Three easy pieces on schema mappings for tree-structured data*

Claire David[1] and Filip Murlak[2]

[1] Université Paris-Est Marne-la-Vallée
[2] University of Warsaw

Abstract. Schema mappings specify how data organized under a source schema should be reorganized under a target schema. For tree-structured data, the interplay between these specifications and the complex structural conditions imposed by the schemas, makes schema mappings a very rich formalism. Classical data management tasks, well-understood in the relational model, once again become challenging theoretical problems. Recent results on non-mixing integrity constraints help to push the decidability frontier further for three such problems: consistency of mappings, membership in the composition of mappings, and query answering.

1 Introduction

Schema mappings specify how data organized under a source schema should be reorganized under a target schema. In the relational setting they are expressed as sets of *tuple generating dependencies (tgds)* of the form $\varphi(\bar{x}) \Rightarrow \psi(\bar{x}, \bar{y})$, where $\varphi(\bar{x})$ and $\psi(\bar{x}, \bar{y})$ are conjunctions of atoms over the source schema $\mathcal{S}$ and the target schema $\mathcal{T}$, respectively. A schema mapping $\mathcal{M}$ defines a relation $[\![\mathcal{M}]\!]$ between instances of $\mathcal{S}$ and instances of $\mathcal{T}$: it contains pairs (S, T) such that for each dependency of the form above, for each tuple $\bar{u}$, if $S \models \varphi(\bar{u})$ then $T \models \psi(\bar{u}, \bar{v})$ for some $\bar{v}$ (we assume a common data domain). Such mappings were intensively studied in the context of data exchange, data integration, and metadata management [8, 9], and finally made their way to commercial tools.

In the peak of the popularity of the XML data format, schema mappings were ported to the setting of tree-structured data, in the hope of developing an equally manageable formalism to deal with data exchange and integration [2].

A convenient model for tree-structured data are data trees: ordered unranked trees, whose nodes carry a label from a finite alphabet A and a data value from an infinite data domain $\mathbb{D}$. For simplicity we shall assume that schemas are DTDs. A DTD describes trees by specifying the labelling of children based on the label of their parent. It consists of rules of the form $a \to \omega$, where $a \in A$ and ω is a regular expression over A. Such rule means that for each node with label a the sequence of labels of its children, read from left to right, gives a word generated by ω. Labels with no rule may be used in leaves only. The root has a fixed label $r \in A$. A (data) tree *conforms* to the DTD if it satisfies all these rules.

* Supported by Poland's National Science Center grant 2013/11/D/ST6/03075.

For instance, trees conforming to the DTD with rules $r \to ss + \varepsilon$, $s \to rr + \varepsilon$, are binary trees in which odd-level nodes have label r and even-level nodes have label s. Notice that DTDs do not talk about data values at all.

Schema mappings are defined using queries selecting tuples of data values from data trees. We focus on queries expressed with *tree patterns*. Let Var be a countably infinite set of variables. The syntax of tree patterns is defined recursively: for $\ell \in A \cup \{\_\}$ and $x \in \mathsf{Var}$, expressions ℓ and $\ell(x)$ are tree patterns, and if $\varphi_1, \ldots, \varphi_k$ are tree patterns, then

$$//\varphi_1\,, \quad \ell(x)[\varphi_1, \ldots, \varphi_k]\,, \quad \text{and} \quad \ell[\varphi_1, \ldots, \varphi_k]$$

are tree patterns as well. We write $\varphi(\bar{x})$ to indicate that φ uses only variables from $\bar{x}$. For $\bar{u} \in \mathbb{D}$, the pattern $\varphi(\bar{u})$ is obtained from $\varphi(\bar{x})$ by substituting variables $\bar{x}$ with elements of the tuple $\bar{u}$. For $a \in A$, the pattern a can be *matched* at each tree node with label a; for $\_$, the label can be arbitrary. In case of $a(d)$ and $\_(d)$, the node additionally must store the data value d. Pattern $//\varphi$ can be matched at each node that has a descendent at which φ can be matched. Finally, $\rho[\varphi_1, \ldots, \varphi_k]$ with $\rho \in \{a, \_, a(d), \_(d)\}$ can be matched at each node at which ρ can be matched, and which has children (not necessarily distinct) at which $\varphi_1, \ldots, \varphi_k$ can be matched. We write $T \models \varphi(\bar{u})$ if $\varphi(\bar{u})$ can be matched at some node of the data tree T. Mappings are defined with dependencies of the form

$$\varphi(\bar{x}) \wedge \alpha(\bar{x}) \Rightarrow \psi(\bar{x}, \bar{y}) \wedge \beta(\bar{x}, \bar{y})\,,$$

where $\varphi(\bar{x}), \psi(\bar{x}, \bar{y})$ are tree patterns and $\alpha(\bar{x}), \beta(\bar{x}, \bar{y})$ are conjunctions of equalities and inequalities among variables. We make the usual safety assumption that $\varphi(\bar{x})$ uses each variable in $\bar{x}$ and $\psi(\bar{x}, \bar{y})$ uses each variable in $\bar{y}$. To simplify descriptions of fragments with restricted use of equalities and inequalities, we make a proviso that $\varphi(\bar{x})$ uses each variable in $\bar{x}$ only once, and $\psi(\bar{x}, \bar{y})$ uses each variable in $\bar{y}$ only once; variables from $\bar{x}$ can be repeated in $\psi(\bar{x}, \bar{y})$.

As mappings define relations, they can be composed in the semantic sense: $[\![\mathcal{M}]\!] \cdot [\![\mathcal{M}']\!]$ is just the usual (left) composition of relations $[\![\mathcal{M}]\!]$ and $[\![\mathcal{M}']\!]$. It has been shown that to represent the resulting relation as a single mapping, the setting must be extended with function symbols (and some restrictions on $\mathcal{M}$ and $\mathcal{M}'$ must be made) [1, 9]. This is done by allowing in $\alpha(\bar{x})$, $\psi(\bar{x}, \bar{y})$, and $\beta(\bar{x}, \bar{y})$ not just variables, but also arbitrary terms over variables $\bar{x}$, and a countably infinite set Fun of functions symbols. These symbols are implicitly quantified existentially; that is, a pair of trees satisfies the set of dependencies constituting a mapping if there is an interpretation of function symbols as functions $\mathbb{D}^n \to \mathbb{D}$ for appropriate values of n, that makes each dependency satisfied when terms are evaluated according to this interpretation. For instance, the dependency

$$a[b(x), c(y)] \Rightarrow a[b(x), c(y), b'(f(x))]$$

invents a value to be stored in the b'-labelled node together with the copied pair (x, y), but the invented value is the same for all pairs sharing the value of x.

Despite the apparent simplicity of DTDs and tree patterns, the interplay between them makes schema mappings for tree-structured data a very rich formalism. Classical data management tasks, well-understood in the relational model, once again become challenging computational problems, for which even decidability is hard to obtain [1, 5]. Here we show how recent results on non-mixing integrity constraints [6] help to push the decidability frontier further for three such problems: consistency of mappings (Section 2), membership in the composition of mappings (Section 3), and query answering (Section 4). Each of these advances raises further open questions, formulated at the end of the sections.

All the upper bounds we obtain also hold if we allow tree automata as schemas, and replace tree patterns with conjunctive queries (with node variables and data value variables) over the signature consisting of label tests, vertical and horizontal axes, and a predicate relating nodes with stored data values.

2 Consistency

A classic static analysis problem for schema mappings is the following consistency problem first considered by Arenas and Libkin [2]:

Problem:	CONS
Input:	mapping $\mathcal{M}$
Question:	Are there trees S, T such that $(S, T) \in [\![\mathcal{M}]\!]$?

It is essentially the satisfiability problem for a formalism defining relations over structures, rather than classes of structures.

This problem is undecidable as soon as either $=$ or $\neq$ is allowed on both sides of the mappings [1], but there are no reasons—other than mathematical elegance—to consider only dependencies with symmetric use of data comparisons. In fact, $=$ on the source side of the mappings is often not essential. For instance, while syntactic closure under composition requires $=$ on the target, it does not require $=$ on the source [1]. Forbidding $=$ on the source side of course means that we cannot have proper joins when querying the source instance, but it might be a price worth paying for decidable consistency. Similarly, $\neq$ on the source may not be always necessary: for instance, key constraints on the source can be expressed using $\neq$ exclusively on the target side [6]. Here we take a closer look at dependencies with asymmetric use of data comparisons. In this new context, we tighten the results of [1] as follows. We use superscripts $=$ and $\neq$ to indicate that a formula uses, respectively, only equalities, and only inequalities.

Proposition 1. CONS *is undecidable already for mappings using dependencies of exactly one of the following forms:*

$$\varphi(\bar{x}) \wedge \alpha^{=}(\bar{x}) \Rightarrow \psi(\bar{x}), \qquad \varphi(\bar{x}) \Rightarrow \psi(\bar{x}, \bar{y}) \wedge \alpha^{\neq}(\bar{x}, \bar{y}),$$

and EXPTIME*-complete for mappings with function symbols and dependencies of the form:*

$$\varphi(\bar{x}) \wedge \alpha^{\neq}(\bar{x}) \Rightarrow \psi(\bar{x}, \bar{y}) \wedge \alpha^{=}(\bar{x}, \bar{y}).$$

Proof. The proof relies on the fact that consistency is undecidable already when dependencies are of the form $\varphi(\bar{x}) \wedge \alpha^{=}(\bar{x}) \Rightarrow \beta^{=}(\bar{x})$, as proved in [1].

We first reduce consistency of such mappings to consistency of mappings with dependencies of the first form given in the statement of the proposition. Let $\mathcal{M}$ be a mapping with dependencies of the form

$$\varphi(x_1, x_2 \ldots, x_n) \wedge \alpha^{=}(x_1, x_2 \ldots, x_n) \Rightarrow \beta^{=}(x_1, x_2, \ldots, x_n)\,.$$

To obtain the new mapping $\mathcal{M}'$, in each dependency we replace $\beta^{=}(x_1, x_2, \ldots, x_n)$ with the pattern

$$b\big[\_(x_1)[c_1], \_(x_2)[c_2], \ldots, \_(x_n)[c_n]\big]\,,$$

where b and $c_1, c_2, \ldots, c_n$ are fresh labels specific to this dependency. Let us partition the set of variables $x_1, x_2, \ldots, x_n$ into nonempty subsets $Z_1, Z_2 \ldots, Z_m$ for some $m \leq n$, in such a way that two variables are in the same subset if and only if there is a sequence of equalities in $\beta^{=}(x_1, x_2, \ldots, x_n)$ that connects them. Let $b_1, b_2, \ldots, b_m$ be fresh labels and let T_b be a tree whose root has label b and each Z_i is represented by a b_i-labelled child of the root that has a child with label c_j if and only if $x_j \in Z_i$. The target schema of $\mathcal{M}'$ allows all trees obtained by combining under one root with label r arbitrary numbers of copies of trees T_b corresponding to all dependencies of $\mathcal{M}$; note that it uses only finitely many labels. The source schema of $\mathcal{M}'$ is inherited from $\mathcal{M}$. By construction, for each source tree S there exists T such that $(S, T) \in [\![\mathcal{M}]\!]$ if and only if there exists T' such that $(S, T') \models [\![\mathcal{M}']\!]$. This completes the proof of the first claim.

For dependencies of the second form we also begin with the mapping $\mathcal{M}$ above. Each of its dependencies is of the form

$$\varphi(\bar{x}) \wedge x_{i_1} = x_{j_1} \wedge \cdots \wedge x_{i_\ell} = x_{j_\ell} \Rightarrow \beta^{=}(\bar{x})\,.$$

This is equivalent to

$$\varphi(\bar{x}) \Rightarrow x_{i_1} \neq x_{j_1} \vee \cdots \vee x_{i_\ell} \neq x_{j_\ell} \vee \beta^{=}(\bar{x})\,,$$

which can be further rewritten as

$$\varphi(\bar{x}) \Rightarrow y_1 \neq x_{j_1} \wedge \cdots \wedge y_\ell \neq x_{j_\ell} \wedge \big(y_1 = x_{i_1} \vee \cdots \vee y_\ell = x_{i_\ell} \vee \beta^{=}(\bar{x})\big)\,,$$

using fresh variables $\bar{y}$. Thus, we can assume that $\mathcal{M}$ has dependencies of the form

$$\varphi(\bar{x}) \Rightarrow \alpha^{\neq}(\bar{x}, \bar{y}) \wedge \big(\alpha_0^{=}(\bar{x}, \bar{y}) \vee \alpha_1^{=}(\bar{x}, \bar{y}) \vee \cdots \vee \alpha_\ell^{=}(\bar{x}, \bar{y})\big)\,,$$

where $\alpha_i^{=}$ are conjunctions of equalities and $\alpha^{\neq}$ is a conjunction of inequalities. In $\mathcal{M}''$ such dependency is replaced with

$$\varphi(\bar{x}) \Rightarrow b\big[\_(z_1)[c_1], \_(z_2)[c_2], \ldots, \_(z_m)[c_m]\big] \wedge \alpha^{\neq}(\bar{x}, \bar{y})\,,$$

where b is a fresh letter specific to this dependency only, and $z_1, z_2, \ldots, z_m$ is an enumeration of $\bar{x}$ and $\bar{y}$. The target schema of $\mathcal{M}''$ is defined like before, except that subtrees with label b in the root have $\ell + 1$ variants, corresponding to

$\alpha^{=}_0, \alpha^{=}_1, \ldots, \alpha^{=}_\ell$, and multiple copies of multiple variants are allowed. The source schema is inherited from $\mathcal{M}$. It is easy to check that the reduction is correct.

For the decidability claim, let $\mathcal{M}$ be a mapping using neither $=$ over the source, not $\neq$ over the target. Take $(S,T) \in [\![\mathcal{M}]\!]$. Replace all data values with the same fixed data value $\bot$. The obtained pair of trees (S',T') is also in $[\![\mathcal{M}]\!]$, and in the witnessing valuation we can assume that all functions are constantly equal to $\bot$. Hence, if $\mathcal{M}$ is consistent, it can be witnessed with the set of data values restricted to a single value $\bot$. Consequently, we can drop all dependencies containing an inequality on the source (their left hand side will never be satisfied if $\bot$ is the only available data value), and all references to data values in the remaining dependencies (it is always the same value $\bot$, so they make no difference). This gives a mapping that does not use variables at all; for such mappings Cons amounts to exponentially many nonemptiness tests for tree automata of exponential size [1, 2]. The lower bound holds already for such mappings [2]. □

Surprisingly, Cons becomes decidable for mappings using only dependencies of the second form in Proposition 1, $\varphi(\bar{x}) \Rightarrow \psi(\bar{x},\bar{y}) \wedge \alpha^{\neq}(\bar{x},\bar{y})$, as soon as we forbid introducing new variables on the target side (as in full tgds from the relational setting) [6]. Here we show that the argument in fact works for a much more general class of mappings.

The approach relies on a strong decidability result proved in [6]. A *non-mixing constraint* is a dependency of the form

$$\varphi(\bar{x}) \Rightarrow \eta^{=}(\bar{x}) \wedge \eta^{\neq}(\bar{x})\,,$$

where $\varphi(\bar{x})$ is a pattern possibly using constants from $\mathbb{D}$, $\eta^{=}(\bar{x})$ is a positive boolean combination of equalities over $\bar{x}$ and constants from $\mathbb{D}$, and $\eta^{\neq}(\bar{x})$ is a positive boolean combination of inequalities over $\bar{x}$ and constants from $\mathbb{D}$, as well as equalities between a variable from $\bar{x}$ and a constant from $\mathbb{D}$. A tree T satisfies such a constraint if for each tuple $\bar{u}$, $T \models \varphi(\bar{u})$ implies that $\eta^{=}(\bar{u}) \wedge \eta^{\neq}(\bar{u})$ holds.

Theorem 1 ([6]). *It is decidable if there exists a tree conforming to a given schema, and satisfying a given set of non-mixing constraints as well as a given set of patterns without variables. The algorithm runs in time* $2^{n^{\mathrm{poly}(\ell)}}$*, where* n *is the total size of the input, and* ℓ *is the maximal size of involved patterns. The problem is in fact* 2ExpTime*-complete.*

We note that the original setting of [6] models schemas as tree automata and allows full conjunctive queries, but disallows equalities in $\eta^{\neq}(\bar{x})$. Because of that, both the upper bound and the lower bound require some routine adjustments.

The second ingredient is a useful normal form for sets of dependencies, implicit in [7], resulting from the decomposition of schemas into so-called kinds [5].

Lemma 1. *For each mapping* $\mathcal{M}$ *one can compute in doubly exponential time sets* $\Sigma_1, \Sigma_2, \ldots, \Sigma_m$ *of dependencies, each obtained from the set of dependencies of* $\mathcal{M}$ *by replacing all target-side patterns* $\psi(\bar{x},\bar{y})$ *with exponential-size positive boolean combinations* $\eta^{=}(\bar{x},\bar{y})$ *of equalities over* $\bar{x},\bar{y}$ *and constants from* $\mathbb{D}$*, such that for each data tree* S*,* $(S,T) \in [\![\mathcal{M}]\!]$ *for some data tree* T *if and only if* $S \models \Sigma_i$ *for some* $i \in \{1,\ldots,m\}$*.*

Proposition 2. CONS *is decidable (*2EXPTIME*-complete) if the input mapping has only dependencies of the following two forms:*

$$\varphi(\bar{x}) \wedge \alpha^{\neq}(\bar{x}) \Rightarrow \psi(\bar{x}, \bar{y}) \wedge \alpha^{=}(\bar{x}, \bar{y}) \,, \qquad \varphi(\bar{x}) \Rightarrow \alpha^{\neq}(\bar{x}) \,.$$

Proof. By the assumption on the form of dependencies in the input mapping, using Lemma 1 we obtain sets Σ_i of dependencies of the forms

$$\varphi(\bar{x}) \wedge \alpha^{\neq}(\bar{x}) \Rightarrow \eta^{=}(\bar{x}, \bar{y}) \wedge \alpha^{=}(\bar{x}, \bar{y}) \,, \qquad \varphi(\bar{x}) \Rightarrow \alpha^{\neq}(\bar{x}) \,.$$

Dependencies of the second form are already non-mixing constraints. Dependencies of the first form can be rewritten as

$$\varphi(\bar{x}) \Rightarrow \neg\alpha^{\neq}(\bar{x}) \vee \big(\eta^{=}(\bar{x}, \bar{y}) \wedge \alpha^{=}(\bar{x}, \bar{y})\big),$$

and because $\neg\alpha^{\neq}(\bar{x})$ is equivalent to a positive Boolean combination of equalities, we can assume they are of the form

$$\varphi(\bar{x}) \Rightarrow \eta^{=}(\bar{x}, \bar{y}) \,.$$

Finally, variables $\bar{y}$ can be eliminated from $\eta^{=}(\bar{x}, \bar{y})$ by rewriting the positive boolean combination in the disjunctive normal form, completing each disjunct with equalities following by transitivity, and then dropping all equalities involving $\bar{y}$. Because $\eta^{=}(\bar{x}, \bar{y})$ has exponential size, it uses only exponentially many constants. Consequently, there can be only exponentially many different disjuncts that do not contain an equality between two different constants; disjuncts containing such equalities can be dropped. After these modifications each Σ_i is a set of non-mixing constraints and by Theorem 1 it can be tested if either of them is satisfiable in a tree conforming to the source schema. The total size of the obtained non-mixing constraints is exponential, because of Lemma 1, but the size of patterns has not increased. Hence, the problem is in 2EXPTIME. The lower bound follows from the proof of Theorem 1. □

Note that dependencies of the form $\varphi(\bar{x}) \Rightarrow \psi(\bar{x}, \bar{y}) \wedge \alpha^{=}(\bar{x}, \bar{y}) \wedge \alpha^{\neq}(\bar{x})$ can be replaced with $\varphi(\bar{x}) \Rightarrow \psi(\bar{x}, \bar{y}) \wedge \alpha^{=}(\bar{x}, \bar{y})$ and $\varphi(\bar{x}) \Rightarrow \alpha^{\neq}(\bar{x})$; hence, they are also covered by Proposition 2.

Problem 1. It is an interesting question if Proposition 2 can be extended to mappings with function symbols. This would correspond to allowing appropriately limited use of function symbols in non-mixing constraints. As function symbols can simulate fresh variables on the target, this will require quite some care.

3 Composition membership

Composition membership is the following decision problem:

Problem: COMP
Input: mappings $\mathcal{M}$, $\mathcal{M}'$, trees S, T
Question: $(S, T) \in [\![\mathcal{M}]\!] \cdot [\![\mathcal{M}']\!]$?

While it is not a static analysis problem—data are clearly there—it does resemble one, because it asks about the existence of an instance I such that $(S, I) \in [\![\mathcal{M}]\!]$ and $(I, T) \in [\![\mathcal{M}']\!]$. Consequently, despite the simplicity of the logical formalism involved, the problem sits astride the decidability frontier: it is undecidable if either $=$ or $\neq$ is allowed, but becomes decidable if both are forbidden [1]. The argument for the latter makes an additional assumption that function symbols are forbidden. Most often, this is just a simplifying assumption. Surprisingly, in this case it is necessary, as the following result shows.

Proposition 3. COMP *is undecidable for mappings with function symbols. In fact, even existence of a source tree for a given target tree with respect to a fixed mapping is undecidable.*

Proof. To show that the latter problem is undecidable, we reduce the following tiling problem: given a set of tiles $\mathcal{X}$, an initial tile $I \in \mathcal{X}$, a final tile $F \in \mathcal{X}$, and relations $H, V \subseteq \mathcal{X} \times \mathcal{X}$, decide if for some N there exists an $N \times N$ tiling; that is, a function $f\colon \{1, 2, \ldots, N\}^2 \to \mathcal{X}$ such that $f(1,1) = I$, $f(N,N) = F$, $(f(i,j), f(i,j+1)) \in V$, and $(f(j,i), f(j+1,i)) \in H$ for all $i \leq N$ and $j < N$.

The mapping does not depend on the instance of tiling: the source schema is $r \to a$, $a \to b$, $b \to b{+}c$, the target schema is $r \to IFV^*H^*$, $V \to V_1V_2$, $H \to H_1H_2$, and the dependencies are

$$\begin{aligned} r\big[a(x), //c(y)\big] &\longrightarrow r\big[I(f(x,x)), F(f(y,y))\big]\,, \\ r\big[//_{\text{-}}(x), //_{\text{-}}(y)[_{\text{-}}(y')]\big] &\longrightarrow V\big[V_1(f(x,y)), V_2(f(x,y'))\big]\,, \\ r\big[//_{\text{-}}(x)[_{\text{-}}(x')], //_{\text{-}}(y)\big] &\longrightarrow H\big[H_1(f(x,y)), H_2(f(x',y))\big]\,. \end{aligned}$$

The target instance T is the standard tree encoding of a relational instance with relations I, F, H, V storing an instance of the tiling problem. Notice that the set $\mathcal{X}$ is not represented explicitly, but it is assumed to be the set of values occurring in the relations H, V.

The source instance is intended to represent a finite linear order with the least element in the a node, and the largest element in the c node.

Let us see that the reduction is correct. If there exists an $N \times N$ tiling, there is a tree satisfying the dependencies: the data values in this tree are just successive numbers from 1 to N, and the function symbol f is interpreted as the tiling function. Conversely, if there is a source tree satisfying the dependencies, there is also one where all data values in the leaves are different:

- no new rule will be fired, because the dependencies do not have inequality on the source side;
- no rule fired previously will be violated, because no data value is used directly on the target side, and the function f on new values can be defined just like on the values they replace.

Let us call these data values $d_1, d_2, \ldots, d_N$, according to their order of appearance in the single branch of the source tree. The interpretation of the function symbol f, witnessing that the dependencies are satisfied, gives an $N \times N$-tiling.

It follows that Comp is undecidable as well, because a source tree exists for T with respect to $\mathcal{M}$ if and only if $(T_0, T) \in [\![\mathcal{M}_0]\!] \cdot [\![\mathcal{M}]\!]$, where T_0 is some fixed tree and $\mathcal{M}_0$ is the mapping from the trivial schema allowing only T_0 to the source schema of $\mathcal{M}$, whose set of dependencies is empty. □

Thus, to regain decidability we need to restrict the use of $=$, $\neq$, and function symbols. But do we have to forbid them entirely, as in [1]? Below we show that the restriction can be much weaker than that. The resulting class of dependencies extends that of Proposition 2 in two ways: the second form allows equalities on the source side and unrestricted use of fresh variables on the target side.

Proposition 4. Comp *is decidable (*2ExpTime*-complete) if the second mapping uses no function symbols and has dependencies of the following two forms:*

$$\varphi(\bar{x}) \wedge \alpha^{\neq}(\bar{x}) \Rightarrow \psi(\bar{x}, \bar{y}) \wedge \alpha^{=}(\bar{x}, \bar{y}), \quad \varphi(\bar{x}) \wedge \alpha^{=}(\bar{x}) \Rightarrow \psi(\bar{x}, \bar{y}) \wedge \alpha^{\neq}(\bar{x}, \bar{y}).$$

Proof. Let $\mathcal{M}$, $\mathcal{M}'$, S, T an instance of Comp. Rather than using Lemma 1, as in Proposition 2, we note that we always evaluate dependencies of $\mathcal{M}'$ over pairs of trees of the form (I, T). Hence, in each dependency we can replace the pattern $\psi(\bar{x}, \bar{y})$ with a formula $\gamma^{=}(\bar{x}, \bar{y})$ that is a disjunction of exponentially many conjunctions of equalities between variables and data values used in T, corresponding to all ways of matching $\psi(\bar{x}, \bar{y})$ in T. Moreover, by the safety assumption we know that in each conjunction all variables $\bar{y}$ occur in at least one equality. Moving all data comparisons to the target side, we get

$$\varphi(\bar{x}) \Rightarrow \neg\alpha^{\neq}(\bar{x}) \vee \big(\gamma^{=}(\bar{x}, \bar{y}) \wedge \alpha^{=}(\bar{x}, \bar{y})\big), \quad \varphi(\bar{x}) \Rightarrow \neg\alpha^{=}(\bar{x}) \vee \big(\gamma^{=}(\bar{x}, \bar{y}) \wedge \alpha^{\neq}(\bar{x}, \bar{y})\big).$$

Dependencies of the first form can be turned into non-mixing constraints as in the proof of Proposition 2. In dependencies of the second form we can also eliminate variables $\bar{y}$. Recall that $\gamma^{=}(\bar{x}, \bar{y}) = \gamma_1^{=}(\bar{x}, \bar{y}) \vee \cdots \vee \gamma_m^{=}(\bar{x}, \bar{y})$, and in each $\gamma_i^{=}(\bar{x}, \bar{y})$ each of the variables $\bar{y}$ appears in an equality with a data value; we can assume that it is exactly one equality, because otherwise $\gamma_i^{=}(\bar{x}, \bar{y})$ is not satisfiable and can be dropped. Hence, we can replace $\gamma^{=}(\bar{x}, \bar{y}) \wedge \alpha^{\neq}(\bar{x}, \bar{y})$ with the disjunction of formulas $\gamma_i^{=}(\bar{x}, \bar{y}) \wedge \alpha^{\neq}(\bar{x}, \bar{y})$ for $i = 1, \ldots, m$. In each $\gamma_i^{=}(\bar{x}, \bar{y}) \wedge \alpha^{\neq}(\bar{x}, \bar{y})$ we eliminate $\bar{y}$ independently, replacing all their occurrences with the data values assigned to them by $\gamma_i^{=}(\bar{x}, \bar{y})$. Because $\neg\alpha^{=}(\bar{x})$ is equivalent to a positive boolean combination of inequalities, we thus transform dependencies of the second form into non-mixing constraints. Let Σ' be the resulting set of constraints.

Since $\mathcal{M}$ can use function symbols, we can assume that its dependencies do not introduce new variables on the target side. Let Δ be the set of all terms occurring in $\mathcal{M}$ with variables substituted with data values used in S. Restrict $\mathbb{D}$ to data values used in S or T, and $|\Delta|$ fresh data values. Iterate through all consistent valuations of terms in Δ. For each valuation consider formulas $\psi(\bar{u}) \wedge \beta(\bar{u})$ for all dependencies $\varphi(\bar{x}) \wedge \alpha(\bar{x}) \Rightarrow \psi(\bar{x}) \wedge \beta(\bar{x})$ of $\mathcal{M}$ and all $\bar{u} \in \mathbb{D}$ such that $S \models \varphi(\bar{u}) \wedge \alpha(\bar{u})$. It one of $\beta(\bar{u})$ does not hold, discard this valuation of Δ. Otherwise, let Ψ be the set of all such $\psi(\bar{u})$. Now it remains to check that there exists a tree I conforming to the target schema of $\mathcal{M}$ and the source schema of

$\mathcal{M}'$, such that $I \models \Psi$ and $I \models \Sigma'$. This is essentially an instance of the problem from Theorem 1, except that we have an intersection of two schemas. This is not problematic, because the setting in [6] abstracts schemas as automata, which are closed under intersection and the resulting automaton is quadratic. The total size of patterns from Ψ and constraints from Σ' is exponential, but the size of involved patterns has not increased. Hence, the problem is in 2ExpTime. The lower bound holds already for mappings without data comparisons [1]. □

Problem 2. Unrestricted use of either $=$ or $\neq$ on both sides of the second mapping immediately leads to undecidability of Comp [1], but it would be very interesting to see if one can allow dependencies of the form

$$\varphi'(\bar{x}) \Rightarrow \psi'(\bar{x}, \bar{y}) \wedge \alpha^{=}(\bar{x}, \bar{y}) \wedge \alpha^{\neq}(\bar{x}, \bar{y}) \,,$$

which is not excluded by the existing undecidability results.

Problem 3. When the mappings are fixed, the algorithm from Proposition 4 runs in exponential time. One easily transfers the NP lower bound from the relational case [9], but no tighter bound is known. This leaves a substantial gap.

4 Certain answers

The central problem of data exchange is answering queries over the target. Given that the solution to a given source instance is typically not unique, the adopted semantics is that of *certain answers*: we only keep those tuples that are returned over each solution to the given source instance. That is, query answering in the context of data exchange boils down to the following problem:

Problem: Cert
Input: mapping $\mathcal{M}$, tree pattern query $q(\bar{x})$, tree S, tuple $\bar{a}$
Question: Does $T \models q(\bar{a})$ hold for each solution T for S wrt. $\mathcal{M}$?

In [1] it is shown that as soon as one forbids inequality in queries, the problem becomes decidable. This proof goes by reduction to a more fundamental problem of certain answers in incomplete information scenarios [3], which in turn can be seen as an instance of query containment for unions of conjunctive queries over trees in the presence of a schema [4]. In [3] the problem is solved by resorting to first order types of fixed rank, which gives data complexity in coNP, but apparently very high combined complexity, not even determined in the paper. Here we show how to use the results on non-mixing constraints to pinpoint the exact complexity, and also to extend decidability to a larger class of queries.

Proposition 5. Cert *is decidable (*2ExpTime*-complete) for unions of queries of the form* $\exists \bar{y}\ \varphi(\bar{x}, \bar{y}) \wedge \alpha^{=}(\bar{x}, \bar{y})$ *and* $\exists \bar{y}\ \varphi(\bar{x}, \bar{y}) \wedge \alpha^{\neq}(\bar{x}, \bar{y})$.

Proof. Let $\mathcal{M}$, S, $q(\bar{x})$, $\bar{u} \in \mathbb{D}$ be an instance of Cert as in the statement. Construct a set Σ' of non-mixing constraints equivalent to the negation of $q(\bar{u})$:

for disjuncts $\exists \bar{y}\ \varphi(\bar{u}, \bar{y}) \wedge \alpha^{=}(\bar{u}, \bar{y})$ and $\exists \bar{y}\ \varphi(\bar{u}, \bar{y}) \wedge \alpha^{\neq}(\bar{u}, \bar{y})$, take constraints $\varphi(\bar{u}, \bar{y}) \Rightarrow \neg\alpha^{=}(\bar{u}, \bar{y})$ and $\varphi(\bar{u}, \bar{y}) \Rightarrow \neg\alpha^{\neq}(\bar{u}, \bar{y})$ with negation pushed down to the atoms. Then proceed exactly like in Proposition 4 to reduce to the problem in Theorem 1 and get the 2ExpTime upper bound. The lower bound follows from the proof of Theorem 1. □

The upper bound transfers to the certain answers problem in the incomplete information scenario, as the latter can be easily seen as a special case of Cert; the bound is also tight, by the same argument as above. In fact, for such queries even validity with respect to a schema is 2ExpTime-hard.

Problem 4. The lower bound in Proposition 5 requires both $=$ and $\neq$. For queries using only $=$, the exact complexity remains an open question, particularly intriguing for the incomplete information scenario.

Problem 5. The upper bound above relies on an algorithm whose data complexity is exponential. We also know that there is, at least for queries using only $=$, a nondeterministic algorithm with polynomial data complexity, but undetermined combined complexity. Can we have an algorithm with both data complexity and combined complexity optimized? What does that even mean in a situation when data complexity is pinpointed in the nondeterministic model, and combined complexity in the deterministic model?

Acknowledgments. We thank Wojtek Czerwiński and Paweł Parys for fruitful discussions. A large part of this work was done while the second author was visiting researcher at Université Paris-Est Marne-la-Vallée.

References

1. Shun'ichi Amano, Claire David, Leonid Libkin, and Filip Murlak. XML schema mappings: Data exchange and metadata management. *J. ACM*, 61(2):12:1–12:48, 2014.
2. Marcelo Arenas and Leonid Libkin. XML data exchange: Consistency and query answering. *J. ACM*, 55(2), 2008.
3. Pablo Barceló, Leonid Libkin, Antonella Poggi, and Cristina Sirangelo. XML with incomplete information. *J. ACM*, 58(1):4, 2010.
4. Henrik Björklund, Wim Martens, and Thomas Schwentick. Optimizing conjunctive queries over trees using schema information. In *Proc. MFCS*, pages 132–143, 2008.
5. Mikolaj Bojanczyk, Leszek Aleksander Kolodziejczyk, and Filip Murlak. Solutions in XML data exchange. *J. Comput. Syst. Sci.*, 79(6):785–815, 2013.
6. Wojciech Czerwiński, Claire David, Filip Murlak, and Paweł Parys. Reasoning about integrity constraints for tree-structured data. In *Proc. ICDT*, pages 20:1–20:18, 2016.
7. Claire David, Piotr Hofman, Filip Murlak, and Michal Pilipczuk. Synthesizing transformations from XML schema mappings. In *Proc. ICDT*, pages 61–71, 2014.
8. 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.
9. 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.

A Discussion of Theorem 1

In the original setting of [6], non-mixing constraints are based on conjunctive queries over label tests and relations child, descendent, next sibling, and following sibling, rather than tree patterns, and schemas are modelled as tree automata. In this respect our setting is much weaker, which means that we need to reprove the lower bound. This is done in Lemma **??** below. The proof is a bit more tedious but not much different.

On the other hand, the original setting allows equalities with constants only in $\eta^{=}(\bar{x})$, but not in $\varphi(\bar{x})$ or $\eta^{\neq}(\bar{x})$. This is happens to be a mild extension, but it does require reexamining the proof of the upper bound. Formally, allowing free use of equalities with constants affects the compositionality lemma for conjunctive queries (Lemma 4 in [6]), the bound on the data cut (Lemma 5 in [6]), and the construction of the register automaton (proof of Theorem 1 in [6]). For compositionality the unary predicates are entirely irrelevant, so the changes in the proof are cosmetic. For the bound on the datacut the proof works almost as is: the only place where the change matters is when the proof says that replacing some occurrences of data values with fresh nulls does not violate constraints with inequalities only, because by making some stored values different we cannot make an inequality false. To deal with equalities with constants it is enough to assume that data values used as constants in the constraints are never replaced in the tree. This increases the data cut by the number of such constants, but does not change its order of magnitude. The construction of the register automaton is only modified in this respect that while matching a tree pattern, a node of the pattern that contains a constant (rather than a variable) can be matched at the currently seen tree node only if the node stores the same data value that is kept in the register representing the constant. Note that this extension does not rely on the fact that we have restricted the left-hand sides of constraints to tree patterns.

Let us now move to the lower bound. The following lemma adapts the reduction given in [6] to our setting. As the original proof already uses only tree patterns, the main change is that we need to rely on DTDs rather than automata.

Lemma 2. *It is* 2ExpTime*-hard to decide if there exists a tree conforming to a given DTD and satisfying a given set of non-mixing constraints without* $\neq$*.*

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.

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 duplicate transitions available in existential states.

We show that for each alternating Turing machine M of the form described above and each $n \in \mathbb{N} - \{0\}$, we can construct (in polynomial time) a DTD and a set of non-mixing constraints of the form $\varphi(\bar{x}) \Rightarrow \alpha^{=}(\bar{x})$, such that there is a tree conforming to the DTD and satisfying the set of constraints if and only if the machine M accepts the empty word without leaving the first 2^n cells of the tape. More precisely, each such tree describes a run tree of M on the empty word. Below we specify how the run tree is encoded, simultaneously explaining how these properties are ensured by the DTD and the constraints.

Let $\Delta \subseteq Q \times A \times Q \times A \times \{\mathsf{left}, \mathsf{stay}, \mathsf{right}\}$ be the set of transition rules of the machine M. In the encoding, nodes labelled by $\Delta \cup \{r\}$ form a binary tree: the root is labelled with r, and each $\Delta \cup \{r\}$-labelled node has zero or two Δ-labelled children. This tree is called the *skeleton* and has the same shape as the encoded run tree. The DTD can ensure that:

– in each skeleton node the source state of the transition equals either the target state of the transition in the parent, or the initial state of M—if the parent is the root;
– transitions in siblings are different (as required in the definition of run trees);
– target states are accepting in leaves and not accepting in internal nodes.

Let us use $\Gamma \times 2^{\{\mathsf{prev},\mathsf{curr}\}}$ to represent the tape alphabet Γ of M enriched with the information on the current and previous position of the head. To each node of the skeleton we additionally attach a path such that the labels of nodes on this path form a word matching the regular expression

$$\$\left(\Gamma \times \left\{\emptyset\right\}\right)^{*}\left(\Gamma \times \left\{\{\mathsf{prev}\},\{\mathsf{curr}\}\right\}\right)^{\leq 1}\left(\Gamma \times \left\{\{\mathsf{prev},\mathsf{curr}\}\right\}\right)^{+}\#\,;$$

this path is called the *configuration path* and it encodes the configuration of M assigned to the corresponding node of the skeleton (which is simultaneously a node of the run tree), as a sequence of cell contents. Under this encoding, the current position of the head is in the first cell that includes curr, and the previous position of the head is in the first cell that includes prev. This makes formulating queries cumbersome, but is necessary to ensure the correct order of labels with the DTD.

Each $\Gamma \times 2^{\{\mathsf{prev},\mathsf{curr}\}}$-labelled node on the configuration path has additionally children with labels $\mathsf{b}_1, \mathsf{b}_2, \ldots, \mathsf{b}_n$. They are used to address the cells of the configuration. To this end, we ensure that in each configuration path these nodes

store values of an n-bit binary counter, using two distinct data values $0, 1 \in \mathbb{D}$. We store $0 \ldots 00$ under the first node of the configuration path, $0 \ldots 01$ under the second node, and so on, until $1 \ldots 11$ under the last node; thus we have 2^n nodes in the configuration path, which equals the length of the tape. To enforce this behaviour we use constraints. To ensure correctness of the first and the last value of the counter, we use constraints

$$\$/\_\big[\mathsf{b}_1(x_1), \mathsf{b}_2(x_2), \ldots, \mathsf{b}_n(x_n)\big] \Rightarrow x_1 = 0 \wedge x_1 = 0 \wedge \ldots \wedge x_n = 0\,,$$
$$\_\big[\#, \mathsf{b}_1(x_1), \mathsf{b}_2(x_2), \ldots, \mathsf{b}_n(x_n)\big] \Rightarrow x_1 = 1 \wedge x_1 = 1 \wedge \ldots \wedge x_n = 1\,,$$

where ρ/φ stands for $\rho[\varphi]$. To ensure correct evolution of the counter, we add

$$\_\big[\mathsf{b}_1(x_1), \mathsf{b}_2(x_2), \ldots, \mathsf{b}_n(x_n), \_[\mathsf{b}_1(y_1), \mathsf{b}_2(y_2), \ldots, \mathsf{b}_n(y_n)]\big] \Rightarrow \alpha^{=}_{succ}(\bar{x}, \bar{y})$$

where $\alpha^{=}_{succ}(\bar{x}, \bar{y})$ is a disjunction of n conjunctions of $\mathcal{O}(n)$ equalities among variables and constants $0, 1$, expressing that $\bar{x}, \bar{y}$ are consecutive counter values.

Finally, to avoid $\neq$ in constraints, each $\Gamma \times 2^{\{\mathsf{prev},\mathsf{curr}\}}$-labelled node has also children with labels $\mathsf{b}'_1, \mathsf{b}'_2, \ldots, \mathsf{b}'_n$, storing the bit-negation of the counter value:

$$\_[\mathsf{b}_i(x), \mathsf{b}'_i(y)] \Rightarrow (x = 0 \wedge y = 1) \vee (x = 1 \wedge y = 0)\,.$$

To ensure that the previous position of the head matches the current position of the head in the previous configuration, we use constraints

$$\_\big[\$/\varphi_{\mathsf{curr}}(\bar{x}), \_/\$/\varphi_{\mathsf{prev}}(\bar{y})\big] \Rightarrow x_1 = y_1 \wedge \cdots \wedge x_n = y_n\,,$$

where $\varphi_{\mathsf{curr}}(\bar{x})$ selects into $\bar{x}$ the counter value from the earliest cell that contains curr in its label, and similarly for $\varphi_{\mathsf{prev}}(\bar{y})$. Because of the way we encode configurations, we need multiple variants of $\varphi_{\mathsf{curr}}(\bar{x})$:

$$//a/b\big[\mathsf{b}_1(x_1), \ldots, \mathsf{b}_n(x_n)\big]\,, \qquad b\big[\mathsf{b}_1(x_1), \ldots, \mathsf{b}_n(x_n)\big]$$

for all $a \in \Gamma \times \{\emptyset, \{\mathsf{prev}\}\}$, $b \in \Gamma \times \{\{\mathsf{curr}\}, \{\mathsf{prev}, \mathsf{curr}\}\}$, and similarly for $\varphi_{\mathsf{prev}}(\bar{y})$ with curr and prev swapped.

To ensure that in the initial configuration the tape is empty (filled with the blank symbol $\mathsf{blank} \in \Gamma$) and the head is over the first cell, we add constraints

$$r/\$//a \Rightarrow 0 = 1$$

for all $a \in \Gamma \times 2^{\{\mathsf{prev},\mathsf{curr}\}} - \big\{(\mathsf{blank}, \{\mathsf{prev}, \mathsf{curr}\})\big\}$; there is no previous configuration, but we fix the previous position of the head to the first cell for concreteness.

Next, we ensure that the transition is applicable and that it is carried out correctly. For this, we need to check the move of the head encoded by prev and curr, the tape symbol at the previous position of the head, and the tape symbol at the current position of the head in the previous configuration. The first two conditions can be checked with constraints

$$\tau/\$/a \Rightarrow 0 = 1\,, \qquad \tau/\$//b/c \Rightarrow 0 = 1\,,$$

where $\tau \in \Delta$ and $a, b, c \in \Gamma \times 2^{\{\mathsf{prev},\mathsf{curr}\}}$ range through all illegal possibilities: for instance, if $\tau = (\sigma, q, \sigma', q', \mathsf{left})$, then $a \in \Gamma \times \big\{\{\mathsf{prev}\}, \{\mathsf{prev}, \mathsf{curr}\}\big\}$ and

– $b \in \Gamma \times \big\{\emptyset, \{\mathsf{prev}\}\big\}$, $c \in \Gamma \times \big\{\{\mathsf{prev}, \mathsf{curr}\}\big\}$ (head moves right or stays), or
– $b \in \Gamma \times \big\{\{\mathsf{curr}\}\big\}$, $c \in \big(\Gamma - \{\sigma'\}\big) \times \big\{\{\mathsf{prev}, \mathsf{curr}\}\big\}$ (wrong symbol written).

The third condition can be checked with constraints

$$\_\big[\tau, \$/a\big] \Rightarrow 0{=}1\,, \qquad \_\big[\tau, \$//b/c\big] \Rightarrow 0{=}1\,,$$

where $\tau \in \Delta$ and $a, b, c \in \Gamma \times 2^{\{\mathsf{prev},\mathsf{curr}\}}$ range through all illegal possibilities: for instance, if $\tau = (\sigma, q, \sigma', q', \mathsf{left})$, then $a \in \Gamma \times \big\{\{\mathsf{curr}\}, \{\mathsf{prev}, \mathsf{curr}\}\big\}$ and

– $b \in \Gamma \times \big\{\emptyset, \{\mathsf{prev}\}\big\}$, $c \in \big(\Gamma - \{\sigma\}\big) \times \big\{\{\mathsf{curr}\}, \{\mathsf{prev}, \mathsf{curr}\}\big\}$.

Finally, using the bit-negation of the counter, we ensure that the content of the tape is preserved, except for the symbol under the head:

$$\_\big[\$/a,\ \_/\$/a'\big] \Rightarrow 0{=}1\,,$$
$$\_\Big[\$//b/c\big[\mathsf{b}_1(x_1), \ldots, \mathsf{b}_n(x_n)\big],\ \_/\$//c'\big[\mathsf{b}'_1(y_1), \ldots, \mathsf{b}'_n(y_n)\big]\Big] \Rightarrow \bigvee_{i=1}^{n} x_i{=}y_i\,,$$

for all $a \in \Gamma \times \big\{\emptyset, \{\mathsf{prev}\}\big\}$, $a' \in \Gamma \times 2^{\{\mathsf{prev},\mathsf{curr}\}}$ such that the tape symbols in a and a' are different, and all $b, c, c' \in \Gamma \times 2^{\{\mathsf{prev},\mathsf{curr}\}}$ such that the tape symbols in c and c' are different and it is not the case that $b \in \Gamma \times \big\{\emptyset, \{\mathsf{prev}\}\big\}$ and $c \in \Gamma \times \big\{\{\mathsf{curr}\}, \{\mathsf{prev}, \mathsf{curr}\}\big\}$. □

B Other lower bounds

Lemma 3. Cons *is* 2ExpTime*-hard even for mappings with dependencies of the form* $\varphi(\bar{x}) \Rightarrow \psi(\bar{x}) \wedge \alpha^{\neq}(\bar{x})$.

Proof. We reduce to Cons the satisfiability problem for non-mixing constraints that use no constants in the patterns and have disjunctions of conjunctions of equalities on the target side. From the proof of Lemma **??** it follows that satisfiability is 2ExpTime-hard already for such constraints. It might seem counterintuitive that we have $=$ in the constraints and $\neq$ in the dependencies. The reason is that we can eliminate $=$ on the target side entirely, using target schema gadgets, but we need $\neq$ to simulate constants, which are not allowed in dependencies.

The reduction is very simple. Let $d_1, d_2, \ldots, d_m \in \mathbb{D}$ be the constants used in the constraints (on the right side). We enrich the input DTD by including children of the root with labels $\mathsf{const}_1, \mathsf{const}_2, \ldots, \mathsf{const}_m$; the resulting DTD is the source schema of the mapping we are constructing. The target schema, for now, generates a single tree with r-labelled root whose m children have labels $\mathsf{const}_1, \mathsf{const}_2, \ldots, \mathsf{const}_m$. In the set of dependencies we include

$$r[\mathsf{const}_i(x), \mathsf{const}_j(y)] \Rightarrow x{\neq}y\,,$$

$$\mathsf{const}_i(x) \Rightarrow \mathsf{const}_i(x)\,,$$

for all distinct $i, j \in \{1, 2, \ldots, m\}$. Then, we replace each constraint $\varphi(\bar{x}) \Rightarrow \alpha^{=}(\bar{x})$ with a dependency

$$\varphi(\bar{x}) \Rightarrow r[\mathsf{const}_1(y_1), \mathsf{const}_2(y_2), \ldots, \mathsf{const}_m(y_m), \psi(\bar{x}, \bar{y})]\,,$$

where $\psi(\bar{x}, \bar{y})$ is the tree pattern obtained as in the proof of Proposition 1 from the disjunction of conjunctions of equalities $\beta^{=}(\bar{x}, \bar{y})$, which is just $\alpha^{=}(\bar{x})$ with each d_i replaced with y_i. This is accompanied by adding appropriate equality gadgets to the target schema, as in the proof of Proposition 1. □

Lemma 4. *The problem of validity with respect to a schema is* 2ExpTime*-hard for unions of queries of the form* $\exists \bar{y}\ \varphi(\bar{x}, \bar{y}) \wedge \alpha^{=}(\bar{x}, \bar{y})$ *and* $\exists \bar{y}\ \varphi(\bar{x}, \bar{y}) \wedge \alpha^{\neq}(\bar{x}, \bar{y})$.

Proof. Because 2ExpTime is closed under complement, the complement of each 2ExpTime-hard problem is also 2ExpTime-hard. Hence, to obtain the lower bound for validity it suffices to reduce instances of the satisfiability problem for non-mixing constraints produced by the reduction given in the proof of Lemma **??** to non-validity. The schema is the DTD described in Lemma **??**, extended with two nodes labelled with const_0 and const_1 attached to the root. The query is the disjunction of the boolean queries

$$\exists z_0, z_1\ r[\mathsf{const}_0(z_0),\ \mathsf{const}_1(z_1)] \wedge z_0 = z_1\,,$$
$$\exists x, z_0, z_1\ r[//\_(x), \mathsf{const}_0(z_0),\ \mathsf{const}_1(z_1)] \wedge x \neq z_0 \wedge x \neq z_1$$

and one disjunction of boolean queries for each constraint from the proof of Lemma **??**, constructed as follows. For constraints of the form $\varphi(\bar{x}) \Rightarrow 0 = 1$, take

$$\exists \bar{x}\ \varphi(\bar{x})\,.$$

For $\varphi(\bar{x}) \Rightarrow x_1 = 0 \wedge \cdots \wedge x_n = 0$, take

$$\bigvee_{i=1}^{n} \exists \bar{x}, z_0, z_1\ \varphi'(\bar{x}, z_0, z_1) \wedge x_i = z_1\,,$$

where $\varphi'(\bar{x}, z_0, z_1)$ is obtained by extending $\varphi(\bar{x})$ in such a way that z_0 and z_1 are matched to the values stored in the unique nodes with labels const_0 and const_1. Similarly for $\varphi(\bar{x}) \Rightarrow x_1 = 1 \wedge \cdots \wedge x_n = 1$. For the constraint of the form $\varphi(x, y) \Rightarrow (x = 0 \wedge y = 1) \vee (x = 1 \wedge y = 0)$, we can just take

$$\exists x, y\ \varphi(x, y) \wedge x = y\,.$$

For $\varphi(\bar{x}, \bar{y}) \Rightarrow x_1 = y_1 \wedge \cdots \wedge x_n = y_n$, take

$$\bigvee_{i=1}^{n} \exists x, y\ \varphi(x, y) \wedge x_i \neq y_i\,,$$

and for $\varphi(\bar{x}, \bar{y}) \Rightarrow x_1 = y_1 \vee \cdots \vee x_n = y_n$, take

$$\exists x, y\ \varphi(x, y) \wedge x_1 \neq y_1 \wedge \cdots \wedge x_n \neq y_n\,.$$

The only case that requires some attention is $\varphi(\bar{x}, \bar{y}) \Rightarrow \alpha^{=}_{succ}(\bar{x}, \bar{y})$, where the formula $\alpha^{=}_{succ}(\bar{x}, \bar{y})$ expresses that $\bar{x}$ and $\bar{y}$ represent consecutive values of the n-bit binary counter. Assuming that variables only take values in $\{0, 1\}$, we can express its negation as a disjunction of n conjunctions of $\mathcal{O}(n)$ equalities over $\bar{x}$, $\bar{y}$ and 0,1. Hence, we can take the disjunction of queries

$$\exists \bar{x}, \bar{y}, z_0, z_1 \ \varphi'(\bar{x}, \bar{y}, z_0, z_1) \wedge \beta^{=}(\bar{x}, \bar{y}, z_0, z_1)$$

with $\beta^{=}(\bar{x}, \bar{y}, z_0, z_1)$ ranging over these conjunctions with all occurrences of 0 and 1 replaced with z_0 and z_1, respectively. □