailment problem. Equivalently, we show that validity and containment of unions of conjunctive queries using navigational axes, labels, data equalities and inequalities is decidable, as long as none of the conjunctive queries uses both equalities and inequalities; without this restriction, both problems are known to be undecidable. In the context of XML data exchange, our result can be used to establish decidability for a consistency problem for XML schema mappings. All the decision procedures are doubly exponential, with matching lower bounds. The complexity may be lowered to singly exponential, when conjunctive queries are replaced by tree patterns, and the number of data comparisons is bounded.

The first, third, and fourth author of this paper were supported by Poland's National Science Centre grant 2013/11/D/ST6/03075.

W. Czerwiński, University of Warsaw, Poland, E-mail: wczerwin@mimuw.edu.pl · C. David, Université Paris-Est Marne-la-Vallée, France, E-mail: claire.david@u-pem.fr · F. Murlak, University of Warsaw, Poland, E-mail: fmurlak@mimuw.edu.pl · P. Parys, University of Warsaw, Poland, E-mail: parys@mimuw.edu.pl

1 Introduction

Static analysis is an area of database theory that focuses on deciding properties of syntactic objects, like queries, integrity constraints, or data dependencies. The unifying paradigm is that because these objects are mostly user-generated, they tend to be small; hence, higher complexities are tolerable. The fundamental problems include satisfiability, validity, containment, and equivalence of queries [9,25], as well as consistency and entailment of integrity constraints [16,28]. More specialized tasks include query rewriting in data integration scenarios [24], and manipulating schema mappings in data exchange and schema evolution scenarios [1,15]. Many of these problems are equivalent to satisfiability of fragments of first order logic, possibly over a restricted class of structures, but they are rarely presented this way, because the involved fragments are tailored for specific applications, and usually do not form natural sublogics. As satisfiability over arbitrary structures is undecidable even for relatively simple fragments of first order logic, in static analysis undecidability is always close [19,20].

In this paper we present a decidability result (with tight complexity bounds) for a problem in static analysis for tree-structured data. The specific model we consider is that of data trees: finite ordered unranked trees whose nodes have a label from a finite alphabet and store a data value from an infinite data domain. The problem has three possible interpretations:

- consistency modulo schema for a class of integrity constraints;
- validity modulo schema for a class of queries; and
- consistency for a class of schema mappings.

The more general problems of entailment (or implication) of constraints and containment of queries are—as is often the case—very close to their restricted counterparts listed above, and can be solved by easy modifications of our decision procedure.

Our basic setting is that of consistency of integrity constraints; it seems best suited for proofs and—in combination with entailment—the most appealing. We consider *non-mixing constraints* of the forms

$$\alpha(\bar{x}) \Rightarrow \eta_{\sim}(\bar{x}) \qquad \text{and} \qquad \alpha(\bar{x}) \Rightarrow \eta_{\nsim}(\bar{x})$$

that require each tuple $\bar{x}$ of nodes selected by α to satisfy, respectively, a positive combination of equalities $\eta_{\sim}$ or a positive combination of inequalities $\eta_{\nsim}$ over the stored data values. As tuple selectors $\alpha(\bar{x})$ we use conjunctive queries over the signature including label tests and the usual navigational axes. For example, the constraint

$$a(x) \ \wedge \ x \downarrow y \ \wedge \ x \downarrow y' \ \wedge \ y \rightarrow^{+} y' \ \Rightarrow \ y \nsim y'$$

expresses that different children of the same a-labelled node store different data values, and the constraint

$$a(x) \ \wedge \ x \downarrow y \ \wedge \ x \downarrow y' \ \wedge \ x \downarrow y'' \ \Rightarrow \ y \sim y' \ \vee \ y' \sim y'' \vee y'' \sim y$$

expresses that at most two different data values are used by children of each a-labelled node. The consistency problem is to decide if there exists an instance of a given schema that satisfies a given set of constraints. In the example above, there exists an instance satisfying both constraints if and only if the schema allows trees without a-labelled nodes with more than two children.

What is the expressive power of non-mixing constraints? Let us first look at what they cannot do. Being first-order constraints, they cannot compare full subtrees, unlike some other formalisms [21,22]. They have purely universal character (can be written as universal sentences of first order logic), so they cannot express general inclusion dependencies nor foreign keys, as these need quantifier alternation. Finally, the inability to mix freely data equalities and inequalities within a single constraint makes them unable to express general functional dependencies. What can they do, then?

Non-mixing integrity constraints can be seen as a special case of the general framework of XML-to-relational constraints (X2R constraints) introduced by Niewerth and Schwentick [27]. Within this framework they cover a wide subclass of functional dependencies, dubbed XKFDs, which are particularly well suited for tree-structured data. They include W3C XML Schema key and unique constraints [18], as well as absolute and relative XML keys by Arenas, Fan, and Libkin [2], and XFDs by Arenas and Libkin [3]. XKFDs can be expressed with non-mixing constraints of the form $\alpha(\bar{x}) \Rightarrow \eta_{\sim}(\bar{x})$; that is, using only data inequalities.

Constraints of the form $\alpha(\bar{x}) \Rightarrow \eta_{\sim}(\bar{x})$—that is, using only equalities—can express all sorts of finite data domain restrictions, either to a specific set of constants or to a set of data values taken from the data tree (the latter can be seen as a limited variant of inclusion constraints), as well as cardinality restrictions over data values (like in the example above).

The novelty of our work is that we allow these two kinds of constraints simultaneously. Unrestricted mixing of data equalities and inequalities in constraints would immediately lead to undecidability [6], but for non-mixing constraints we can show decidability of the consistency problem, and a slight extension of the proof gives decidability for entailment (with the same complexity bounds).

Our approach leads through a simple model property, which asserts that a set of constraints is satisfiable if and only if it has a model of *bounded data cut* [7]; that is, the number of data values shared by any subforest of the model and its complement is bounded. This property can be seen as a strengthening of the bounded clique-width property [11], in which decompositions must follow the structure of data trees. The robustness of our approach is witnessed by the fact that it can be naturally extended to constraints in which tuple selectors $\alpha(\bar{x})$ are expressed in monadic second order logic (MSO) using label tests and navigational predicates. At the core of our argument lies a simple lemma of geometric nature.

Under the second interpretation our result shows decidability of validity and containment for unions of conjunctive queries where each conjunctive query can use either data equality or inequality, but never both. Seen this way,

our result is a uniform extension of decidability results for UCQs using only data equality, and UCQs using only data inequality by Björklund, Martens and Schwentick [6] (see also [12]). However, it cannot be obtained via a combination of techniques used in these cases, as they are virtually contradictory: they require assuming that almost all data values in counter-examples are, respectively, different and equal. If data equalities and inequalities are mixed freely in UCQs, even validity is undecidable [6].

In its third incarnation, our result gives decidability of the consistency problem for XML schema mappings with source integrity constraints, which asks to decide if there exists a source instance which satisfies the integrity constraints and admits a target instance satisfying the requirements imposed by the schema mapping.

In all three cases (excluding the unsurprisingly non-elementary MSO extension), the decision procedure is doubly exponential. This bound is tight, as already validity modulo schema for UCQs over trees without data values is 2ExpTime-complete [6]. We show that restricting the CQs to tree patterns does not help. However, the complexity does drop to ExpTime-complete when we replace CQs with tree patterns and bound the number of variables used in data comparisons.

A broader context for our work is the rich landscape of results on static analysis for the popular XML query language XPath [5,26] and related formalisms like alternating register automata [17,23] or the two-variable fragment of first order logic with data comparisons [8]. These formalisms do not compare easily with ours. Arbitrary alternation of quantifiers (implicit, in the case of XPath) lets them reach far beyond conjunctive queries. But the restriction on the number of registers or variables (reflected in the the syntax of XPath) limits data comparisons: one cannot compare data values from too many nodes at the same time. In their basic form, our results imply decidability (with the same tight complexity bounds) of the containment problem in the presence of a schema for unions of XPath queries without negation, where each query uses either equality or inequality, but never both. The extension to MSO constraints allows free use of negation as long as data comparisons are not used under negation.

The remainder of the paper begins with a precise definition of non-mixing constraints and a short discussion of their scope (Section 2). Then we present the decision procedure for consistency of non-mixing constraints and show its optimality (Section 3). We continue with a potpourri of extensions and connections: the entailment problem (Section 4.1), the lower-complexity fragment (Section 4.2), the relationships with existing constraint formalisms (Section 4.4), the two alternative interpretations of our results (Section 4.3 and 4.5), a comparison with clique-width (Section 4.6), and the MSO extension (Section 4.7). We conclude with a brief discussion of further possible extensions and open questions (Section 5).

This is an extended version of an 18-pages-long paper under the same title presented at ICDT 2016. The new material includes full proofs of all results, as well as the comparison with clique-width and the MSO extension. There is

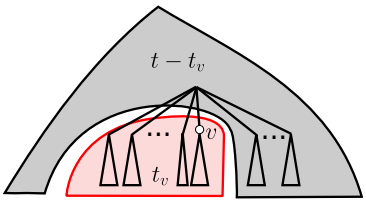

Fig. 1 A tree t and a subforest t_v associated with a node v.

also a major difference in the way the proof of the main result is presented. In the conference version, register tree automata are used to recognize witnesses for consistency of bounded data cut. Here, we encode such witnesses as trees over a finite alphabet and use ordinary tree automata. The new formulation encapsulates reasoning about data values within the encoding, and harmonizes with the clique-width approach and the MSO extension.

2 Non-mixing constraints

2.1 Preliminaries

Let us fix a finite labelling alphabet Γ and a countably infinite set of data values $\mathbb{D}$. A *data tree* t is a finite ordered unranked tree whose nodes are labelled with elements of Γ by function $\mathsf{lab}_t : \mathsf{dom}_t \to \Gamma$, and with elements of $\mathbb{D}$ by function $\mathsf{val}_t : \mathsf{dom}_t \to \mathbb{D}$; here, dom_t stands for the *domain* of tree t, that is, the set of its nodes. If $\mathsf{lab}_t(v) = a$ and $\mathsf{val}_t(v) = d$, we say that node v has label a and stores data value d. A *data forest* f is a sequence of data trees whose roots are considered siblings (with the inherited order); lab_f, val_f, and dom_f are defined naturally. While each data tree contains at least the root, a data forest can be empty. For a node v of t, we write t_v for the data forest consisting of subtrees of t rooted at v itself and at all preceding siblings of v; by slight abuse of notation we write $t - t_v$ for the remaining part of t (see Figure 1 for illustration). For a forest f we use the analogous notation, f_v and $f - f_v$.

We abstract schemas as tree automata in the "previous sibling, last child" variant. A *tree automaton* $\mathcal{A}$ is a tuple (Q, q_0, F, δ), where Q is a finite set of states, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is a set of accepting states, and $\delta \subseteq Q \times Q \times \Gamma \times Q$ is a set of transitions. During the computation the automaton assigns a state to each node v of the input tree t, based on the accumulated information about t_v. More precisely, the state for the node v depends on the label of v and the states from the previous sibling and the last child of v. In leftmost siblings and in leaves we resort to imaginary nodes outside of the actual tree t, which are always assigned the initial state q_0. Formally, let dom_t^{cl} be the set containing each node of t, an artificial previous sibling for each leftmost sibling in t, and an artificial (last) child for each leaf in t. A run of $\mathcal{A}$ on t is a function $\rho : \mathsf{dom}_t^{cl} \to Q$ such that $\rho(v) = q_0$ for every

node $v \in \mathsf{dom}_t^{cl} - \mathsf{dom}_t$, and for every node $v \in \mathsf{dom}_t$ with previous sibling v_{ps} and last child v_{lc} there is a transition $(\rho(v_{ps}), \rho(v_{lc}), \mathsf{lab}_t(v), \rho(v)) \in \delta$. A run ρ is *accepting* if it assigns a state from F to the root of t, and a tree t is *accepted* by $\mathcal{A}$ if it admits an accepting run. Runs on forests are defined entirely analogously; acceptance is based on the state in the root of the last tree (if the forest is empty, we take the initial state q_0). An automaton is *deterministic* if δ is a function $Q \times Q \times \Gamma \to Q$. Each deterministic automaton has a unique run on each tree (and forest), and can be complemented (negated) simply by replacing the set of final states F with its complement $Q - F$.

To facilitate the use of the standard first order semantics, we model data trees and data forests as relational structures over signature

$$\mathsf{sig}_{dt} = \{\downarrow, \downarrow^+, \rightarrow, \rightarrow^+, \sim, \nsim\} \cup \Gamma \cup \mathbb{D} \cup \check{\mathbb{D}}$$

with $\check{\mathbb{D}} = \{\check{d} \mid d \in \mathbb{D}\}$; that is, we have

- binary relations: *child* $\downarrow$, *descendant* $\downarrow^+$, *next sibling* $\rightarrow$, and *following sibling* $\rightarrow^+$;
- data equality relation $\sim$ and data inequality relation $\nsim$ that contain pairs of nodes storing, respectively, the same data value and different data values;
- unary relation a for each label $a \in \Gamma$;
- unary relations d and $\check{d}$ for each data value $d \in \mathbb{D}$ that contain nodes storing, respectively, data value d and any data value different from d.

Signature sig_{dt} is infinite (because of $\mathbb{D}$ and $\check{\mathbb{D}}$), but queries use only finite fragments. We include $\check{\mathbb{D}}$ in the signature to keep negation out of the syntax.

A *conjunctive query* $\alpha(x_1, \ldots, x_n)$ over a signature sig is a first order formula of the form

$$\exists y_1 \ldots \exists y_m \, \beta(x_1, \ldots, x_n, y_1, \ldots, y_m)\,,$$

where $\beta(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is a conjunction of atoms over signature sig and variables $x_1, \ldots, x_n, y_1, \ldots, y_m$.

2.2 Definition

In their most general form, *non-mixing integrity constraints* σ are formulas of the form

$$\alpha(\bar{x}) \Rightarrow \eta_\sim(\bar{x}) \wedge \eta_\nsim(\bar{x})$$

where

- $\alpha(\bar{x})$ is a conjunctive query over the signature $\mathsf{sig}_{nav} = \{\downarrow, \downarrow^+, \rightarrow, \rightarrow^+\} \cup \Gamma$;
- $\eta_\sim(\bar{x})$ is a finite positive Boolean combination of atoms over the signature $\mathsf{sig}_\sim = \{\sim\} \cup \mathbb{D}$ and variables $\bar{x}$;
- $\eta_\nsim(\bar{x})$ is a finite positive Boolean combination of atoms over the signature $\mathsf{sig}_\nsim = \{\nsim\} \cup \check{\mathbb{D}}$ and variables $\bar{x}$.

Query α is called the *selector* of σ, and $\eta_\sim, \eta_\nsim$ are its *assertions*. Non-mixing constraints have the usual semantics of first order logic formulas: a data tree t satisfies constraint σ, denoted $t \models \sigma$, if each tuple $\bar{v}$ of nodes of t selected by α satisfies both $\eta_\sim$ and $\eta_\nsim$; that is,

$$t \models \alpha(\bar{v}) \quad \text{implies} \quad t \models \eta_\sim(\bar{v}) \wedge \eta_\nsim(\bar{v})\,.$$

For a set Σ of non-mixing constraints, we write $t \models \Sigma$ if $t \models \sigma$ for all $\sigma \in \Sigma$.

Note that $\alpha \Rightarrow \eta_\sim \wedge \eta_\nsim$ is equivalent to $\{\alpha \Rightarrow \eta_\sim\,,\ \alpha \Rightarrow \eta_\nsim\}$. Consequently, each set Σ of non-mixing constraints is equivalent to $\Sigma_\sim \cup \Sigma_\nsim$, where $\Sigma_\sim$ is a set of constraints of the form $\alpha \Rightarrow \eta_\sim$, $\Sigma_\nsim$ is a set of constraints of the form $\alpha \Rightarrow \eta_\nsim$, and the sizes of $\Sigma_\sim$ and $\Sigma_\nsim$ are bounded by the size of Σ. Thus, without loss of generality, we restrict our attention to sets of constraints of the form $\Sigma_\sim \cup \Sigma_\nsim$, which do not mix $\mathsf{sig}_\sim$ and $\mathsf{sig}_\nsim$ (hence "non-mixing"). One can also assume that α is quantifier free: $\exists \bar{y}\, \alpha(\bar{x}, \bar{y}) \Rightarrow \eta(\bar{x})$ is equivalent to $\alpha(\bar{x}, \bar{y}) \Rightarrow \eta(\bar{x})$.

2.3 Scope

Using non-mixing constraints one can express a variety of useful constraints. Let us consider a database storing information about banks, each in a separate sub-document. We want each bank to be identified by its BIC number. This **key constraint** can be expressed as

$$q_{\text{BIC}}(x, x') \wedge q_{\text{BIC}}(y, y') \wedge x \neq y \;\Rightarrow\; x' \nsim y'$$

where q_{BIC} selects the root of the sub-document for bank, and the node storing the BIC number. Depending on the schema, query q_{BIC} could be for instance $q_{\text{BIC}}(x, x') = \texttt{bank}(x) \wedge x \downarrow x' \wedge \texttt{BIC}(x')$. Node inequality $\neq$ is not part of the signature, but can be expressed using sig_{nav}. Assuming that the roots of the sub-documents for banks are siblings, $x \neq y$ can be replaced by $x \rightarrow^+ y$. In general, we also need to consider four other possible ways in which two different nodes x and y can be positioned in a tree (up to swapping x and y):

$$x \downarrow^+ y\,,\ x \rightarrow^+ z \wedge z \downarrow^+ y\,,\ z \downarrow^+ x \wedge z \rightarrow^+ y\,,\ \text{ and } \ z \downarrow^+ x \wedge z \rightarrow^+ z' \wedge z' \downarrow^+ y\,,$$

which means that we need five non-mixing constraints to express a single key constraint.

Another natural constraint is that account numbers should be different for every account within the same bank, but different banks may use the same account numbers. Such a **relative key constraint** can also be expressed as

$$\texttt{bank}(z) \wedge z \downarrow^+ x \wedge z \downarrow^+ y \wedge q_{\text{ACC}}(x, x') \wedge q_{\text{ACC}}(y, y') \wedge x \neq y \;\Rightarrow\; x' \nsim y'\,.$$

where $q_{\text{ACC}}(x, x')$ selects account x and its number x', similarly to q_{BIC}.

We can also express **multi-attribute keys** (i.e. keys using composite fields). For example

$$q_{\text{BIC}}(u,u') \wedge q_{\text{BIC}}(v,v') \wedge u{\downarrow^+}x \wedge v{\downarrow^+}y \wedge q_{\text{ACC}}(x,x') \wedge q_{\text{ACC}}(y,y') \wedge x \neq y \;\Rightarrow \\ \Rightarrow\; u' \nsim v' \vee x' \nsim y' \,.$$

asserts that BIC and account number form an absolute key, not relative to bank sub-document.

If, as a result of redundancy, BIC appears in several places within a bank sub-document, using the **singleton constraint**

$$\texttt{bank}(x) \wedge x \downarrow^+ x' \wedge \texttt{BIC}(x') \wedge x \downarrow^+ x'' \wedge \texttt{BIC}(x'') \;\Rightarrow\; x' \sim x''$$

we can guarantee that each time it gives the same value (for the same bank).

Assume now that each bank has a director and several branches, each of them having a team of employees among which one is the manager of the branch. The information about each employee is stored in a sub-document of its branch's sub-document. Each employee reports either to the manager of the branch or directly to the director of the bank. Using a conjunctive query $q_{\text{SUPER}}(x,y,z)$, we can select the director's ID node x, the branch manager's ID node y and the node z storing the supervisor's ID for an employee of the same branch. The constraint on employee's supervisor can be encoded as

$$q_{\text{SUPER}}(x,y,z) \;\Rightarrow\; x \sim z \vee y \sim z \,.$$

Following this idea we can express **inclusion constraints of a restricted form**, where the intended superset is a tuple of values that can be selected by a conjunctive query. This includes **enumerative domain restrictions**, like the constraint

$$\texttt{creditCard}(x) \wedge x \downarrow^+ x' \wedge \texttt{brand}(x') \Rightarrow \\ \Rightarrow\; \mathit{Visa}(x') \vee \mathit{MasterCard}(x') \vee \mathit{AmericanExpress}(x') \,,$$

ensuring that banks issue only Visa, Master Card, and American Express cards. Unrestricted inclusion constraints are beyond the scope of our formalism. Indeed, non-mixing constraints cannot be violated by removing nodes, which is not the case even for the simplest unary inclusion constraints, like *each value stored in an a node is also stored in a b node.*

Our formalism is also capable of expressing **cardinality constraints**. Assume, for instance, that banks support charity projects by delegating their employees to help. The projects are organized by category (culture, education, environment, etc.) and each project sub-document carries the list of involved employees. For the sake of balance, we want each category to involve at most ten different employees in total. This can be imposed by selecting eleven employee nodes below a single category node and imposing at least two of them to carry the same data value by means of a long disjunction of data equalities. We can also ensure that no employee is involved in more than three different projects: the conjunctive query selects four different project nodes and

an employee for each of them; the assertion imposes at least two of the four employees to have different ID.

Let us remark that while these constraints look clumsy expressed as non-mixing constraints, one can easily imagine a syntactic-sugar layer on top of our formalism. The point is that all these constraints can be rewritten as non-mixing constraints of linear size (except for the cardinality constraints, where the size would grow by a factor proportional to the numerical bounds).

In Section 4 we examine the expressive power of non-mixing constraints further by comparing them to other existing formalisms.

3 Consistency problem

Our main result is decidability of the consistency problem for non-mixing constraints:

PROBLEM:	**Consistency of non-mixing constraints**
INPUT:	A set Σ of non-mixing constraints, a tree automaton $\mathcal{A}$.
QUESTION:	Is there a data tree $t \in L(\mathcal{A})$ such that $t \models \Sigma$?

More precisely, we show the following theorem, establishing tight complexity bounds.

Theorem 1 *Consistency of non-mixing constraints is* 2EXPTIME*-complete.*

The reminder of this section is devoted to the proof of Theorem 1. The proof is based on a simple idea with a geometric flavour, but does not require any specialist knowledge from geometry or linear algebra. Consider a family of finite unions of affine subspaces of an Euclidean space. The intersection of this family can be also represented as a finite union of affine subspaces and we show that their number can be bounded independently of the cardinality of the family. From this bound we infer a "bounded data cut" model property for non-mixing constraints, where by *data cut* of a data tree t, denoted by $\mathit{datacut}(t)$, we mean the maximum over nodes $v \in \mathsf{dom}_t$ of the number of data values shared by t_v and $t - t_v$. With bounded data cut, we can reduce the consistency problem to the emptiness problem for tree automata (over a finite alphabet). In the final subsection we prove the lower bound.

3.1 Intersecting unions of subspaces

By a *subspace* of $\mathbb{D}^\ell$ we mean a subset of $\mathbb{D}^\ell$ defined by equating pairs of coordinates and fixing coordinates; that is, it is a set of points $(x_1, x_2, \ldots, x_\ell)$ in space $\mathbb{D}^\ell$ defined by a conjunction of equalities of the form $x_i = x_j$ or $x_i = d$ where $d \in \mathbb{D}$. By simple rewriting of equalities, each nonempty subspace of $\mathbb{D}^\ell$ can be defined with a *canonical* set of at most ℓ equalities such that

- for each coordinate i we have either $x_i = x_j$ with $i < j$, or $x_i = d$ with $d \in \mathbb{D}$, or nothing;

– each coordinate j occurs at most once on the right side of an equality; and
– no data value d is used in more than one equality.

A subspace of $\mathbb{D}^\ell$ has *dimension* k if its canonical definition consists of $\ell - k$ equalities. In other words, each equality that does not follow from the others decreases the dimension by one. To enhance intuitions, let us remark that if we equip $\mathbb{D}^\ell$ with the structure of linear space by assuming that $\mathbb{D}$ is a field, this notion of dimension coincides with the classical notion of dimension for affine subspaces (of which the subspaces above are a special case).

An intersection $X \cap Y$ of subspaces X, Y is also a subspace, defined by the conjunction of conditions defining X and Y. If $X \not\subseteq Y$, then the canonical definition of $X \cap Y$ contains at least one more equation than that of X; consequently, the dimension of $X \cap Y$ is strictly smaller than the dimension of X. Similarly, intersecting unions of subspaces, we obtain a union of subspaces; the following lemma gives a bound on the size of such union.

Lemma 1 *Let $Z_1, Z_2, \ldots, Z_m \subseteq \mathbb{D}^\ell$ be such that each Z_i is a union of at most n subspaces of $\mathbb{D}^\ell$. Then, $Z_1 \cap Z_2 \cap \cdots \cap Z_m$ can be represented as a union of at most n^ℓ subspaces of $\mathbb{D}^\ell$.*

Proof Assume that $Z_1 \cap Z_2 \cap \cdots \cap Z_{i-1}$ is a union $X_1 \cup X_2 \cup \cdots \cup X_p$ of subspaces of $\mathbb{D}^\ell$. We can write Z_i as $Y_1 \cup Y_2 \cup \cdots \cup Y_n$, where some of subspaces Y_k may be empty. We have

$$\begin{aligned} Z_1 \cap Z_2 \cap \cdots \cap Z_i &= (X_1 \cup X_2 \cup \cdots \cup X_p) \cap Z_i = \\ &= (X_1 \cap Z_i) \cup (X_2 \cap Z_i) \cup \cdots \cup (X_p \cap Z_i)\,. \end{aligned}$$

Let us examine a single $X_j \cap Z_i$. If $X_j \subseteq Y_k$ for some k, then $X_j \cap Z_i = X_j$. Otherwise, $X_j \cap Z_i$ is a union of n subspaces, $X_j \cap Y_1, X_j \cap Y_2, \ldots, X_j \cap Y_n$, where each $X_j \cap Y_k$ is either empty or has dimension strictly smaller than X_j. Thus, when $X_1 \cup X_2 \cup \cdots \cup X_p$ is intersected with Z_i, each X_j either does not change, or is split into at most n subspaces of strictly smaller dimension; if X_j is a point, in the second possibility it disappears.

Now, consider the following process: begin with $\mathbb{D}^\ell$, a single subspace of dimension ℓ, and then intersect with Z_i for i from 1 to m, one by one. Since with each split, the dimension strictly decreases, each non-empty subspace in the resulting union is obtained in the course of at most ℓ splits. Since each split generates at most n subspaces, we cannot obtain more than n^ℓ subspaces in this process. □

We remark that the bound in Lemma 1 is tight, as shown by the following example.

Example 1 [1] Assume $0, 1 \in \mathbb{D}$ and let $Z_i = \left\{ \bar{x} \subseteq \mathbb{D}^\ell \;\middle|\; x_i = 0 \vee x_i = 1 \right\}$ for $i = 1, 2, \ldots, \ell$. Then $Z_1 \cap Z_2 \cap \cdots \cap Z_\ell = \{0, 1\}^\ell$ is a union of 2^ℓ (disjoint) subspaces of $\mathbb{D}^\ell$ of dimension 0.

[1] Provided by Michał Pilipczuk, during the Warsaw Automata Group's research camp *Autobóz 2015*.

3.2 Bounding the data cut

Based on the geometric fact we have proved in the previous subsection, in Lemma 3 we bound the data cut of data trees witnessing consistency of non-mixing constraints. The proof relies on a simple compositionality property for conjunctive queries over trees, shown in Lemma 2.

Lemma 2 *Let $\alpha(\bar{x}, \bar{y})$ be a conjunction of atoms over sig_{nav}, where $\bar{x}$ and $\bar{y}$ are disjoint, and let w be a node of a data tree t. For all tuples $\bar{u}, \bar{u}'$ of nodes from t_w and tuples $\bar{v}, \bar{v}'$ of nodes from $t - t_w$, if*

$$t \models \alpha(\bar{u}, \bar{v}) \quad \text{and} \quad t \models \alpha(\bar{u}', \bar{v}')\,,$$

then

$$t \models \alpha(\bar{u}, \bar{v}') \quad \text{and} \quad t \models \alpha(\bar{u}', \bar{v})\,.$$

Proof Let $\bar{u}, \bar{u}', \bar{v}, \bar{v}'$ be as in the statement of the lemma. Since $\alpha(\bar{x}, \bar{y})$ is a conjunction of atoms, we only need to check that each atom of $\alpha(\bar{u}, \bar{v}')$ and $\alpha(\bar{u}', \bar{v})$ is satisfied in t. Given that $t \models \alpha(\bar{u}, \bar{v})$ and $t \models \alpha(\bar{u}', \bar{v}')$, it is enough to examine atoms using variables from both $\bar{x}$ and $\bar{y}$. That excludes unary relations and leaves us with atoms of the forms $x_i \downarrow y_j, x_i \downarrow^+ y_j, x_i \rightarrow y_j, x_i \rightarrow^+ y_j$, and symmetrical. Given that variables $\bar{x}$ are matched within t_w, and variables $\bar{y}$ are matched within $t - t_w$, atoms $x_i \downarrow y_j$, $x_i \downarrow^+ y_j$, $y_j \rightarrow x_i$, and $y_j \rightarrow^+ x_i$ are excluded by the combination of two things: the way t_w and $t - t_w$ are positioned within tree t (see Figure 1 on page 5), and the fact that $t \models \alpha(\bar{u}, \bar{v})$. That is, it remains to consider $y_j \downarrow x_i$, $y_j \downarrow^+ x_i$, $x_i \rightarrow y_j$, and $x_i \rightarrow^+ y_j$. Suppose $y_j \downarrow x_i$ occurs in α. We know that $v_j \downarrow u_i$ and $v'_j \downarrow u'_i$. Since nodes u_i, u'_i are from t_w and nodes v_j, v'_j are from $t - t_w$, it follows immediately that v_j and v'_j are equal to the parent of node w, and u_i, u'_i are siblings of w or w itself. Consequently, $v_j \downarrow u'_i$ and $v'_j \downarrow u_i$. For the remaining three kinds of atoms the reasoning is similar. If $v_j \downarrow^+ u_i$ and $v'_j \downarrow^+ u'_i$, then v_j, v'_j are ancestors of w and u_i, u'_i are nodes in t_w, so $v_j \downarrow^+ u'_i$ and $v'_j \downarrow^+ u_i$ follows. If $u_i \rightarrow v_j$ and $u'_i \rightarrow v'_j$, then $u_i = u'_i = w$ and $v_j = v'_j$ is w's next sibling. Finally, if $u_i \rightarrow^+ v_j$ and $u'_i \rightarrow^+ v'_j$, then u_i, u'_j are preceding siblings of w (or w itself) and v_j, v'_j are following siblings of w. □

Lemma 3 *If $\Sigma_\sim \cup \Sigma_{\not\sim}$ is satisfied in a data tree t, it is also satisfied in some data tree t' obtained from t by changing data values, such that*

$$\mathit{datacut}(t') \leq \ell \cdot 2^\ell \cdot (\ell + m)^{\ell^2} \cdot |\Sigma_\sim|\,,$$

where ℓ and m are the maximal numbers of, respectively, variables and predicates from $\mathbb{D} \cup \tilde{\mathbb{D}}$ in the constraints from $\Sigma_\sim$.

Proof Assume that $t \models \Sigma_\sim \cup \Sigma_{\not\sim}$. We show that for each node w of the data tree t, one can replace all but $\ell \cdot 2^\ell \cdot (\ell + m)^{\ell^2} \cdot |\Sigma_\sim|$ data values used in t_w with distinct fresh data values without violating $\Sigma_\sim \cup \Sigma_{\not\sim}$. After this operation is performed for a node w, the number of data values used both in t_w and $t - t_w$

is bounded by $\ell \cdot 2^\ell \cdot (\ell + m)^{\ell^2} \cdot |\Sigma_\sim|$. Moreover, as the fresh data values are to be distinct, the new $\sim$ relation over nodes of t is a subset of the old one. In consequence, the operation does not increase the number of data values shared by $t_{w'}$ and $t - t_{w'}$ for other nodes w'. Consequently, applying this operation for each node of t in an arbitrary order, we obtain a model of bounded data cut.

Let us fix a node w of t. As long as the fresh values are pairwise different, the obtained tree will still satisfy $\Sigma_\nsim$. Hence, we only need to ensure that $\Sigma_\sim$ is not violated. Consider a constraint $\alpha \Rightarrow \eta_\sim$ in $\Sigma_\sim$. Recall that we assume that α is quantifier free. Let $\bar{x}$, $\bar{y}$ be a partition of variables used in α (one of the tuples $\bar{x}$, $\bar{y}$ may be empty). We shall indicate the partition of variables by writing the constraint as $\alpha(\bar{x}, \bar{y}) \Rightarrow \eta_\sim(\bar{x}, \bar{y})$. The intended meaning is that the variables $\bar{x}$ refer to nodes in t_w, and the variables $\bar{y}$ refer to nodes outside of t_w. Directly from the definition it follows that $t \models \alpha \Rightarrow \eta_\sim$, if and only if for each partition $\bar{x}, \bar{y}$ of variables in α, for each tuple $\bar{u}$ of nodes from t_w and each tuple $\bar{v}$ of nodes from $t - t_w$, if $t \models \alpha(\bar{u}, \bar{v})$, then $t \models \eta_\sim(\bar{u}, \bar{v})$.

Fix a partition $\bar{x}$, $\bar{y}$. By Lemma 2, the condition above is equivalent to: for all tuples $\bar{u}, \bar{u}'$ of nodes from t_w and all tuples $\bar{v}, \bar{v}'$ of nodes from $t - t_w$, if $t \models \alpha(\bar{u}, \bar{v})$ and $t \models \alpha(\bar{u}', \bar{v}')$, then $t \models \eta_\sim(\bar{u}, \bar{v}')$. Let us turn this into a condition on stored data values. Define $\eta(\bar{x}, \bar{y})$ as the formula obtained from $\eta_\sim(\bar{x}, \bar{y})$ by replacing $\sim$ with $=$, and $d(z)$ with $z = d$ for all variables z and all $d \in \mathbb{D}$. Reformulating the condition above we obtain: for each tuple $\bar{u}$ of nodes from t_w such that $t \models \alpha(\bar{u}, \bar{v})$ for some tuple $\bar{v}$ of nodes from $t - t_w$, the tuple $\mathsf{val}_t(\bar{u})$ of data values belongs to the set

$$Z_{\alpha(\bar{x},\bar{y}) \Rightarrow \eta_\sim(\bar{x},\bar{y})} = \bigcap_{\bar{v}'} \left\{ \bar{c} \in \mathbb{D}^{|\bar{x}|} \;\middle|\; \eta(\bar{c}, \mathsf{val}_t(\bar{v}')) \right\},$$

where $\bar{v}'$ ranges over tuples of nodes from $t - t_w$ satisfying $t \models \alpha(\bar{u}', \bar{v}')$ for some tuple $\bar{u}'$ of nodes from t_w.

Writing $\eta(\bar{x}, \mathsf{val}_t(\bar{v}'))$ in the disjunctive normal form, we see that the set $\left\{ \bar{c} \in \mathbb{D}^{|\bar{x}|} \;\middle|\; \eta(\bar{c}, \mathsf{val}_t(\bar{v}')) \right\}$ is a union of subspaces of $\mathbb{D}^{|\bar{x}|}$. How many subspaces? The canonical definition of each nonempty subspace has for each coordinate i either an equality $x_i = x_j$ for some $j > i$, or an equality $x_i = d$ for some $d \in \mathbb{D}$, or nothing. In our case, d is a data value used explicitly in η or occurring in the data tuple $\mathsf{val}_t(\bar{v}')$. Consequently, the number of these subspaces can be bounded by $(N + |\bar{x}| + |\bar{y}|)^{|\bar{x}|}$, where N is the number of data values used explicitly in η. That is, $Z_{\alpha(\bar{x},\bar{y}) \Rightarrow \eta_\sim(\bar{x},\bar{y})}$ is an intersection of unions of at most $(N + |\bar{x}| + |\bar{y}|)^{|\bar{x}|}$ subspaces of $\mathbb{D}^{|\bar{x}|}$. By Lemma 1, it can be represented as a union of at most $(N + |\bar{x}| + |\bar{y}|)^{|\bar{x}|^2}$ subspaces. In the canonical definition of each of these subspaces, there are at most $|\bar{x}|$ equalities of the form $x_i = d$ for $d \in \mathbb{D}$. That is, we can define $Z_{\alpha(\bar{x},\bar{y}) \Rightarrow \eta_\sim(\bar{x},\bar{y})}$ using explicitly at most $|\bar{x}| \cdot (N + |\bar{x}| + |\bar{y}|)^{|\bar{x}|^2}$ data values. From this we shall derive a bound on the number of important data values in t_w that ensure satisfaction of $\Sigma_\sim$, and conclude that we can safely replace others with fresh ones.

Let $\mathsf{val}' : \mathsf{dom}_t \to \mathbb{D}$ be a new data labelling of t. As we are only going to change data values in t_w, keeping a constraint $\alpha \Rightarrow \eta_\sim$ satisfied requires only

that for each partition $\bar{x}, \bar{y}$ of its variables, for each tuple $\bar{u}$ of nodes from t_w, if $t \models \alpha(\bar{u}, \bar{v})$ for some tuple $\bar{v}$ of nodes from $t - t_w$, then $\mathsf{val}'(\bar{u}) \in Z_{\alpha(\bar{x},\bar{y})\Rightarrow\eta_\sim(\bar{x},\bar{y})}$. Moreover, replacing all occurrences of a data value d in t_w with a given data value d' does not affect equalities of the form $x_i = x_j$ in the canonical definition of the set $Z_{\alpha(\bar{x},\bar{y})\Rightarrow\eta_\sim(\bar{x},\bar{y})}$. We only need to ensure that equalities of the form $x_i = d$ are not violated. Let $D \subseteq \mathbb{D}$ be the set of data values occurring in these equalities for all sets $Z_{\alpha(\bar{x},\bar{y})\Rightarrow\eta_\sim(\bar{x},\bar{y})}$, with $\alpha(\bar{x}, \bar{y}) \Rightarrow \eta_\sim(\bar{x}, \bar{y})$ ranging over constraints from $\Sigma_\sim$ with all possible partitions of variables. A labelling val' that replaces each data value from $\mathbb{D} - D$ used in t_w with a fresh data value does not violate $\Sigma_\sim$. For each constraint $\alpha \Rightarrow \eta_\sim$ there is at most 2^ℓ partitions $\bar{x}, \bar{y}$ of variables; each partition corresponds to a set $Z_{\alpha(\bar{x},\bar{y})\Rightarrow\eta_\sim(\bar{x},\bar{y})}$, which contributes at most $\left(\ell \cdot (m + \ell)^{\ell^2}\right)$ data values, where ℓ and m are the maximal numbers of variables and predicates from $\mathbb{D} \cup \check{\mathbb{D}}$ in constraints from $\Sigma_\sim$. Hence, we have $|D| \leq |\Sigma_\sim| \cdot 2^\ell \cdot \left(\ell \cdot (m + \ell)^{\ell^2}\right)$. □

3.3 From bounded data cut to automata

In order to use automata, we need to encode data trees of bounded data cut as trees over a finite alphabet. Let $C, D \subseteq \mathbb{D}$ be two disjoint finite sets of data values; we shall call elements of C *colours* and elements of D *distinguished* data values. As encodings we shall use trees over the alphabet

$$\Gamma \times (C \cup D) \times \mathbb{P}(C)\,;$$

we shall refer to the values in the third component of label as *refresh sets*. The distinguished data values (corresponding to data values used explicitly in the constraints) are represented explicitly in the encoding. The remaining data values are represented implicitly by colours and refresh sets: two nodes u, v store the same data value if and only if they have the same colour c and there is no node w such that $u \in t_w$ and $v \in t - t_w$ (or symmetrically), and the refresh set for w contains c. We define the semantics of the encoding slightly more generally: to every forest f over the alphabet $\Gamma \times (C \cup D) \times \mathbb{P}(C)$ we associate a data forest $\widehat{f}$. It has the same domain as f, the structure and labelling with elements of Γ is inherited from f, and the assignment of data values is defined inductively based on the remaining two components of the labelling of f. If the forest f is empty, so is $\widehat{f}$. Otherwise, let us decompose f into a forest f' followed by a tree further decomposed into the root and a forest f'' (see Figure 2); both f' and f'' may be empty. Assume the root has label (a, d, R). The forest $\widehat{f}$ is obtained by plugging $\widehat{f''}$ under a root with label a and data value d, appending the resulting tree to $\widehat{f'}$, and then replacing each colour from the refresh set R used in the resulting forest with a globally fresh data value from $\mathbb{D} - (C \cup D)$. Note that $\widehat{f}$ is unique up to permutations of $\mathbb{D} - (C \cup D)$. By construction, $\mathit{datacut}(\widehat{f}) \leq |C \cup D|$.

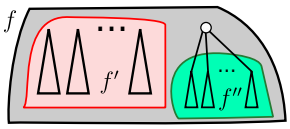

Fig. 2 Forest f decomposed into a forest f' followed by a tree further decomposed into the root and a forest f''.

Lemma 4 *For each data forest f of data cut n and all finite disjoint sets $C, D \subseteq \mathbb{D}$ such that*

$$|C| > \frac{3}{2} \cdot n\,,$$

there exists a forest g over $\Gamma \times (C \cup D) \times \mathbb{P}(C)$ such that $\widehat{g} = f$ up to a permutation of $\mathbb{D} - D$.

Proof Since we only aim at equality modulo a permutation of $\mathbb{D} - D$, we may assume that no data value from C is used in f. As the structure of g must be identical to that of f, we only need to define the labelling. Moreover, in the label (a, d, R) of a node w, we must always have $a = \mathsf{lab}_f(w)$. The remaining two components, d and R, are defined in the course of a procedure processing nodes in the usual bottom-up, left-to-right order, maintaining the following invariants:

1. $\widehat{g_w} = f_w$ up to a bijection i_w between the colours from C used in $\widehat{g_w}$ and the data values from $\mathbb{D} - D$ shared by f_w and $f - f_w$;
2. if v is the last child of the next sibling of w, then i_v and i_w coincide over $\mathsf{dom}(i_v) \cap \mathsf{dom}(i_w)$, and $|\mathsf{dom}(i_v) \cup \mathsf{dom}(i_w)| \leq \frac{3}{2} \cdot n$.

For a node w the procedure first sets the values d and R in such a way that the first invariant is satisfied, and then applies a permutation of C to the whole g_w to ensure the second invariant. Note that applying such a permutation affects $\widehat{g_w}$, but does not violate the first invariant.

Let w' and w'' be the previous sibling and the last child of w (if some of these do not exist, the argument is adjusted easily). For d there are three cases:

- if $\mathsf{val}_f(w) \in D$, set $d = \mathsf{val}_f(w)$;
- if $\mathsf{val}_f(w) = i_{w'}(c)$ or $\mathsf{val}_f(w) = i_{w''}(c)$ for some colour $c \in C$, set $d = c$;
- otherwise, set $d = c$ for an arbitrary colour $c \in C - \big(\mathsf{dom}(i_{w'}) \cup \mathsf{dom}(i_{w''})\big)$, which exists by the second invariant.

The refresh set R contains each colour $c \in C$ currently used in $\widehat{g_w}$, that represents a data value occurring in f_w but not in $f - f_w$. After these colours have been refreshed, all colours $C_0 \subseteq C$ used in $\widehat{g_w}$ represent different data values shared by f_w and $f - f_w$; the bijection i_w can be defined by restricting to C_0 the union of $i_{w'}$, $i_{w''}$, and $\{(d, \mathsf{val}_f(w))\}$ if $d \in C_0 - \big(\mathsf{dom}(i_{w'}) \cup \mathsf{dom}(i_{w''})\big)$.

If w has no next sibling or the next sibling has no children, we are done. Otherwise, let v be the last child of the next sibling u of w. We need a permutation π of C such that i_v and the updated bijection $i_w \circ (\pi \restriction C_0)^{-1}$ satisfy

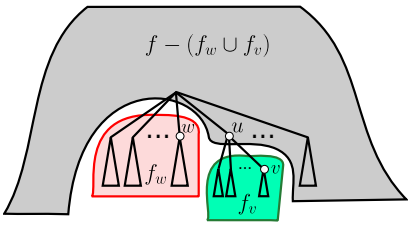

Fig. 3 The positioning of f_w, f_v, and $f - (f_w \cup f_v)$ in a forest f, when v is the last child of the next sibling u of w.

the second invariant. Let $W, V, U \subseteq \mathbb{D} - D$ be the sets of data values used, respectively, in the fragments f_w, f_v, and $f - (f_w \cup f_v)$ shown in Figure 3, and let $k = |W \cap U - V|$, $\ell = |V \cap U - W|$, $m = |W \cap V - U|$, and $r = |W \cap V \cap U|$. By the definition of data cut applied to w, v, and u, we have

$$k + m + r \leq n\,, \qquad \ell + m + r \leq n\,, \qquad k + \ell + r + e \leq n\,,$$

where $e = 1$ if the data value in u is used in $f - f_u$ and does not belong $V \cup W$, and $e = 0$ otherwise. In order to represent values in $W \cap U$, $V \cap U$, and $W \cap V$, we need exactly $k + \ell + m + r$ colours. By adding the three inequalities above, we obtain that $k + \ell + m + r \leq \frac{3n-r-e}{2}$. Hence, $\frac{3}{2} \cdot n$ colours are sufficient to accommodate the domains of i_v and the updated bijection $i_w \circ (\pi \restriction C_0)^{-1}$.

As the bounds in the three inequalities above can be attained simultaneously, with $r = 0$ and $e = 0$, $\frac{3}{2} \cdot n$ colours are also necessary. Consequently, the assumption in the lemma is tight, because if the data value in the node w does not occur anywhere else, we need one more colour to represent it. □

Let $\Sigma_\sim \cup \Sigma_{\not\sim}$ be a set of non-mixing integrity constraints and let $\mathcal{A}$ be a tree automaton. By Lemma 3, it is enough to test satisfiability of $\Sigma_\sim \cup \Sigma_{\not\sim}$ over trees of data cut bounded by a number n, singly exponential in the total size of constraints in $\Sigma_\sim \cup \Sigma_{\not\sim}$. Let $D \subseteq \mathbb{D}$ be the set of data values used explicitly in $\Sigma_\sim \cup \Sigma_{\not\sim}$, and let $C \subseteq \mathbb{D} - D$ be a fixed set such that $|C| = \left\lfloor \frac{3}{2} \cdot n \right\rfloor + 1$. By Lemma 4, each tree of data cut bounded by n can be encoded as a tree over $\Gamma \times (C \cup D) \times \mathbb{P}(C)$ up to a permutation of $\mathbb{D} - D$. Since such permutations do not affect relations used in $\Sigma_\sim \cup \Sigma_{\not\sim}$, there exists a data tree accepted by $\mathcal{A}$ and satisfying $\Sigma_\sim \cup \Sigma_{\not\sim}$ if and only if there exists a tree t such that the data tree $\widehat{t}$ is accepted by $\mathcal{A}$ and satisfies $\Sigma_\sim \cup \Sigma_{\not\sim}$. We reduce consistency of $\Sigma_\sim \cup \Sigma_{\not\sim}$ to the emptiness problem for tree automata, by constructing an automaton that recognizes such trees t.

The automaton is obtained by taking the product of two automata, testing acceptance by $\mathcal{A}$ and satisfaction of $\Sigma_\sim \cup \Sigma_{\not\sim}$, respectively. The first automaton is just the automaton $\mathcal{A}$ lifted to the product alphabet $\Gamma \times (C \cup D) \times \mathbb{P}(C)$: it looks only at the first component of each label. Note that already this automaton has doubly exponential size, because of the size of the alphabet. The second automaton is the product over all constraints $\sigma \in \Sigma_\sim \cup \Sigma_{\not\sim}$ of automata $\mathcal{B}_\sigma$ recognizing trees t such that $\widehat{t} \models \sigma$, which will be constructed in

the next subsection. Each automaton $\mathcal{B}_\sigma$ is doubly exponential and so is the whole construction. As the emptiness problem for tree automata is in PTime, we can conclude that the consistency problem is in 2ExpTime.

3.4 Translating constraints to automata over encodings

To complete the proof of the upper bound of Theorem 1, it remains to construct an automaton recognizing encodings of trees that satisfy a given constraint.

Let us fix a constraint $\alpha(\bar{x}) \Rightarrow \eta(\bar{x})$ with $\bar{x} = (x_1, x_2, \ldots, x_\ell)$; for the present construction, it needs not to be non-mixing. Let $D \subseteq \mathbb{D}$ be a finite set of data values, containing each data value used explicitly in $\eta(\bar{x})$, and let $C \subseteq \mathbb{D} - D$ be an arbitrary finite set. We shall construct an automaton $\mathcal{B}$ over the alphabet $\Gamma \times (C \cup D) \times \mathbb{P}(C)$ recognizing the language

$$\left\{ t \;\middle|\; \widehat{t} \models \alpha(\bar{x}) \Rightarrow \eta(\bar{x}) \right\} .$$

It will read a tree t, compute a representation of tuples selected from the associated data tree $\widehat{t}$ by the selector query $\alpha(\bar{x})$, and accept if all these tuples satisfy the assertion $\eta(\bar{x})$. The representation of the selected tuples will be computed based on the maintained information about partial matchings of the selector query in the forest encoded by the processed part of the tree. This can be done in the usual way, except that we need to systematically refresh the colours, as specified in t. To explain the details, we need some auxiliary notions.

Recall that t and $\widehat{t}$ have the same domain, structure, and labelling with elements of Γ; the only difference lies in the way data values are represented: encoded in t and explicit in $\widehat{t}$. Consequently, as long as we do not care about data values, we can blur the distinction between the encoded and decoded data tree. Similarly, t_w, $\widehat{t_w}$, and $\left(\widehat{t}\right)_w$ are the same forest, up to the representation of data values. A *partial valuation* of variables $x_1, x_2, \ldots, x_\ell$ is a function

$$g \colon \{x_1, x_2, \ldots, x_\ell\} \to \mathsf{dom}_t \cup \{\bot\} .$$

If $g(x_i) \neq \bot$, we say that x_i is *matched* at $g(x_i)$, and if $u_i = \bot$ we say that x_i is *not matched*. Two partial valuations of $x_1, x_2, \ldots, x_\ell$ are *disjoint*, if no variable is matched by both of them. The *union* of disjoint partial valuations g, h of variables $x_1, x_2, \ldots, x_\ell$ is given as

$$(g \cup h)(x_i) = \begin{cases} g(x_i) & \text{if } g(x_i) \neq \bot , \\ h(x_i) & \text{otherwise} . \end{cases}$$

Recall that $\alpha(\bar{x})$ is a conjunction of atoms. A *partial matching* of $\alpha(\bar{x})$ in $\widehat{t_w}$ is a partial valuation g of variables $\bar{x}$ such that

- variables are matched only in the nodes of t_w;
- each atom in $\alpha(g(\bar{x}))$ that does not contain $\bot$ holds true in $\widehat{t_w}$; and

– each atom that contains both a node from t_w and $\bot$ is of the form

$$w \to \bot, \quad w' \to^+ \bot, \quad \bot \downarrow w', \quad \text{or} \quad \bot \downarrow^+ v\,,$$

where w' is a preceding sibling of w or w itself, and v is an arbitrary node of t_w.

The last condition means that each such atom can be made true (independently of others) by replacing $\bot$ with a node from $t - t_w$, unless w has no following siblings or no ancestors in t.

If $\widehat{t} \models \alpha(\bar{u})$, each partial valuation matching a subset of variables x_i at nodes u_i from $\widehat{t_w}$ is a partial matching of α. Conversely, if a partial matching g matches all variables $\bar{x}$, then $\widehat{t} \models \alpha(g(\bar{x}))$. Note, however, that not every partial matching can be extended so that it matches all variables: remaining atoms may be satisfiable on their own, but not together.

The automaton collects information about tuples selected by $\alpha(\bar{x})$ node by node: when it is in a node w of the input tree t, it has information corresponding to partial matchings of $\alpha(\bar{x})$ in $\widehat{t_w}$. More precisely, the states of the automaton $\mathcal{B}$ are subsets Δ of

$$(C \cup D \cup \{\top_1, \top_2, \ldots, \top_\ell, \bot\})^\ell\,.$$

Each such tuple represents a partial matching of $\alpha(\bar{x})$ in $\widehat{t_w}$, and the whole Δ represents a set of such partial matchings. The intended meaning of the symbolic values is as follows:

– $c \in C \cup D$ in the coordinate j of the tuple means that the variable x_j is matched in a node of $\widehat{t_w}$ storing the data value c;
– $\top_i$ means that the variable x_j is matched in a node storing some data value $d_j \in \mathbb{D} - (C \cup D)$, where $d_1, d_2, \ldots, d_\ell$ are distinct and depend on the tuple;
– $\bot$ means that variable x_j has not been matched yet.

The initial state is $\{(\bot, \bot, \ldots, \bot)\}$. The accepting states are the ones whose each tuple either contains $\bot$ or satisfies the assertion $\eta(\bar{x})$.

Let us describe the transition relation. Assume that automaton $\mathcal{B}$ is about to determine the state in a node w. Let w' and w'' be, respectively, the previous sibling and the last child of w. The set of partial matchings of $\alpha(\bar{x})$ in $\widehat{t_w}$ depends only on the sets of partial matchings in $\widehat{t_{w'}}$ and $\widehat{t_{w''}}$, and the label of w. Indeed, a partial valuation of $\bar{x}$ is a partial matching of $\alpha(\bar{x})$ in $\widehat{t_w}$ if it is the union of disjoint partial matchings of $\alpha(\bar{x})$ in $\widehat{t_{w'}}$ and $\widehat{t_{w''}}$ possibly extended by matching some (yet unmatched) variables at node w, respecting two conditions. For all atoms $x_i \to x_j$, $x_i \to^+ x_j$ in $\alpha(\bar{x})$, either x_i, x_j are both matched in $\widehat{t_{w''}}$ or none is; and the new matching of variables at w does not violate the definition of partial matching. The latter can be expressed as follows:

– if $\alpha(\bar{x})$ contains $x_i \downarrow x_j$ or $x_i \downarrow^+ x_j$, we may match x_i at w only if x_j is matched in $t_{w''}$; for $x_i \downarrow x_j$, if x_j is matched, we must match x_i, unless it is matched in $t_{w''}$ already;

- if $\alpha(\bar{x})$ contains $x_i \to x_j$ or $x_i \to^+ x_j$, we may match x_j at w only if x_i is matched in $t_{w'}$; for $x_i \to x_j$, if x_i is matched, we must match x_j, unless it is matched in $t_{w'}$ already;
- if $\alpha(\bar{x})$ contains $a(x_i)$, we may match x_i at w only if $\mathsf{lab}_t(w) = (a, c, R)$ for some c, R.

Checking the conditions above requires only information about which variables are matched in $\widehat{t_{w'}}$ and $\widehat{t_{w''}}$; the used tree nodes are not relevant. Consequently, one can determine the set of tuples representing partial matchings in $\widehat{t_w}$ based on the sets of tuples representing partial matchings in $\widehat{t_{w'}}$ and $\widehat{t_{w''}}$, and the label (a, c, R) of the current node w. Notice that the symbolic values $\top_i$ represent different data values in $\widehat{t_{w'}}$ and $\widehat{t_{w''}}$, so before combining two tuples we rename these values to guarantee that none is used in both tuples (ℓ values are always sufficient for this). The final step is to refresh colours: in each tuple we replace all occurrences of $c \in R$ with some $\top_i$ not yet used in this tuple.

3.5 Lower bound

Lemma 5 *Consistency of non-mixing constraints is* 2ExpTime*-hard.*

Proof Relying on the fact that 2ExpTime = AExpSpace, we will be using alternating Turing machines. Such machines can be defined in multiple similar ways, and we use a definition that is most convenient for our encoding. We do not divide states of our machine into existential and universal; we only distinguish accepting states. Instead, we use the following notion of a run tree, requiring that from every configuration two different transitions can be applied. A *run tree* of an alternating Turing machine M on input word w is a tree labelled by configurations of M, where

- the root is labelled by the initial configuration for the input word w;
- every node not labelled by an accepting configuration has exactly two children, labelled by successors of this configuration, reached by applying to it two different transitions;
- every node labelled by an accepting configuration is a leaf.

We say that an input word w is accepted by M if there is a finite run tree of M for w.

To turn a standard machine with existential and universal states into a machine of the form above, one simply ensures that in universal states the machine has exactly two available transitions, and duplicates transitions available in existential states.

Consider an alternating Turing machine M (of the form described above) that works in space bounded by $2^{|w|}$, where w is the input word. Note that we limit the space to $2^{|w|}$ instead of considering any exponential function, but already among such machines there is one solving an AExpSpace-hard problem. We show that for every input word w we can construct (in polynomial time) a tree automaton $\mathcal{A}$ and a set Σ of non-mixing constraints such that $\mathcal{A}$

and Σ are consistent if and only if M accepts w. More precisely, every tree $t \in L(\mathcal{A})$ such that $t \models \Sigma$ describes a run tree of M on w. Below we specify how such a tree t encodes a run tree of M on w, simultaneously saying how these properties are ensured by $\mathcal{A}$ or Σ.

Nodes labelled by s form a prefix of t that is a binary tree: the parent of every s-labelled node (if exists) is s-labelled, and every s-labelled node has zero or two s-labelled children. This is ensured by the automaton $\mathcal{A}$. This part of t is called the *skeleton*, and will have the same shape as the run tree.

Additionally, each node of the skeleton has a c-labelled child (in addition to the zero or two children from the skeleton). The subtree rooted in this child forms a path, whose labels match the regular expression

$$\mathsf{c}\, A\, Q\, A\, Q\, (\mathsf{l} + \mathsf{n} + \mathsf{r})\, (\$\, (\mathsf{h} + \mathsf{n})\, (\mathsf{p} + \mathsf{n})\, \mathsf{c}^{2n+1})^+ \#\,,$$

where $\mathsf{b}, \mathsf{l}, \mathsf{n}, \mathsf{r}, \mathsf{h}, \mathsf{p}$ are new alphabet symbols, Q and A are the state set and the tape alphabet of M, and $n = |w|$, Again, this is ensured by $\mathcal{A}$. Such path, called a *configuration path*, describes a configuration of M assigned to the corresponding node of the skeleton (which is also a node of the run tree).

At the beginning of each configuration path we have the transition used to reach this configuration: the second node is labelled by the letter present on the tape under the head in the previous configuration; the third node is labelled by the previous state; the fourth by the letter written on the tape; the fifth by the new state; the sixth by the direction in which the head was moved (left, no move, right). The automaton ensures that this is indeed a valid transition of M (except for the configuration path directly below the root, where we only ensure that the fifth node contains the initial state); that the label of the third node (previous state) is equal to the label of the fifth node (current state) of the parent configuration; that the transitions assigned to sibling configurations are different (as required in the definition of a run tree); that states are accepting in leaf configurations and not accepting in non-leaf configurations.

The next part of a configuration path consists of multiple *blocks* of length $2n+4$; each of them describes a single letter on the tape. To identify a block, we use the first n c-labelled nodes for a binary counter encoding the position in the tape, using data values $0, 1 \in \mathbb{D}$. We assign data values $0, \ldots, 0, 0$ to these nodes in the first block, $0, \ldots, 0, 1$ in the second block, and so on, until $1, \ldots, 1, 1$ in the last block (we have 2^n blocks, which equals to the length of the tape). The next n nodes of the block also contain such a counter, but going back: we assign $1, \ldots, 1, 1$ to these nodes in the first block, and $0, \ldots, 0, 0$ in the last block. Notice that when one counter of a block contains bits $b_1, \ldots, b_n$, then the other counter contains their inverses $1-b_1, \ldots, 1-b_n$. This double encoding of the position is the key trick that allows using non-mixing integrity constraints to check correctness of the run between two consecutive configurations. To enforce this behaviour of counters we use constraints. In the constraints we shall use queries matching tuples of variables

$$\bar{x} = (x_\$, x_h, x_p, x_1, \ldots, x_n, x'_1, \ldots, x'_n, x_d)$$

to the $2n + 4$ consecutive nodes of blocks in configuration paths. It is easy to write a conjunctive query $\alpha_{fb}(\bar{x})$ that matches $\bar{x}$ to the first block in any configuration path. We include in Σ the constraint

$$\alpha_{fb}(\bar{x}) \Rightarrow 0(x_1) \wedge \cdots \wedge 0(x_n) \wedge 1(x'_1) \wedge \cdots \wedge 1(x'_n)\,.$$

We deal analogously with the last block. Then, using a conjunctive query $\alpha_{cb}(\bar{x}, \bar{y})$ with $\bar{y}$ defined as $\bar{x}$ above, that matches two consecutive blocks, we include the constraint

$$\alpha_{cb}(\bar{x}, \bar{y}) \Rightarrow \eta_{\sim}(\bar{x}, \bar{y})$$

ensuring that the counters in these two blocks encode consecutive numbers. It is a standard task to express this property as a positive Boolean combination $\eta_{\sim}(\bar{x}, \bar{y})$ of atoms over $\{\sim, 0, 1\}$, of a quadratic size.

The second node of each block is marked by h if the head of M is placed over this position of the tape, and the third node is marked by p if the head was placed over this position in the previous configuration. The automaton ensures that in each configuration path exactly one block is marked by h and exactly one block is marked by p; that in the initial configuration the head is over the first letter; that the relation between the p and h markers on a configuration path is as described by the sixth node of that path (l, n, or r). To ensure that the position of p corresponds to the position of h in the previous configuration we use the constraint

$$\alpha_{ch}(\bar{x}, \bar{y}) \Rightarrow x_1 \sim y_1 \wedge \cdots \wedge x_n \sim y_n\,,$$

where $\alpha_{ch}(\bar{x}, \bar{y})$ matches $\bar{x}$ to the h-marked block of a configuration and $\bar{y}$ to the p-marked block of a child configuration.

The last node of each block carries the tape letter (from A) in the data value. To ensure that the initial configuration starts with the input word, we write a constraint

$$\alpha_{ini}(x_1, \ldots, x_n) \Rightarrow \eta_{\sim}(x_1, \ldots, x_n)\,,$$

where $\alpha_{ini}(x_1, \ldots, x_n)$ selects the last node from each of the first n blocks of the topmost configuration path (to make sure that only the topmost configuration path is selected, we can check for the presence of the initial state, assuming w.l.o.g. that M cannot reach the initial state in any transition). Another constraint

$$\alpha_{bl}(x) \Rightarrow \mathsf{b}(x)$$

ensures that the rest of the initial tape contains blanks $\mathsf{b} \in A$, where $\alpha_{bl}(x)$ matches the last node of a block of the topmost configuration path other than the first n blocks. Next $|A|$ constraints ensure that the p-marked block contains the letter written in the fourth node of the configuration path (letter written under the head), and another $|A|$ constraints that the h-marked block of the previous configuration contains the letter written in the second node of the configuration path (letter seen under the head).

Finally, we have to ensure that the content of the tape is preserved (except the single letter under the head). Let $\alpha_{2b}(\bar{x}, \bar{y})$ be a conjunctive query matching some blocks on consecutive configuration paths, where the first of them is not marked by h. For every such pair $\bar{x}, \bar{y}$, we want to enforce that either the two corresponding blocks carry the same letter or they represent two different positions in the tape. Using the double complementary encoding of the position in the blocks, this can be enforced using only $\sim$ in the following constraint:

$$\alpha_{2b}(\bar{x}, \bar{y}) \Rightarrow x_1 \sim y'_1 \vee \cdots \vee x_n \sim y'_n \vee x_d \sim y_d \,.$$

Notice that the property "$\bar{x}$ and $\bar{y}$ encode different positions in the tape" seems to require $\nsim$, but thanks to the inverted counter stored in each block we may use $\sim$ instead, avoiding the illegal mixture of $\sim$ and $\nsim$ in the assertion of the constraint.

By construction, witnesses for the obtained automaton $\mathcal{A}$ and set of constraints Σ correspond to run trees of the machine M, which ensures correctness of the reduction. □

We remark that all conjunctive queries used in the above proof could be written using tree patterns (see Section 4.2 for the definition), and that the set $\Sigma_{\nsim}$ was empty. Thus the 2ExpTime-hardness result holds already for constraints of this form. If we only allow tree patterns as selectors and $\Sigma_{\sim}$ is empty, the complexity might be lower. In Section 4.3 we shall see a different hardness argument, showing that there is no hope for lower complexity without restricting selectors.

4 Extensions, connections, and applications

4.1 Entailment of non-mixing constraints

A static analysis problem more general than consistency is entailment. Recall that a set of constraints Σ' is *entailed* by a set of constraints Σ modulo a tree automaton $\mathcal{A}$, written as $\Sigma \models_{\mathcal{A}} \Sigma'$, if for each data tree t accepted by automaton $\mathcal{A}$,

$$t \models \Sigma \quad \text{implies} \quad t \models \Sigma' \,.$$

The entailment problem is then defined as follows:

Problem:	**Entailment problem for non-mixing constraints**
Input:	Sets Σ, Σ' of non-mixing constraints, tree automaton $\mathcal{A}$.
Question:	$\Sigma \models_{\mathcal{A}} \Sigma'$?

Entailment is a more general problem than consistency, but for non-mixing constraints the results on consistency generalize to entailment almost effortlessly.

Theorem 2 *Entailment of non-mixing constraints is* 2ExpTime*-complete.*

Proof Inconsistency is a special case of entailment: Σ is inconsistent with respect to an automaton $\mathcal{A}$ if and only if $\Sigma \models_{\mathcal{A}} \bot$, where $\bot$ is an inconsistent set of constraints, say $\{a(x) \Rightarrow 0(x) \wedge 1(x) \mid a \in \Gamma\}$. Thus, the lower bound follows.

Lemma 3 shows that witnesses for consistency can have bounded data cut. The same is true for counter-examples to entailment. Suppose $t \models \Sigma$ and $t \not\models \Sigma'$. Then, $t \models \alpha'(\bar{u}) \wedge \neg\eta'(\bar{u})$ for some constraint $\alpha'(\bar{x}) \Rightarrow \eta'(\bar{x})$ from Σ' and some tuple $\bar{u}$ of nodes of t. Let D_0 be the set of data values used in the nodes $\bar{u}$. We can repeat the construction of the tree t' word for word, except that we replace the set D of values not to be touched by $D \cup D_0$. This increases $\mathit{datacut}(t')$ by the maximal number of variables in the constraints of Σ'.

The automata construction in Section 3.3 is modified similarly: the set D contains also the data values used explicitly in Σ', in the product automaton we include additionally the automata $\mathcal{B}_\sigma$ for $\sigma \in \Sigma'$, and we let it accept if at least one of these components rejects and all previously described components accept. As the automata $\mathcal{B}_\sigma$ are deterministic, this does not involve any additional cost.

Note that the argument above works also if Σ' mixes predicates from $\mathsf{sig}_\sim$ and $\mathsf{sig}_\nsim$. □

4.2 A singly exponential fragment

A closer look at the complexity of our algorithm reveals that it is doubly exponential only in the maximal number ℓ of variables in the constraints. This number appears in three roles: in the exponent in the factors $(\ell + m)^{\ell^2}$ and 2^ℓ of the bound on the data cut, and as the length of tuples representing partial matchings of selectors. A slightly more detailed analysis of the proof of Lemma 3 shows that in the first role, ℓ could be replaced by the maximal number ℓ' of variables actually used in the assertions. Indeed, since data equalities involve only variables occurring in the atoms of the assertions, everything is in fact happening in a space of dimension at most ℓ'. While limiting the size of selector queries to lower the complexity makes little sense, limiting the number of variables actually used in assertions seems acceptable. But what about the other two roles of ℓ?

Concerning the third role, the need to represent all partial matchings (up to data equality type) comes from the fact that the automaton is essentially evaluating conjunctive queries. The standard technique to lower the complexity in such cases is to replace conjunctive queries with tree patterns, which are essentially tree-structured conjunctive queries. In the most basic form, with only $\downarrow$ and $\downarrow^+$ axes allowed, a *tree pattern* is a conjunctive query α over signature $\{\downarrow, \downarrow^+\} \cup \Gamma$, such that graph

$$(A_\alpha, \downarrow_\alpha \cup \downarrow^+_\alpha)$$

is a directed tree, where

$$\mathbb{A}_\alpha = (A_\alpha, \downarrow_\alpha, \downarrow^+_\alpha, \{a_\alpha\}_{a \in \Gamma})$$

is the canonical relational structure associated to query α in the usual way: the universe A_α is the set of variables of α, and relations are given by the respective atoms in α.

Finally, we also have the factor 2^ℓ in the bound on the data cut. This factor appears because in the proof of Lemma 3, we consider separately each partition of variables into two tuples. As we shall see, the number of partitions can also be reduced for tree patterns. To this end, we prove the following analogue of Lemma 2.

Lemma 6 *Let $\alpha(\bar{x}, \bar{y}, \bar{z})$ be a tree pattern, where $\bar{x}$, $\bar{y}$, and $\bar{z}$ are pairwise disjoint, and in $\downarrow_\alpha \cup \downarrow_\alpha^+$ there are no edges from variables in $\bar{z}$ to variables in $\bar{x}$, $\bar{y}$. Let w be a node of a data tree t. For all tuples $\bar{u}, \bar{u}'$ of nodes from t_w, and tuples $\bar{v}, \bar{v}'$ of nodes from $t - t_w$, if*

$$t \models \exists \bar{z}\, \alpha(\bar{u}, \bar{v}, \bar{z}) \quad \text{and} \quad t \models \exists \bar{z}\, \alpha(\bar{u}', \bar{v}', \bar{z})\,,$$

then

$$t \models \exists \bar{z}\, \alpha(\bar{u}, \bar{v}', \bar{z}) \quad \text{and} \quad t \models \exists \bar{z}\, \alpha(\bar{u}', \bar{v}, \bar{z})\,.$$

Proof We only prove that $t \models \exists \bar{z}\, \alpha(\bar{u}, \bar{v}', \bar{z})$, as the other part is symmetric. Let $\bar{w}$ and $\bar{w}'$ be tuples of nodes from t such that $t \models \alpha(\bar{u}, \bar{v}, \bar{w})$ and $t \models \alpha(\bar{u}', \bar{v}', \bar{w}')$. The claim holds trivially if both $\bar{x}$ and $\bar{y}$ are empty. Assume that at least one of these tuples is nonempty. Then the root of the tree pattern belongs to $\bar{x}$ or to $\bar{y}$. Let $\bar{z} = (z_1, \ldots, z_k)$, $\bar{w} = (w_1, \ldots, w_k)$, $\bar{w}' = (w'_1, \ldots, w'_k)$. For $i \in \{1, \ldots, k\}$ we look at the nearest ancestor of z_i that is in $\bar{x}$ or in $\bar{y}$. If it is in $\bar{x}$, we take $w''_i = w_i$, otherwise $w''_i = w'_i$, and we define $\bar{w}'' = (w''_1, \ldots, w''_k)$.

We need to check that every atom of $\alpha(\bar{u}, \bar{v}', \bar{w}'')$ is satisfied in t. This is clear for unary atoms. Assume a binary atom involves a variable from $\bar{z}$ that is valuated as in $\bar{u}, \bar{v}, \bar{w}$. From the definition of $\bar{w}''$ it then follows that the other variable, being its child or its parent in the tree pattern, is also valuated as in $\bar{u}, \bar{v}, \bar{w}$. It follows that the atom is satisfied, because $t \models \alpha(\bar{u}, \bar{v}, \bar{w})$. We argue analogously for binary atoms with a variable from $\bar{z}$ valuated as in $\bar{u}', \bar{v}', \bar{w}'$. It remains to consider binary atoms involving only variables from $\bar{x}$ and $\bar{y}$, and this can be done as in the proof of Lemma 2. □

We remark that for non-mixing integrity constraints, restricting selectors to tree patterns alone does not suffice to lower the complexity: the reduction in Lemma 5 uses only such constraints (and no assertions over sig_∞). But together with the bound on the number of variables in assertions, it does suffice.

Proposition 1 *For non-mixing constraints whose selectors are tree patterns and whose assertions use constantly many variables, consistency and entailment are* ExpTime*-complete.*

Proof We first complete the proof that the data cut can be bounded polynomially. We have already argued that in the factor $\ell \cdot (\ell + m)^{\ell^2}$ of the bound given by Lemma 3 we can replace ℓ by the maximal number ℓ' of variables used in the assertions, which is assumed to be constant. It remains to deal

with the factor 2^ℓ. We show that the number of considered partitions can be limited by a polynomial.

Fix a tree t, its node w, and a constraint $\alpha \Rightarrow \eta_\sim$ in $\Sigma_\sim$. We say that a variable x used in α is *important* if either x or some descendant of x (in the sense of $\downarrow_\alpha \cup \downarrow_\alpha^+$) is used in $\eta_\sim$; otherwise x is *unimportant*. We shall partition only important variables: we write the tree pattern as $\alpha(\bar{x}, \bar{y}, \bar{z})$, where $\bar{x}, \bar{y}$ is a partition of important variables, and $\bar{z}$ contains all unimportant variables. Notice that in $\downarrow_\alpha \cup \downarrow_\alpha^+$ there are no edges from unimportant variables to important variables, and thus Lemma 6 can be used.

We additionally restrict ourselves to *tame* partitions, defined as follows: a partition $\bar{x}, \bar{y}$ of important variables is tame if in $\downarrow_\alpha \cup \downarrow_\alpha^+$ there are no edges from variables in $\bar{x}$ to variables in $\bar{y}$. This way we only prune empty cases, because in the proof of Lemma 3 we only valuate variables from $\bar{x}$ with nodes from t_w, and variables from $\bar{y}$ with nodes from $t - t_w$.

We thus have the following statement: $t \models \alpha \Rightarrow \eta_\sim$ if and only if for each tame partition $\bar{x}, \bar{y}$ of important variables in α, for each tuple $\bar{u}$ of nodes from t_w, each tuple $\bar{v}$ of nodes from $t - t_w$, and each tuple $\bar{w}$ of nodes of t, if $t \models \exists \bar{z}\, \alpha(\bar{u}, \bar{v}, \bar{z})$, then $t \models \eta_\sim(\bar{u}, \bar{v})$. This allows us to continue as in the proof of Lemma 3; the unimportant variables do not appear in $\eta_\sim$, so it is irrelevant whether they are valuated in t_w or in $t - t_w$. Finally, we observe that the number of tame partitions of important variables is polynomial in the size of the tree pattern, assuming that the number of variables used in $\eta_\sim$ is constant. Indeed, for each of the constantly many variables used in $\eta_\sim$, we only have to decide how many of its closest ancestors (including itself) are to be taken to $\bar{x}$ (the ancestors being farther are then taken to $\bar{y}$).

We have thus proved that the bound on the data cut is polynomial. Hence, the size of the set of colours C is also polynomial. It remains to optimize the automaton $\mathcal{B}_\sigma$ verifying a single constraint σ, assuming that the selector of σ is a tree pattern.

We use the standard method relying on the fact that subtrees of a tree pattern can be matched independently. By definition, the domain of a partial matching of a tree pattern is a collection of disjoint full subtrees of the pattern. Such a collection can be matched, if each of its elements can be matched independently; the information sufficient to represent all possible matchings is a set of subtrees that can be matched. For our purposes this is insufficient: we are interested not in just matching the selector, but in all tuples of data values that can be associated with the variables used in the assertion. This information cannot be stored separately for each subtree, as we are interested in the equalities and inequalities between data values assigned to variables in different subtrees; this is a property of a set of matched subtrees, and there can be ways of matching the same set that yield different equalities and inequalities. The solution is to treat subtrees with variables used in the assertion in a special way. The automaton remembers in each state a collection of subtrees without assertion variables and a collection of pairs consisting of

– a set of pairwise disjoint subtrees with assertion variables, and

– a tuple representing the associated data values (like before).

As the number of assertion variables is constant, the number of such sets and such tuples is polynomial. Hence, the whole automaton is singly exponential. This shows that both consistency and entailment are in ExpTime.

The lower bound follows immediately from ExpTime-hardness of consistency of schema mappings with trivially unsatisfiable right hand sides of dependencies [1, Proposition 18.2], which can be also reinterpreted as validity of unions of tree patterns modulo a given tree automaton. □

4.3 Static analysis of unions of conjunctive queries

Our results can be reinterpreted in the framework of static analysis of unions of conjunctive queries (UCQs). Note that

$$t \not\models \alpha(\bar{x}) \Rightarrow \eta(\bar{x}) \qquad \text{if and only if} \qquad t \models \exists \bar{x}\, \alpha(\bar{x}) \wedge \neg\eta(\bar{x})\,.$$

It follows immediately that the problem of validity of UCQs over signature sig_{dt} that never mix predicates from $\mathsf{sig}_{\sim}$ and $\mathsf{sig}_{\nsim}$—call them *non-mixing UCQs*—reduces in polynomial time to *inconsistency* of non-mixing constraints. Similarly, containment of such queries reduces to entailment of non-mixing constraints. The converse reduction is also possible, but it involves exponential blow-up when arbitrary Boolean combinations in assertions are rewritten in disjunctive normal form. This correspondence brings our results very close to the work by Björklund, Martens, and Schwentick on static analysis for UCQs over signature $\mathsf{sig}_{nav} \cup \{\sim, \nsim\}$ [6].

On one hand, our results immediately give the following new decidability result for the setting considered by Björklund, Martens, and Schwentick (constraints used in the lower bound of Lemma 5 can be rewritten without blow-up).

Theorem 3 *Over* $\mathsf{sig}_{nav} \cup \{\sim, \nsim\}$*, both validity of non-mixing UCQs and containment of UCQs in non-mixing UCQs (with respect to a given tree automaton) are* 2ExpTime*-complete.*

Results of Björklund, Martens, and Schwentick give 2ExpTime upper bound for containment (with respect to a tree automaton) in UCQs over $\mathsf{sig}_{nav} \cup \{\sim\}$ and UCQs over $\mathsf{sig}_{nav} \cup \{\nsim\}$. The original work is on CQs, but arguments for UCQs are the same [12]. Essentially, they amount to an observation that in counter-examples to containment of a query p in a query q, all data values can be set equal (in the case with $\nsim$) or different (in the case with $\sim$), except for a bounded number of them needed to witness satisfaction of p; such counter-examples can be easily encoded as trees over a finite alphabet, and recognized by an automaton evaluating p and q in the usual way. Theorem 3 extends both these results. Since we have both $\sim$ and $\nsim$ in query q, we cannot assume that all data values are equal, nor that all are different; our more involved approach

seems necessary. The third relevant result of [6] is that containment of p in q is 2ExpTime-complete under the assumption that p is a CQ over $\mathsf{sig}_{nav} \cup \{\sim\}$ and q is a CQ over $\mathsf{sig}_{nav} \cup \{\sim, \nsim\}$. It looks stronger than ours because query q can mix $\sim$ and $\nsim$. In fact, it is much weaker, depending entirely on the fact that q is a single CQ, not a UCQ. More specifically, the argument is as follows: if q uses $\nsim$, the answer is *yes* if and only if p is not satisfiable with respect to the tree automaton (otherwise p is satisfiable in a tree with all data values equal, and no such tree can satisfy q because of its $\nsim$ atoms); if q does not use $\nsim$, we are back in the case of UCQs over $\mathsf{sig}_{nav} \cup \{\sim\}$.

On the other hand, some results of Björklund, Martens, and Schwentick give a broader context to our results. They show that validity with respect to a given automaton is already 2ExpTime-complete for unions of conjunctive queries over signature sig_{nav}, that is, for trees without data. Consequently, restricting only assertions of non-mixing constraints would not lower the complexity. This is complementary to our lower bound of Lemma 5, which shows 2ExpTime-hardness for constraints using tree patterns as selectors. Hence, the only way to lower the complexity is to restrict both, selectors and assertions. Björklund, Martens, and Schwentick also show that for UCQs over $\mathsf{sig}_{nav} \cup \{\sim, \nsim\}$ validity is undecidable; this means that we cannot go beyond non-mixing assertions.

4.4 XML constraints

Non-mixing constraints form an instance of the general framework of XML-to-relational (X2R) constraints proposed by Niewerth and Schwentick [27], where selectors are arbitrary queries defining relations by selecting tuples of nodes and data values (in separate columns), and assertions are arbitrary relational constraints over the defined relations; the considered problem is entailment modulo schema. Our setting corresponds to a fragment in which selectors are conjunctive queries over sig_{nav} interpreted as queries selecting tuples of data values, assertions are positive quantifier-free formulas using constants and either $=$ or $\neq$, and schemas are tree automata. Niewerth and Schwentick investigate two classes of assertions: functional dependencies (FDs) and XML-key FDs (XKFDs). In an FD

$$A_1\, A_2\, \ldots\, A_m \to B\,,$$

$A_1, A_2, \ldots, A_m, B$ are arbitrary columns of the relation defined by the selector (each referring either to nodes or to data values); in an XKFD, B is required to be a node column. Our setting captures XKFDs, but not general FDs. Consider an X2R constraint given by a CQ $\alpha(x_1, \ldots, x_n)$ populating a table with tuples $(x_1, \ldots, x_n, @x_1, \ldots, @x_n)$, where $@x_i$ stands for the data value stored in the node represented by variable x_i, and an XKFD $x_1, \ldots, x_j, @x_{j+1}, \ldots, @x_{n-1} \to x_n$ (it makes no sense to use both x_i and $@x_i$

in the same constraint). Such constraint can be rewritten as

$$\alpha(x_1,\ldots,x_n) \wedge \alpha(x_1,\ldots,x_j,x'_{j+1},\ldots,x'_n) \wedge x_n \neq x'_n \;\Rightarrow \\ \Rightarrow\; x_{j+1} \nsim x'_{j+1} \vee \cdots \vee x_{n-1} \nsim x'_{n-1}$$

which can be turned into a set of five non-mixing constraints by replacing $x_n \neq x'_n$ with simple subqueries describing possible ways of arranging two different nodes in a tree, as explained in Section 2.3. Note that these constraints do not use $\sim$. Hence, for XKFDs with UCQs over sig_{nav} as tuple selectors decidability of entailment follows already from the results on containment of UCQs over $\mathsf{sig}_{nav} \cup \{\sim\}$, discussed in the previous subsection; the challenge tackled by Niewerth and Schwentick is to determine the exact complexity and identify tractable fragments.

If we replace the XKFD above with an FD $x_1,\ldots,x_j,@x_{j+1},\ldots,@x_{n-1} \to @x_n$ we have

$$\alpha(x_1,\ldots,x_n) \wedge \alpha(x_1,\ldots,x_j,x'_{j+1},\ldots,x'_n) \;\Rightarrow \\ \Rightarrow\; x_{j+1} \nsim x'_{j+1} \vee \cdots \vee x_{n-1} \nsim x'_{n-1} \vee x_n \sim x'_n\,,$$

which cannot be expressed without mixing $\sim$ and $\nsim$. As we have explained, consistency and entailment is undecidable for such constraints, but one can investigate fragments with restricted schemas and tuple-selectors. This is what Niewerth and Schwentick do.

As XKFDs with tree patterns as tuple-selectors can express XML Schema key and unique constraints [18], XML keys by Arenas, Fan, and Libkin [2], and XFDs by Arenas and Libkin [3], so can non-mixing constraints. A technical subtlety is that some of these classes of constraints apply to nodes of a specified type (playing the role of a state in XML Schemas). As proposed by Niewerth and Schwentick, we can deal with it by annotating tree nodes with types (verified by the automaton encoding the schema), and let the patterns refer to types and labels. This slight extension does not affect our complexity bounds. Also, XML Schema unique constraints demand that each field path selects at most one node, and XML Schema key constraints demand *exactly* one node; the latter can be checked by the automaton too. In practice, one often wants at most (or exactly) one *data value*, not tree node. This may or may not be equivalent. To express that at most one data value is selected, we can use the singleton constraints discussed in Section 2.3. Note that this requires assertions over $\mathsf{sig}_{\sim}$.

4.5 Consistency of XML schema mappings

Schema mappings are a formalism used in data exchange scenarios to specify relations between instances of two database schemas, a *source* schema and

a *target* schema [1,14]. In the basic setting for XML [4], schemas can be abstracted as tree automata, and the relation between source and target instances can be defined by a set Σ of dependencies of the form

$$\alpha(\bar{x}) \Rightarrow \alpha'(\bar{x})$$

where α, α' are conjunctive queries over sig_{nav}, treated as queries selecting data values, not nodes. That is, a pair of data trees (t, t') satisfies dependency σ of the form above, written as $(t, t') \models \sigma$, if

$$\left\{\mathsf{val}_t(\bar{u}) \;\middle|\; t \models \alpha(\bar{u})\right\} \subseteq \left\{\mathsf{val}_{t'}(\bar{u}) \;\middle|\; t' \models \alpha'(\bar{u})\right\}.$$

The *consistency problem for XML schema mappings* [4] is to decide for a given schema mapping $\mathcal{M} = (\mathcal{A}, \mathcal{A}', \Sigma)$, whether there exists a tree t accepted by automaton $\mathcal{A}$ and a tree t' accepted by automaton $\mathcal{A}'$ such that $(t, t') \models \Sigma$. This problem is known to be decidable: without loss of generality one may assume that all data values in t and t' are equal, and use standard automata techniques ignoring data values. This is not only uninspiring theoretically, but also not very practical: an instance with all data values equal is not a convincing witness that the mapping makes sense. What if the source schema includes constraints, say XML Schema key or unique constraints? We cannot assume that all data values are equal any more. As we have argued in the previous subsection, such constraints can be expressed with non-mixing constraints, which leads us to the problem of *consistency with source constraints*, a common generalization of consistency of constraints and schema mappings: given a schema mapping $\mathcal{M} = (\mathcal{A}, \mathcal{A}', \Sigma)$ and a set of non-mixing constraints Σ_{src}, decide if there exist a tree t accepted by automaton $\mathcal{A}$ and a tree t' accepted by automaton $\mathcal{A}'$ such that $t \models \Sigma_{src}$ and $(t, t') \models \Sigma$.

The following lemma gives the connection between XML schema mappings and non-mixing constraints that allows us to apply our decidability result. It was proved in a slightly different but equivalent form in [13]. A non-mixing constraint with *free data value predicates* uses additional unary predicate symbols in the assertions. A data tree t satisfies a set Σ of such constraints (possibly sharing some additional predicate symbols) if it satisfies Σ' obtained from Σ by replacing each additional predicate symbol with some $d \in \mathbb{D}$. Free data value predicates are not problematic for the consistency algorithm, as it can guess the data values to replace them; up to equality type with respect to data values already used in Σ, there are only exponentially many possibilities.

Lemma 7 *For each schema mapping $\mathcal{M} = (\mathcal{A}, \mathcal{A}', \Sigma)$ one can compute in doubly exponential time sets $\Sigma^1_\sim, \Sigma^2_\sim, \ldots, \Sigma^m_\sim$ of non-mixing constraints with free data value predicates, each obtained from Σ by replacing target-side queries $\alpha'(\bar{x})$ with assertions $\eta_\sim(\bar{x})$ of exponential size, such that for each data tree t, $t \models \Sigma^i_\sim$ for some $i \in \{1, \ldots, m\}$ if and only $(t, t') \models \Sigma$ for some data tree t' accepted by automaton $\mathcal{A}'$.*

Thus, mapping $\mathcal{M}$ is consistent with source constraints Σ_{src} if and only if at least one of the sets $\Sigma^i_\sim \cup \Sigma_{src}$ obtained via Lemma 7 is consistent with respect

to automaton $\mathcal{A}$. Since the number of variables in each involved constraint is linear, the latter can be tested in 2ExpTime, as the algorithm from Section 3 is doubly exponential only in the maximal number of variables. As Lemma 7 translates mappings into constraints with assertions over $\mathsf{sig}_{\sim}$, even if Σ_{src} is just a set of key constraints (expressible with assertions over $\mathsf{sig}_{\nsim}$), we need the full power of non-mixing constraints, allowing assertions over $\mathsf{sig}_{\sim}$ and $\mathsf{sig}_{\nsim}$.

4.6 Data cut and clique-width

A classical measure of simplicity for relational structures is that of clique-width [11]. As has been noticed before for unordered data trees, clique-width and data cut are related [7]. We shall now reexamine briefly this relationship for ordered data trees, and in the following subsection we shall see how it can be used to extend our decidability results to constraints with much more expressive selector queries.

Let $\tau = \{R_1, \ldots, R_\ell\}$ be a relational signature, that is, a set of predicate symbols with arities $\mathrm{ar}(R_i)$. A (finite) τ-structure $\mathbb{A}$ is a tuple $\langle A, R_1^{\mathbb{A}}, \ldots, R_\ell^{\mathbb{A}}\rangle$ consisting of a finite universe A and relations $R_i^{\mathbb{A}} \subseteq A^{\mathrm{ar}(R_i)}$ (interpretations of the predicates). A k-coloured τ-structure is a pair $(\mathbb{A}, \gamma)$, consisting of a τ-structure $\mathbb{A}$ and a mapping $\gamma : A \to \{1, \ldots, k\}$, assigning colours to elements of the universe of $\mathbb{A}$.

Clique-width of structures is defined by means of an appropriate notion of decomposition, traditionally known as *k-expression (over τ)*. It is defined as a term over the following set of operations (function symbols) $\mathrm{Op}(\tau, k)$:

- $\mathrm{new}(i)$ for $1 \leq i \leq k$, nullary,
- $\mathrm{col}(i, j)$ for $1 \leq i, j \leq k$, unary,
- $R(i_1, \ldots, i_r)$ for predicates $R \in \tau$ of arity r and $1 \leq i_1, \ldots, i_r \leq k$, unary,
- $\oplus$, binary.

With each k-expression e we associate a k-coloured τ-structure $[\![e]\!]$:

- $[\![\mathrm{new}(i)]\!]$ is a structure with a single element, coloured i, and empty relations;
- $[\![\mathrm{col}(i, j)(e)]\!]$ is obtained from $[\![e]\!]$ by recolouring all elements of colour i to j;
- $[\![R(i_1, \ldots, i_r)(e)]\!]$ is obtained from $[\![e]\!] = (\mathbb{A}, \gamma)$ by adding to $R^{[\![e]\!]}$ all tuples $(a_1, \ldots, a_r)$ such that $a_j \in A$ and $\gamma(a_j) = i_j$ for $1 \leq j \leq r$;
- $[\![e \oplus e']\!]$ is the disjoint union of $[\![e]\!]$ and $[\![e']\!]$.

A *k-expression for* $\mathbb{A}$ is any k-expression e such that $[\![e]\!] = (\mathbb{A}, \gamma)$ for some γ. The *clique-width* of $\mathbb{A}$ is the least k such that there exists a k-expression for $\mathbb{A}$.

Example 2 Consider a data tree t_n consisting of a root w and two branches $u_1, u_2, \ldots, u_n$ and $v_1, v_2, \ldots, v_n$, in which all nodes have label a and the data values correspond to the node's depth in the tree, as shown in Figure 4. Then, t_{u_1} and $t - t_{u_1}$ share n different data values, and the data cut of t_n is n.

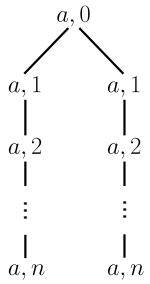

Fig. 4 A tree of data-cut n and clique width bounded by 7.

Let us see t_n as a relational structure over the signature $\mathsf{sig}_{nav} \cup \{\sim\}$, where $\sim$ is interpreted as the equivalence relation with abstraction classes $\{w\}$ and $\{u_i, v_i\}$ for $i = 1, 2, \ldots, n$. We claim that the clique width of t_n is bounded by 7: if we construct t_n top-down, level by level, at any point of the construction it is enough to distinguish between the root, the internal nodes on two branches, the two current leaves, and the two new nodes.

To see this, begin with a node of colour root and set its label with $a(\mathsf{root})$, add two nodes of colours leaf_1 and leaf_2 with relations specified by

$$a(\mathsf{leaf}_i),\ {\downarrow}\,(\mathsf{root}, \mathsf{leaf}_i),\ {\downarrow^+}\,(\mathsf{root}, \mathsf{leaf}_i),\ {\rightarrow}\,(\mathsf{leaf}_1, \mathsf{leaf}_2),\ {\rightarrow^+}\,(\mathsf{leaf}_1, \mathsf{leaf}_2),\ {\sim}\,(\mathsf{leaf}_1, \mathsf{leaf}_2)$$

for $i = 1, 2$ and then repeat the following $n-1$ times: add two nodes of colours new_1 and new_2 with relations specified by

$$a(\mathsf{new}_i),\ {\downarrow}\,(\mathsf{leaf}_i, \mathsf{new}_i),\ {\downarrow^+}\,(\mathsf{root}, \mathsf{new}_i),\ {\downarrow^+}\,(\mathsf{internal}_i, \mathsf{new}_i),\ {\downarrow^+}\,(\mathsf{leaf}_i, \mathsf{new}_i),\ {\sim}\,(\mathsf{new}_1, \mathsf{new}_2)$$

and recolour using $\mathrm{col}(\mathsf{leaf}_i, \mathsf{internal}_i)$, $\mathrm{col}(\mathsf{new}_i, \mathsf{leaf}_i)$ for $i = 1, 2$. □

The example shows that trees of bounded clique width can have arbitrary large data cut. We shall now see that bounded data cut implies bounded clique width.

For each set $D \subseteq \mathbb{D}$, data trees can be seen as relational structures over the signature $\mathsf{sig}_{nav} \cup \mathsf{sig}^D_\sim$, where $\mathsf{sig}^D_\sim = \{\sim\} \cup D$; that is, we restrict the unary predicates in $\mathsf{sig}_\sim$ to those associated to data values from D.

Proposition 2 *For each finite $D \subseteq \mathbb{D}$ and each data tree t seen as a relational structure over $\mathsf{sig}_{nav} \cup \mathsf{sig}^D_\sim$,*

$$\mathit{cliquewidth}(t) \leq 4 \cdot \left(\frac{3}{2} \cdot \mathit{datacut}(t) + 2 + |D| \right) .$$

Proof Let $C = \{1, 2, \ldots, N\}$ with $N = \left\lfloor \frac{3}{2} \cdot \mathit{datacut}(t) \right\rfloor + 1$. By Lemma 4, there exists a tree s over the alphabet $\Gamma \times (C \cup D) \times \mathbb{P}(C)$ such that the encoded data tree $\widehat{s}$ equals t up to a permutation of $\mathbb{D} - D$. That is, $\widehat{s}$ and t are equal

when seen as relational structures over $\mathsf{sig}_{nav} \cup \mathsf{sig}^D_{\sim}$. We shall turn s into a $4(|C| + |D| + 1)$-expression for $\widehat{s}$, interpreting colours as elements of the set

$$\{\mathsf{last\text{-}root}, \mathsf{other\text{-}root}, \mathsf{new\text{-}root}, \mathsf{not\text{-}root}\} \times \big(C \cup D \cup \{\bot\}\big) .$$

Processing the nodes of s in the usual order (bottom-up and left-to-right), for each node w we construct a $4(|C| + |D| + 1)$-expression e_w such that

$$[\![e_w]\!] = (\widehat{s_w}, \gamma)$$

and for each node u the colour $\gamma(u)$ satisfies the following properties:

1. the first coordinate describes the status of the node u in the forest $\widehat{s_w}$: the last root, one of the other roots, or not a root (the value $\mathsf{new\text{-}root}$ will be used later);
2. the second coordinate is the data value stored in the node in $\widehat{s_w}$ if this value belongs to $C \cup D$, or $\bot$ if it does not.

Let w be a node of s, labelled with (a, c, R). To build e_w, we begin by creating a new node and specifying the unary relations for it (label and data value) with operations

$$\mathrm{new}\big((\mathsf{new\text{-}root}, c)\big) , \ a\big((\mathsf{new\text{-}root}, c)\big) , \ c\big((\mathsf{new\text{-}root}, c)\big) ,$$

where the last operation is included only if $c \in D$ (that is, c is in the signature). If w has children, let w'' be its last child. Then, the expression $e_{w''}$ is already constructed and we incorporate it into the expression e_w as follows:

- combine the expression built so far with $e_{w''}$ using the operation $\oplus$;
- specify structural relations between the two parts using the operations

$$\begin{aligned} &\downarrow \big((\mathsf{new\text{-}root}, c), (\mathsf{last\text{-}root}, d)\big) , && \downarrow^+ \big((\mathsf{new\text{-}root}, c), (\mathsf{last\text{-}root}, d)\big) , \\ &\downarrow \big((\mathsf{new\text{-}root}, c), (\mathsf{other\text{-}root}, d)\big) , && \downarrow^+ \big((\mathsf{new\text{-}root}, c), (\mathsf{other\text{-}root}, d)\big) , \\ & && \downarrow^+ \big((\mathsf{new\text{-}root}, c), (\mathsf{not\text{-}root}, d)\big) \end{aligned}$$

  for all $d \in C \cup D \cup \{\bot\}$;
- change $\mathsf{last\text{-}root}$ and $\mathsf{other\text{-}root}$ to $\mathsf{not\text{-}root}$ with the operations

$$\mathrm{col}\big((\mathsf{last\text{-}root}, d), (\mathsf{not\text{-}root}, d)\big) , \qquad \mathrm{col}\big((\mathsf{other\text{-}root}, d), (\mathsf{not\text{-}root}, d)\big)$$

  for all $d \in C \cup D \cup \{\bot\}$;

Similarly, if w' is the previous sibling of w, we incorporate the expression $e_{w'}$ as follows:

- combine the expression build so far with $e_{w'}$ using the operation $\oplus$;
- specify structural relations between the two parts using the operations

$$\begin{aligned} &\rightarrow \big((\mathsf{last\text{-}root}, d), (\mathsf{new\text{-}root}, c)\big) , && \rightarrow^+ \big((\mathsf{last\text{-}root}, d), (\mathsf{new\text{-}root}, c)\big) , \\ & && \rightarrow^+ \big((\mathsf{other\text{-}root}, d), (\mathsf{new\text{-}root}, c)\big) \end{aligned}$$

  for all $d \in C \cup D \cup \{\bot\}$;

– change $\mathsf{last\text{-}root}$ to $\mathsf{other\text{-}root}$ with the operations

$$\mathrm{col}\big((\mathsf{last\text{-}root}, d), (\mathsf{other\text{-}root}, d)\big)$$

for all $d \in C \cup D \cup \{\bot\}$.

Finally, we take care of data equalities and clean up the colours:

– specify data equalities between the combined parts using the operations

$$\sim \big((\xi, d), (\zeta, d)\big)$$

for all $\xi, \zeta \in \{\mathsf{new\text{-}root}, \mathsf{other\text{-}root}, \mathsf{not\text{-}root}\}$ and $d \in C \cup D$;
– change $\mathsf{new\text{-}root}$ to $\mathsf{last\text{-}root}$ with the operation

$$\mathrm{col}\big((\mathsf{new\text{-}root}, c), (\mathsf{last\text{-}root}, c)\big)\,;$$

– refresh the colours with the operations

$$\mathrm{col}\big((\xi, d), (\xi, \bot)\big)$$

for all $\xi \in \{\mathsf{last\text{-}root}, \mathsf{other\text{-}root}, \mathsf{not\text{-}root}\}$ and $d \in R$;

By construction, the resulting expression e_w satisfies properties 1 and 2. □

Thus, bounded data cut is a stronger property than bounded clique width. It can be seen as a strengthening of bounded clique-width for data trees, in which decompositions must closely follow the structure of data trees.

4.7 MSO constraints

Our decidability results for consistency and entailment of non-mixing constraints can be naturally extended by allowing selectors expressed in monadic second-order logic (MSO), a powerful extension of first order-logic in which quantification over subsets of the universe is available. However, as is usually the case when MSO is involved, the complexity will be non-elementary.

The syntax of MSO formulae over sig_{nav} is

$$\varphi, \psi ::= \exists X\, \varphi \,|\, \exists x\, \varphi \,|\, \varphi \wedge \psi \,|\, \neg\varphi \,|\, x \in X \,|\, x \downarrow y \,|\, x \downarrow^+ y \,|\, x \to y \,|\, x \to^+ y \,|\, a(x)$$

for $a \in \Gamma$; the semantics is the natural one, with the usual distinction between first-order variables (lower case) referring to elements of the universe and second-order variables (upper case) referring to subsets of the universe.

We consider *MSO constraints* of the form

$$\varphi(\bar{x}) \Rightarrow \eta_{\sim}(\bar{x}) \wedge \eta_{\not\sim}(\bar{x})\,,$$

where the selector $\varphi(\bar{x})$ is an MSO formula over sig_{nav} in which all free variables are first-order.

As a first step, we reprove the bound on the data cut, shown in Lemma 3. Instead of using Lemma 2 we rely on the compositionality of MSO. For a forest

f over Γ, a tuple $\bar{v} = (v_1, \ldots, v_m)$ of nodes of f, and a tuple $\bar{V} = (V_1, \ldots, V_n)$ of sets of nodes of f, let

$$\langle f; \bar{v}; \bar{V} \rangle$$

be the forest over the alphabet $\Gamma \times \{0,1\}^{m+n}$ obtained from f by extending labels with binary vectors encoding $\bar{v}$ and $\bar{V}$: a node w is labelled with

$$(\mathsf{lab}_f(w), e_1, \ldots e_m, E_1, \ldots, E_n)\,,$$

where $e_i = 1$ if and only if $w = v_i$, and $E_j = 1$ if and only if $w \in V_j$. If $\bar{v}$ or $\bar{V}$ is empty, we skip it and write, for instance, $\langle f; \bar{v} \rangle$. It is well known that for a given MSO formula

$$\varphi(x_1, \ldots, x_m, X_1, \ldots X_n)$$

one can effectively construct a deterministic automaton $\mathcal{A}_\varphi$ (of non-elementary size) recognizing the tree language

$$\left\{ \langle t; \bar{v}; \bar{V} \rangle \,\middle|\, t \models \varphi(\bar{v}, \bar{V}) \right\}\,.$$

The construction follows the syntactic structure of MSO formulas: it begins with explicit automata for atomic formulas, and then turns logical connectives into Boolean operations on automata, and existential quantification into projecting out the corresponding binary coordinate from the alphabet. Defining the *φ-type* of a forest f over $\Gamma \times \{0,1\}^{m+n}$ as the state of the automaton $\mathcal{A}_\varphi$ in the root of the last tree of f (in the unique run over f), we obtain the following analogue of Lemma 2.

Lemma 8 *Let $\varphi(\bar{x}, \bar{y})$ be an MSO formula over sig_{nav}, where $\bar{x}$ and $\bar{y}$ are disjoint tuples of first-order variables, and let w be a node of a data tree t. For all tuples $\bar{u}, \bar{u}'$ of nodes from t_w and tuples $\bar{v}, \bar{v}'$ of nodes from $t - t_w$, if*

$$t \models \varphi(\bar{u}, \bar{v}) \quad \textit{and} \quad t \models \varphi(\bar{u}', \bar{v}')\,,$$

and the φ-types of $\langle t_w; \bar{u} \rangle$ and $\langle t_w; \bar{u}' \rangle$ are equal, then

$$t \models \varphi(\bar{u}, \bar{v}') \quad \textit{and} \quad t \models \varphi(\bar{u}', \bar{v})\,.$$

Proof As $t \models \varphi(\bar{u}, \bar{v})$ and $t \models \varphi(\bar{u}', \bar{v}')$, the trees $\langle t; \bar{u}, \bar{v} \rangle$ and $\langle t; \bar{u}', \bar{v}' \rangle$ are accepted by the deterministic automaton $\mathcal{A}_\varphi$. Moreover, the state in the node w in the unique runs of $\mathcal{A}_\varphi$ over these trees is the same, because $\langle t; \bar{u}, \bar{v} \rangle_w = \langle t_w; \bar{u} \rangle$ and $\langle t; \bar{u}', \bar{v}' \rangle_w = \langle t_w; \bar{u}' \rangle$, and the φ-types of $\langle t_w; \bar{u} \rangle$ and $\langle t_w; \bar{u}' \rangle$ are equal. Hence, swapping $\langle t; \bar{u}, \bar{v} \rangle_w$ and $\langle t; \bar{u}', \bar{v}' \rangle_w$ does not affect acceptance by $\mathcal{A}_\varphi$. That is, the resulting trees $\langle t; \bar{u}', \bar{v} \rangle$ and $\langle t; \bar{u}, \bar{v}' \rangle$ are accepted by $\mathcal{A}_\varphi$. Consequently, $t \models \varphi(\bar{u}', \bar{v})$ and $t \models \varphi(\bar{u}, \bar{v}')$. □

Now, we can show a bound on the data cut for non-mixing MSO constraints. Unlike in Lemma 3, the bound is non-elementary: it is proportional to the maximal size of the automata for the MSO formulas used in the constraints.

Lemma 9 *If a set $\Sigma_\sim \cup \Sigma_{\not\sim}$ of non-mixing MSO constraints is satisfied in a data tree t, it is also satisfied in some data tree t' obtained from t by changing data values, such that*

$$\mathit{datacut}(t') \leq S \cdot \ell \cdot 2^\ell \cdot (\ell + m)^{\ell^2} \cdot |\Sigma_\sim|\,,$$

where ℓ and m are the maximal numbers of, respectively, free variables and predicates from $\mathbb{D} \cup \check{\mathbb{D}}$ in the constraints from $\Sigma_\sim$, and S is the maximal number of types for the selector formulas in the constraints from $\Sigma_\sim$.

The same is true for counter-examples to entailment, except that the bound on the data cut needs to be increased by the number of variables in the violated assertion.

Proof We proceed just like for Lemma 3. Let us take a node w of the data tree t and an MSO constraint $\varphi(\bar{x}, \bar{y}) \Rightarrow \eta_\sim(\bar{x}, \bar{y})$ from $\Sigma_\sim$ with a fixed partition $\bar{x}$, $\bar{y}$ of the free variables of φ. Using Lemma 8, we arrive at the following condition: for all tuples $\bar{u}, \bar{u}'$ of nodes from t_w and all tuples $\bar{v}, \bar{v}'$ of nodes from $t - t_w$, if $t \models \varphi(\bar{u}, \bar{v})$, $t \models \varphi(\bar{u}', \bar{v}')$, and the φ-types of $\langle t_w, \bar{u}\rangle$ and $\langle t_w, \bar{u}'\rangle$ are equal, then $t \models \eta_\sim(\bar{u}, \bar{v}')$. This can be reformulated as follows: for each tuple $\bar{u}$ of nodes from t_w such that the φ-type of $\langle t_w; \bar{u}\rangle$ is q and $t \models \varphi(\bar{u}, \bar{v})$ for some tuple $\bar{v}$ of nodes from $t - t_w$, the tuple $\mathsf{val}_t(\bar{u})$ of data values belongs to the set

$$Z^q_{\varphi(\bar{x},\bar{y}) \Rightarrow \eta_\sim(\bar{x},\bar{y})} = \bigcap_{\bar{v}'} \left\{ \bar{c} \in \mathbb{D}^{|\bar{x}|} \;\middle|\; \eta(\bar{c}, \mathsf{val}_t(\bar{v}')) \right\},$$

where $\bar{v}'$ ranges over tuples of nodes from $t - t_w$ satisfying $t \models \varphi(\bar{u}', \bar{v}')$ for some tuple $\bar{u}'$ of nodes from t_w such that the φ-type of $\langle t_w; \bar{u}'\rangle$ is q.

Like before, we modify the tree t by changing to a fresh one each data value used in t_w, except for those from the set $D \subseteq \mathbb{D}$ of data values used in the definitions of the sets $Z^q_{\varphi(\bar{x},\bar{y}) \Rightarrow \eta_\sim(\bar{x},\bar{y})}$, with $\varphi(\bar{x}, \bar{y}) \Rightarrow \eta_\sim(\bar{x}, \bar{y})$ ranging over constraints from $\Sigma_\sim$ with all possible partitions of free variables, and q ranging over all possible φ-types. As the bound on the number of data values used in the canonical definition of a single set $Z^q_{\varphi(\bar{x},\bar{y}) \Rightarrow \eta_\sim(\bar{x},\bar{y})}$ remains unchanged, we have $|D| \leq |\Sigma_\sim| \cdot 2^\ell \cdot S \cdot \left(\ell \cdot (m + \ell)^{\ell^2}\right)$. Performing this modification for each node w, we guarantee the bound on data cut as stated in the lemma.

For the second claim, extend the set D with the data values used in the tuple of nodes violating the assertion, as described in Section 4.1 for constraints with CQ selectors. □

As each set of MSO constraints can be rewritten as a single MSO formula over the signature $\mathsf{sig}_{nav} \cup \mathsf{sig}_\sim$, by Lemma 9 and Proposition 2 from Section 4.6, the consistency problem and the entailment problem reduce to satisfiability of MSO over structures of bounded clique-width (one has to ensure that the structure is indeed a data tree, but this can be easily expressed in MSO). As the latter is known to be decidable [10], we immediately obtain decidability of consistency and entailment. For completeness, we give a direct proof, avoiding the notion of clique-width.

Theorem 4 *Consistency and entailment of non-mixing MSO constraints is decidable.*

Proof Let $\Sigma_\sim \cup \Sigma_{\not\sim}$ be a set of non-mixing constraints and let $\mathcal{A}$ be a tree automaton. By Lemma 9, it is enough to test satisfiability of $\Sigma_\sim \cup \Sigma_{\not\sim}$ over trees of data cut bounded by a number N, computable from $\Sigma_\sim \cup \Sigma_{\not\sim}$. Let $D \subseteq \mathbb{D}$ be the set of data values used explicitly in $\Sigma_\sim \cup \Sigma_{\not\sim}$, and let $C \subseteq \mathbb{D} - D$ be a fixed set such that $|C| = \lfloor \frac{3}{2} \cdot N \rfloor + 1$. Like before, by Lemma 4, the proof boils down to constructing an automaton recognizing the set of trees t over $\Gamma \times (C \cup D) \times \mathbb{P}(C)$ such that the data tree $\widehat{t}$ satisfies $\Sigma_\sim \cup \Sigma_{\not\sim}$. Each MSO constraint $\varphi(\bar{x}) \Rightarrow \eta(\bar{x})$ is equivalent to a closed MSO formula $\forall \bar{x} \big(\varphi(\bar{x}) \rightarrow \eta(\bar{x})\big)$ over the signature $\mathsf{sig}_{nav} \cup \mathsf{sig}_\sim$ (in the presence of negation, $\mathsf{sig}_{\not\sim}$ is redundant). Hence, a finite set of MSO constraints is equivalent to a conjunction of such formulas. Thus, it suffices to construct an automaton accepting trees t over $\Gamma \times (C \cup D) \times \mathbb{P}(C)$ such that $\widehat{t}$ satisfies φ, where φ is an MSO formula over $\mathsf{sig}_{nav} \cup \mathsf{sig}_\sim$, using only predicates associated with data values from the set D.

We modify the standard construction of the automaton $\mathcal{A}_\varphi$ for a formula φ of MSO over sig_{nav}. As the structure of the tree and the labelling with elements of Γ is the same in t and $\widehat{t}$, we only need to provide explicit constructions for the atomic formulas over $\mathsf{sig}_\sim$.

For formulas of the form $d(x)$, we have $d \in D$, so the data value d is represented explicitly in t. Hence, the automaton simply identifies the node with value 1 in the corresponding binary coordinate of the label, and accepts if and only if the non-binary component of this label is (a, d, R) for some a and R.

For formulas of the form $x \sim x'$, the automaton also identifies the nodes x and x' in the input tree t, and then accepts if they are labelled with (a, d, R) and (a', d, R') for some a, a', R, R', and additionally, if $d \in C$, then it is not refreshed before reaching the first node w such that t_w contains both x and x'.

For entailment, we use the additional claim of Lemma 9 and include in D also the data values used explicitly in the second set of constraints, $\Sigma'_\sim \cup \Sigma'_{\not\sim}$. As the fact that $\Sigma_\sim \cup \Sigma_{\not\sim}$ holds and $\Sigma'_\sim \cup \Sigma'_{\not\sim}$ does not hold can also be expressed with a single closed MSO formula, we can use directly the construction described above. □

Both approaches give non-elementary complexity, as already the bound of Lemma 9 is non-elementary. This cannot be improved, as the satisfiability problem for MSO over sig_{nav}, well known to be non-elementary, easily reduces to inconsistency of MSO constraints: a closed formula φ is satisfiable if and only if

$$\big\{\varphi \wedge a(x) \;\Rightarrow\; 0(x) \wedge 1(x) \;\big|\; a \in \Gamma\big\}$$

is inconsistent with respect to the trivial automaton accepting all trees.

5 Conclusions

We have shown that consistency and entailment of non-mixing constraints are decidable. Both problems are 2ExpTime-complete, but become ExpTime-complete when we restrict selector queries to tree patterns and bound the number of variables in assertions; decidability can be pushed further to constraints with selector queries defined in monadic second order logic over the signature sig_{nav}, but the complexity becomes non-elementary. We have reinterpreted these results in terms of validity and containment of conjunctive queries, as well as consistency of schema mappings. The latter setting best illustrates the benefits of combining assertions over $\mathsf{sig}_{\sim}$ and $\mathsf{sig}_{\not\sim}$. Indeed, equalities are involved even in the simplest schema mappings, and inequalities allow to cover key constraints over the source database.

We worked with ordered trees, but all discussed results immediately carry over to unordered trees: as long as the signature does not contain the horizontal axes, one can freely move back and forth between ordered and unordered trees by forgetting the sibling order or introducing it arbitrarily. As both 2ExpTime lower bounds, the one from Lemma 5 and the one from [6], do not use the horizontal axes, they also hold for unordered trees. The same is true of the undecidability for the settings that mix equality and inequality [6]. Similarly, restricting to ranked trees does not change the picture: the upper bounds carry over immediately, and the lower bounds only use trees of bounded branching. The reductions can be also adapted to the case of unlabelled trees: one can simulate labels with unique small tree gadgets attached to the main nodes of the tree and use the automaton to ensure that each main node has exactly one gadget attached. However, referring to the gadgets with selector queries requires either the next sibling or the following sibling relation. For unordered unlabelled trees the complexity might drop.

One might also ask how the presence of the schema affects the complexity. The fact that we model schemas as tree automata is inessential: all lower bounds can be adjusted to the setting where the schema language is restricted to DTDs [6]. When there is no schema at all, the consistency problem trivializes, because if a tree satisfies a set of constraints, so does any tree obtained by removing nodes. Hence, it suffices to look for witnesses among trees with a single node, which leads to a polynomial-time algorithm. The question is more interesting for the entailment problem, because there the counter-example must contain enough nodes to falsify the non-entailed constraint. It is plausible that the complexity is lower than with a schema.

Acknowledgements We thank the anonymous referees of ICDT 2016 and TOCS for their insightful questions.

References

1. Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. *Foundations of Data Exchange*. Cambridge University Press, 2014.

2. Marcelo Arenas, Wenfei Fan, and Leonid Libkin. On the complexity of verifying consistency of XML specifications. *SIAM J. Comput.*, 38(3):841–880, 2008.
3. Marcelo Arenas and Leonid Libkin. A normal form for XML documents. *ACM Trans. Database Syst.*, 29:195–232, 2004.
4. Marcelo Arenas and Leonid Libkin. XML data exchange: Consistency and query answering. *J. ACM*, 55(2), 2008.
5. Michael Benedikt, Wenfei Fan, and Floris Geerts. XPath satisfiability in the presence of DTDs. *J. ACM*, 55(2), 2008.
6. Henrik Björklund, Wim Martens, and Thomas Schwentick. Conjunctive query containment over trees using schema information. *Acta Informatica*, pages 1–40, 2016.
7. Mikołaj Bojańczyk, Filip Murlak, and Adam Witkowski. Containment of monadic datalog programs via bounded clique-width. In *Proc. ICALP 2015*, pages 427–439, 2015.
8. Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3):13:1–13:48, 2009.
9. Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. STOC 1977*, pages 77–90, 1977.
10. Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.
11. Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.
12. Claire David, Amélie Gheerbrant, Leonid Libkin, and Wim Martens. Containment of pattern-based queries over data trees. In *Proc. ICDT 2013*, pages 201–212, 2013.
13. Claire David, Piotr Hofman, Filip Murlak, and Michał Pilipczuk. Synthesizing transformations from XML schema mappings. In *Proc. ICDT 2014*, pages 61–71, 2014.
14. Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
15. Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.
16. Ronald Fagin and Moshe Y. Vardi. The theory of data dependencies - a survey. In *Mathematics of Information Processing*, volume 34 of *Proceedings of Symposia in Applied Mathematics*, pages 19–71, Providence, Rhode Island, 1986. American Mathematical Society.
17. Diego Figueira. Alternating register automata on finite words and trees. *Logical Methods in Computer Science*, 8(1), 2012.
18. S. Gao, C. M. Sperberg-McQueen, H.S. Thompson, N. Mendelsohn, D. Beech, and M. Maloney. W3C XML Schema Definition Language (XSD) 1.1, Part 1: Structures. Technical report, World Wide Web Consortium, April 2009.
19. Tomasz Gogacz and Jerzy Marcinkowski. All-instances termination of chase is undecidable. In *Proc. ICALP 2014*, pages 293–304, 2014.
20. Tomasz Gogacz and Jerzy Marcinkowski. Red spider meets a rainworm: Conjunctive query finite determinacy is undecidable. In *Proc. PODS 2016*, pages 121–134, 2016.
21. Sven Hartmann and Sebastian Link. More functional dependencies for XML. In *Proc. ADBIS 2003*, pages 355–369, 2003.
22. Sven Hartmann, Sebastian Link, and Thu Trinh. Solving the implication problem for XML functional dependencies with properties. In *Proc. WoLLIC 2010*, pages 161–175, 2010.
23. Marcin Jurdzinski and Ranko Lazic. Alternating automata on data trees and XPath satisfiability. *ACM Trans. Comput. Log.*, 12(3):19:1–19:21, 2011.
24. Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proc. PODS 2002*, pages 233–246, 2002.
25. Gerome Miklau and Dan Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, 2004.
26. Frank Neven and Thomas Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Log. Meth. Comput. Sci.*, 2(3), 2006.
27. Matthias Niewerth and Thomas Schwentick. Reasoning about XML constraints based on XML-to-relational mappings. In *Proc. ICDT 2014*, pages 72–83, 2014.
28. Moshe Y. Vardi. Fundamentals of dependency theory. In E. Borger, editor, *Trends in Theoretical Computer Science*, pages 171–224. Computer Science Press, 1987.