Complexity of Typechecking XML Views of Relational Databases

Piotr Wieczorek*

Institute of Computer Science, University of Wrocław,
Joliot-Curie 15, PL-50-383 Wrocław, Poland
piotr.wieczorek@ii.uni.wroc.pl

Abstract. The typechecking problem for transformations of relational data into tree data is the following: given a TreeQL program P (called *transformation*), and a DTD d (called *output type*), decide whether for every database instance D the result of the transformation P of D is of a type consistent with d (see [2]). TreeQL programs with projection-free conjunctive queries and DTDs with arbitrary regular expressions are considered here.
A non-elementary upper bound for the typechecking problem is given in [2] (although in a more general setting, where equality and negation in projection-free conjunctive queries and additional universal integrity constraints are allowed).
In this paper we show that the typechecking problem is in coNEXPTIME.
As an intermediate step we consider the following problem, which can be formulated in a language independent of XML notions. Given a set of triples of the form (φ, k, j), where φ is a projection-free conjunctive query and k, j are natural numbers, decide whether there exists a database D such that for each triple (φ, k, j) in the set, there exists a natural number α, such that there are exactly $k + j * \alpha$ tuples satisfying the query φ in D. Our main technical contribution consists of a NEXPTIME algorithm for the last problem.

1 Introduction

During the last years XML has become the standard in data exchange in the web. Often the actual data resides in relational databases. In order to be published such data should be transformed to XML. XML documents have their *types* — a type is a tree language. There are many formalisms to define a type e.g. RELAX NG, which can define the full class of regular tree languages or DTD/XML Schema, which can define some fragments of this class. Typically, a community agrees on a certain type and then all members of the community publish documents consistent with the type. Here comes the problem of typechecking for transformations of relational data into tree data: for a given transformation, an

* Partially supported by Polish Ministry of Science and Higher Education research project N206022 31/3660, 2006/2009.

output type and a set of integrity constraints, decide whether every database satisfying the integrity constraints is transformed to a tree consistent with the output type. Thus the problem can be parameterized by:

- the class of transformations,
- the class of output tree languages,
- the class of integrity constraints.

Alon et al. [2] present a study of decidability and complexity of many versions of the problem. As a formalism to define transformations the authors introduce TreeQL programs. TreeQL is an abstraction of practical languages such as RXL (SilkRoute [4]). A TreeQL program is a tree with each node labeled with a symbol from a finite alphabet and with a logical formula, which in our paper is always a projection-free conjunctive query. The result of a transformation of a relational structure is a tree reflecting the structure of the program tree, such that each node t of the program tree is substituted by as many nodes as there are tuples in the database satisfying the formula being the label of t. The nodes of the output tree inherit, as their labels, the symbols that label nodes of the program tree. The output type is specified by a DTD — a formalism which puts local restrictions on trees, that is, it restricts how the sequence of child labels of a node looks like.

Decidability results in [2] include a coNEXPTIME upper bound on typechecking TreeQL programs with conjunctive queries (with negation and equality), DTDs with star-free regular languages as the output types and the integrity constraints in FO($\exists^*\forall^*$). When arbitrary regular expressions are allowed in DTDs the authors show decidability of typechecking TreeQL programs with projection-free conjunctive queries [1] (with negation and equality) and integrity constraints in FO($\forall^*$). In the latter case, however, the complexity is prohibitively high – the proof uses a combinatorial argument based on Ramsey's Theorem and yields a non-elementary upper bound. It was left as an open problem in [2] whether the bound can be improved. We show that such an improvement is possible, at least for a restricted case. We show a coNEXPTIME upper bound on the typechecking problem for DTDs with arbitrary regular expressions as the output types and projection-free conjunctive queries in TreeQL programs, but without integrity constraints.

Our approach is as follows. Inspired by the notion of *the modulo property* [2], we perform the reduction of the complement of the typechecking problem to the following problem. Given a set of triples of the form (φ, k, j), where φ is a projection-free conjunctive query and k, j are natural numbers, decide whether there exists a database D such that for each triple (φ, k, j) in the set, there exists a natural number α, such that there are exactly $k + j * \alpha$ tuples satisfying the query φ in D. Notice, that a triple (φ, k, j) is a kind of a constraint on a relational database. We call such constraints *modulo constraints*. Our main technical contribution consists of a NEXPTIME algorithm for the problem of

[1] If conjunctive queries with projections are allowed, the problem (even in our simple setting) is not known to be decidable.

satisfiability of a set of modulo constraints. We use an elementary technique, namely a direct construction of a counterexample database of exponential size (by a counterexample database we mean such a database that is transformed to a tree not in the output tree language).

Related work. Recently the problem of typechecking gained a lot of attention in literature, especially in the context of typechecking XML-to-XML transformations, which, since relational structures can easily be encoded as XML trees, is closely related to ours. In the context we are given input and output tree languages and a transformation and we are asked whether every tree in the input tree language is transformed to a tree in the output language. The problem was studied in [7], where the input and output types were regular tree languages and transformations were expressed by k-pebble transducers. As long as the data values in trees are not considered, the problem is decidable, however the complexity is non-elementary. If the nodes in trees can be equipped with data values from an infinite domain, in addition to the tags from a finite alphabet, then, as it was shown in [3], the problem quickly gets undecidable and in the decidable cases the complexity is rather high. In [5] and [6] Martens and Neven considered transformations in a form of a single top-down traversal of the input tree, during which every node can be replaced by a new tree or deleted. Such transformations can be used for restructuring and filtering rather than for advanced querying, but on the other hand, the obtained complexity results range from EXPTIME to PTIME.

Outline of the paper. The rest of the paper is organized as follows. In Sect. 2 we give the necessary preliminaries. In a short Sect. 3 we state Theorem 1, which is our main theorem, and formulate an intermediate result – the main lemma needed for the proof of Theorem 1. In Sect. 4, which is the main technical part, we prove the intermediate result. Finally, in Sect. 5 we use some of the ideas from [2] and show how the intermediate result implies the main result.

2 Preliminaries

XML and XML Types. We abstract XML documents as ordered, unranked, finite trees whose nodes are labeled with symbols from some finite alphabet Σ (see Fig. 2). We denote the label of the node v by $\mathrm{lab}(v)$ and the root node of the tree t by $\mathrm{root}(t)$. *A Document Type Definition (DTD)* is a way of defining a tree language. A DTD d defines for each symbol $\sigma \in \Sigma$ a regular language $d(\sigma)$. We say that a tree t is consistent with d if for every node v of t with children $v_1, \ldots, v_n$ the word $lab(v_1) \ldots lab(v_n)$ is in the regular language $d(lab(v))$. If v is a leaf, then the empty word ϵ has to be in $d(lab(v))$. The language of trees consistent with a DTD d is denoted by $L(d)$.

Databases and queries. Let S be a vocabulary consisting of relational symbols. *A database over S* is a finite structure over S ([1]). We denote the domain of a structure $\mathcal{A}$ by $\mathrm{dom}(\mathcal{A})$. In the paper we consider also structures over vocabularies containing constant symbols. *A projection-free conjunctive query (projection-free CQ)* $\varphi(x_1, \ldots, x_n)$ is a conjunction of atomic formulas over vo-

cabulary S. By Vars(φ) we denote the set of variables of φ (note that all variables in projection-free CQs are free). Let $\varphi(\bar{x})$ be a projection-free CQ, $\mathcal{A}$ a database and $\bar{a}$ a tuple of elements in $\mathcal{A}$. We define $\mathcal{A} \models \varphi(\bar{a})$ in the usual way.

TreeQL and typechecking. The following definitions come from [2], but are tailored for our setting.

Definition 1. *1. A TreeQL program is an ordered, unranked tree P with labels.*
- *the root is labeled with an element from alphabet Σ.*
- *every non-root element node is labeled with a pair (σ, φ), where $\sigma \in \Sigma$ and φ is a projection-free CQ. The formula in a node v is denoted by* formula(v).
- Vars(formula(v)) $\subseteq$ Vars(formula(v')), *for all non-root nodes v and v', where v' is a descendant of v.*

2. Let $\mathcal{A}$ be a database and P a TreeQL program. A tree $P(\mathcal{A})$ generated from $\mathcal{A}$ is defined as follows:
- *The root is (root(P), $\emptyset$).*
- *The non-root nodes consist of pairs (v, θ), where v is a non-root node of P and θ is a substitution for variables* Vars(formula(v)), *such that $\mathcal{A} \models \varphi[\theta]$, for every formula φ labeling v or labeling an ancestor of v in P.*
- *The edges in $P(\mathcal{A})$ are $((v, \theta), (v', \theta'))$ such that v' is a child of v in P and θ' is an extension of θ (i.e. θ' agrees with θ on variables in the formula in v).*
- *Sibling nodes in $P(\mathcal{A})$ are ordered as follows: if v and v' are siblings in P and v occurs before v', then all nodes (v, θ) occur before all nodes (v', θ') in $P(\mathcal{A})$. For a given v in P, the ordering of nodes (v, θ) and (v, θ') is irrelevant in our setting, so it is not considered here (see remark below).*
- *Finally, the label of a node (v, θ) is the Σ-label of v in P.*

Remark. We use the following observation from [2]. If d is a DTD then d does not distinguish among trees $P(\mathcal{A})$ for distinct orderings of the nodes (v, θ) and (v, θ'), for each v in P. As we consider DTDs as XML types, we abstract from the ordering of such nodes.

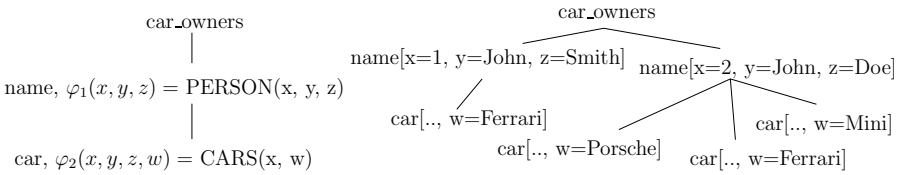
Definition 2. *A TreeQL program P* typechecks *with respect to an output type d iff $P(\mathcal{A}) \subseteq L(d)$, for every database $\mathcal{A}$.*

Example 1. Consider a database $\mathcal{A}$ containing information about car owners, with two relations `PERSON(Id, FirstName, LastName)` and `CARS(Id, Car)`:

PERSON	Id	FirstName	LastName
	1	John	Smith
	2	John	Doe

CARS	Id	Car
	1	Ferrari
	2	Porsche
	2	Ferrari
	2	Mini

In Fig. 1 we present a program R and a tree $R(\mathcal{A})$ resulting from the transformation of the database $\mathcal{A}$ by the program R. The tree $R(\mathcal{A})$ is consistent with the following DTD d: d(`car_owners`) = `name`*, d(`name`) = `car`*, d(`car`) = ϵ.



(a) The program R.

(b) The XML tree $R(\mathcal{A})$ –the annotations of substitutions (in brackets) are not part of the tree.

Fig. 1. A TreeQL query and its result (Example 1)

3 Our main result

Now we are able to formulate our main theorem.

Theorem 1 (Main Theorem). *The problem of typechecking a TreeQL program with projection-free conjunctive queries w.r.t. DTD with arbitrary regular expressions is in coNEXPTIME.*

The rest of the paper is devoted to the proof of Theorem 1.

Definition 3. *A set of modulo constraints Γ is a finite set of triples of the form (φ, k, j), where φ is a projection-free conjunctive query and k, $j \in \mathbb{N}$. We say that a database $\mathcal{A}$ satisfies a set of modulo constraints Γ (we write $\mathcal{A} \models \Gamma$) iff for each $(\varphi_i, k_i, j_i) \in \Gamma$ there exists $\alpha_i \in \mathbb{N}$ such that:*

$$|\{\bar{t} \,|\, \mathcal{A} \models \varphi_i(\bar{t})\}| = k_i + (\alpha_i * j_i)$$

Of course, we assume that $0 \in \mathbb{N}$, so in particular Γ can contain some triples of the form $(\varphi, k, 0)$.

Now, we formulate the intermediate result, which is the main technical contribution of this paper.

Theorem 2 (Intermediate Result). *Let Γ be a set of constraints with projection-free conjunctive queries. The problem whether there exists a database $\mathcal{A}$ such that $\mathcal{A} \models \Gamma$ is in NEXPTIME.*

In Sect. 4 we prove the intermediate result, and in Sect. 5 we show how it implies the main result.

4 Proof of the intermediate result

We use the following notation. Let Γ be a set of modulo constraints, then: Γ_{CONST} is the set of *constant* constraints: $\Gamma_{\text{CONST}} = \{(\varphi, k, j) \in \Gamma \mid j = 0\}$, and Γ_{PROP} is the set of *proper* constraints: $\Gamma_{\text{PROP}} = \{(\varphi, k, j) \in \Gamma \mid j > 0\}$. Of

course, we have: $\Gamma = \Gamma_{\text{CONST}} \cup \Gamma_{\text{PROP}}$. In the sequel, when talking about the modulo constraints we sometimes omit the word *modulo*.

A canonical structure $\mathcal{C}_\varphi$ for a projection-free CQ φ is defined as usually: elements of $\mathcal{C}_\varphi$ are variables and constants of φ and relations of $\mathcal{C}_\varphi$ consist of tuples of variables and constants from conjuncts of φ.

Outline of the proof. We present an algorithm, which for a satisfiable set of (modulo) constraints $\Gamma = \Gamma_{\text{CONST}} \cup \Gamma_{\text{PROP}}$ constructs a witness database $\mathcal{B}$ of exponential size w.r.t. the size of Γ. The general idea of the algorithm is to guess a database $\mathcal{A}_{\text{CONST}}$ satisfying Γ_{CONST} and then to satisfy the proper constraints Γ_{PROP} one by one, by extending the database $\mathcal{A}_{\text{CONST}}$ with some number of copies of canonical structures of the formulas of Γ_{PROP}. But won't satisfying a constraint in such a way cause some of the constraints, which have been already satisfied, to fail? In Sect. 4.2 we show that the problem can be overcome if constraints are in some normal form, and if the order in which we try to satisfy them is correct. Earlier, in Sect. 4.1 we show that, for each set of modulo constraints, one can construct, in NEXPTIME an equisatisfiable set of constraints which is in the desired normal form.

Operation REPLACE. Suppose that formulas $\varphi_1, \ldots, \varphi_n$ in some constraints $(\varphi_i, k_i, j_i) \in \Gamma_{\text{PROP}}$ (for $i \in \{1, \ldots, n\}$) are equivalent (i.e. canonical structures C_{φ_i} are isomorphic). We define a single constraint (φ_1, k, j), where j is the least common multiple of the numbers $j_1, \ldots, j_n$ (recall that $j_i > 0$ in constraints in Γ_{PROP}) and k is the smallest number such that, for each $i = 1, \ldots, n$, there exists $\alpha_i \in \mathbb{N}$ such that it holds $k = k_i + \alpha_i * j_i$. Using Chinese Remainder Theorem it is possible to show that if the constraints are satisfiable such a number k exists, otherwise we know that the constraints are inconsistent and the algorithm stops.

This allows us to define an operation REPLACE. The operation transforms a set of proper constraints Γ by replacing each set of constraints having equivalent formulas with a single constraint, while preserving satisfiability. After applying the operation, there are no two distinct constraints in REPLACE(Γ) with equivalent formulas. Notice that equivalence of CQs is in NP, so we do not run out of time.

Lemma 1. *For every set of proper constraints* Γ *and every database* $\mathcal{A}$ *we have* $\mathcal{A} \models \Gamma$ *iff* $\mathcal{A} \models \text{REPLACE}(\Gamma)$.

Dealing with constant constraints. In the following lemma we show that given a satisfiable set of constraints $\Gamma = \Gamma_{\text{CONST}} \cup \Gamma_{\text{PROP}}$, it is possible to guess a database $\mathcal{A}_{\text{CONST}}$ of at most exponential size, such that $\mathcal{A}_{\text{CONST}} \models \Gamma_{\text{CONST}}$.

Lemma 2. *Let* $\Gamma = \Gamma_{\text{CONST}} \cup \Gamma_{\text{PROP}}$ *be a set of constraints. If there exists a database* $\mathcal{A}$ *such that* $\mathcal{A} \models \Gamma$ *then there exists a database* $\mathcal{A}_{\text{CONST}}$ *such that* $\mathcal{A}_{\text{CONST}} \models \Gamma_{\text{CONST}}$. *The size of* $\mathcal{A}_{\text{CONST}}$ *is at most exponential w.r.t.* $|\Gamma|$.

Proof (sketch). Consider the database $\mathcal{A}$ and the set $\Gamma = \Gamma_{\text{CONST}} \cup \Gamma_{\text{PROP}}$. Since $\mathcal{A} \models \Gamma$ obviously we have $\mathcal{A} \models \Gamma_{\text{CONST}}$. Let $\mathcal{A}_{\text{CONST}}$ be a substructure of $\mathcal{A}$ which consists exactly of the elements from $\mathcal{A}$ that are in some tuple satisfying a formula in a constraint from Γ_{CONST} or are a constant in Γ.

The number of such elements is bounded by the sum of the number of constants in Γ and the number of constraints in Γ_{CONST} multiplied by the maximal number of variables in formulas in Γ_{CONST} and the value of the maximal number k from Γ_{CONST}. Hence, the size of $\mathcal{A}_{\mathrm{CONST}}$ is at most exponential w.r.t. $|\Gamma|$. Clearly, $\mathcal{A}_{\mathrm{CONST}}$ satisfies Γ_{CONST}. □

From now on we assume that the set of constraints $\Gamma = \Gamma_{\mathrm{CONST}} \cup \Gamma_{\mathrm{PROP}}$ and the database $\mathcal{A}_{\mathrm{CONST}}$ are fixed. All databases which we are going to consider will be superstructures of $\mathcal{A}_{\mathrm{CONST}}$. We extend the vocabulary provided by Γ with a new set of constants $\mathrm{dom}(\mathcal{A}_{\mathrm{CONST}})$, interpreted as elements of $\mathcal{A}_{\mathrm{CONST}}$.

4.1 Construction of a modified set of constraints

The algorithm starts the modifications with a set Γ_{PROP} and goes through three steps. Each of these steps will produce an equisatisfiable set of constraints that will be the input of the next step. All steps are of a similar structure:

1. For each constraint t from the input set, a set Γ_t of new constraints is generated.
2. The set $\mathrm{REPLACE}(\bigcup \Gamma_t)$ is returned as the output of the step.

Step 1. Each constraint $t = (\varphi, k, j)$ asserts the existence of exactly $k + \alpha * j$, for some $\alpha \in \mathbb{N}$, tuples $\bar{a}$ of elements of a database such that $\varphi(\bar{a})$ is true. In this step we want to fix which variables are substituted with elements of $\mathcal{A}_{\mathrm{CONST}}$. Thus we have to produce separate constraints for each (possibly partial) substitution of variables of φ with elements of $\mathcal{A}_{\mathrm{CONST}}$. Additionally, we ensure that such a substitution is final i.e. we forbid the substitution of remaining variables to the constants in the resulting constraints.

Recall the basic intuition behind the algorithm – for each proper constraint, we want to extend $\mathcal{A}_{\mathrm{CONST}}$ with copies of the canonical structure of the formula in the constraint. After Step 1 it is clear how to do it – the elements of the canonical structure corresponding to the constants should be identified with elements of $\mathcal{A}_{\mathrm{CONST}}$ and the other elements should be fresh.

Consider a constraint $t = (\varphi, k, j) \in \Gamma_{\mathrm{PROP}}$. We define Γ_t to be the set of triples of the form $(\psi_\theta, k_\theta, j)$, for each $V \subseteq \mathrm{Vars}(\varphi)$ and $\theta \colon V \to \mathrm{dom}(\mathcal{A}_{\mathrm{CONST}})$, where:

$$\psi_\theta = \varphi[\theta] \wedge \mathrm{NotConstants}(\mathrm{Vars}(\varphi[\theta])).$$

By $\varphi[\theta]$ we mean the result of the substitution θ on φ. $\mathrm{NotConstants}(X)$ is the conjunction of inequalities of the form $x \neq c$, for each $x \in X$ and $c \in \mathrm{dom}(\mathcal{A}_{\mathrm{CONST}})$. The inequalities are introduced to ensure that for any database $\mathcal{A}$ such that $\mathcal{A} \models \Gamma_t$, in any tuple satisfying $\varphi[\theta]$ no variable from $\mathrm{Vars}(\varphi[\theta])$ is substituted with an element from $\mathcal{A}_{\mathrm{CONST}}$.

The numbers k_θ are guessed in such a way that $\sum_\theta k_\theta = k + \alpha * j$, for some $\alpha \in \mathbb{N}$. It is enough to consider the numbers bounded by $k + j$. Intuitively numbers k_θ determine how the total number of tuples satisfying the constraint $t = (\varphi, k, j)$ is distributed among its versions $\varphi[\theta]$, for all θ.

We define $\Gamma_1 = \mathrm{REPLACE}(\bigcup_{t \in \Gamma_0} \Gamma_t)$.

Lemma 3.

1. *Let $\mathcal{A}$ be a database, such that $\mathcal{A}_{\text{CONST}} \subseteq \mathcal{A}$ and $\mathcal{A} \models \Gamma$. There exists a choice of the numbers k_θ in Γ_1 such that $\mathcal{A} \models \Gamma_1$.*
2. *If there exists $\mathcal{A}' \supseteq \mathcal{A}_{\text{CONST}}$ such that $\mathcal{A}' \models \Gamma_{\text{CONST}}$ and $\mathcal{A}' \models \Gamma_1$ then $\mathcal{A}' \models \Gamma$.*
3. *The size of Γ_1 is exponential w.r.t. the size of Γ.*

Step 2.

Definition 4. *Let $\mathcal{B}$ be a relational structure over the vocabulary containing relational symbols from Γ and constant symbols* $\text{dom}(\mathcal{A}_{\text{CONST}})$. *Define GRAPH($\mathcal{B}$) to be a graph, whose vertices are the elements of $\mathcal{B}$ which are not a constant in* $\text{dom}(\mathcal{A}_{\text{CONST}})$. *There is an edge between vertices e_1, e_2 of GRAPH($\mathcal{B}$) if there is a tuple $\bar{e}$ of elements of $\mathcal{B}$ containing both e_1 and e_2, such that an atom $R(\bar{e})$ is true in $\mathcal{B}$, for some relation R in $\mathcal{B}$.*

Consider a formula $\varphi(\bar{x})$ from one of the constraints from Γ_1. The formula $\varphi(\bar{x})$ is of the form $\bigwedge_k R_k(\bar{x}_k) \wedge \text{NotConstants}(\bar{x})$.

Notice that the set of vertices of $\text{GRAPH}(\mathcal{C}_\varphi)$ (i.e. the graph for the canonical structure for φ) is exactly the set of variables of φ.

Definition 5. A connected subformula *of φ is a formula $\varphi_D(\bar{x}_D)$ defined as $\bigwedge_{R \in D} R(\bar{x}_R) \wedge \text{NotConstants}(\bar{x}_D)$, where D is a maximal set of non-ground atoms (i.e. atoms with variables), such that GRAPH($\mathcal{C}_{\varphi_D}$) is a connected component of GRAPH($\mathcal{C}_\varphi$).*

Notice, that the formula φ is a conjunction of its connected subformulas and its ground atoms. (i.e. atoms without variables).

Example 2. Consider following formula $\varphi(x_1, \ldots, x_5)$:

$$R_1(x_1, x_2) \wedge R_2(x_2, x_3, c_1, x_5, c_2) \wedge R_1(c_1, c_3) \wedge R_1(c_3, c_3)$$

$$\wedge R_1(x_4, c_1) \wedge R_1(c_2, x_4) \wedge \text{NotConstants}(\{x_1, \ldots, x_5\}),$$

where c_1, c_2, c_3 are constants from $\mathcal{A}_{\text{CONST}}$. Vertices of $\text{GRAPH}(\mathcal{C}_\varphi)$ are $\{x_1, \ldots, x_5\}$, and edges of $\text{GRAPH}(\mathcal{C}_\varphi)$ are $\{x_1, x_2\}$, $\{x_2, x_3\}$, $\{x_2, x_5\}$ and $\{x_3, x_5\}$. Clearly, $\text{GRAPH}(\mathcal{C}_\varphi)$ has two connected components, namely $\{x_1, x_2, x_3, x_5\}$ and $\{x_4\}$.

There are two connected subformulas of φ:

$$\varphi_1(x_1, x_2, x_3, x_5) = R_1(x_1, x_2) \wedge R_2(x_2, x_3, c_1, x_5, c_2) \wedge \text{NotConstants}(\{x_1, x_2, x_3, x_5\})$$

and

$$\varphi_2(x_4) = R_1(x_4, c_1) \wedge R_1(c_2, x_4) \wedge \text{NotConstants}(\{x_4\}).$$

Ground atoms of φ are $R_1(c_1, c_3)$ and $R_1(c_3, c_3)$.

The motivation for Step 2 can be best explained using the following example. Again, recall that our goal is to order the proper constraints in such a way that extending $\mathcal{A}_{\text{CONST}}$ with the canonical structure of the formula in the constraint does not increase the number of tuples satisfying the earlier (in the order) constraints.

Example 3. Let $\varphi_1 = R_1(c_1, x_1) \wedge R_2(c_1, x_2) \wedge \text{NotConstants}(\{\bar{x}\})$ and $\varphi_2 = R_1(c_1, x_1) \wedge R_2(c_1, x_2) \wedge R_3(c_1, x_3) \wedge \text{NotConstants}(\{\bar{x}\})$. Thus φ_1 consists of two and φ_2 consists of three connected subformulas. Let t_1 be a constraint containing the formula φ_1 and t_2 a constraint containing the formula φ_2. Clearly, it is impossible to order t_1 and t_2 in a right way –adding a copy of the canonical structure $\mathcal{C}_{\varphi_2}$ changes the number of tuples satisfying φ_1, and, in presence of at least one copy of $\mathcal{C}_{\varphi_2}$, adding a copy of $\mathcal{C}_{\varphi_1}$ changes the number of tuples satisfying φ_2.

Step 2 is performed in order to avoid the problem from Example 3. The step consists of replacing each constraint t in Γ_1 with separate constraints for each connected subformula of the formula in t. We also forget about ground atoms in formulas in the constraints. The reason is that after Step 1 the ground atoms are already determined to be either true or false.

Let $t = (\varphi, k, j)$ be a constraint in Γ_1, let $\varphi_1, \ldots, \varphi_n$ be connected subformulas of φ and let $\psi_1, \ldots, \psi_l$ be ground atoms of φ. Let $m_i \in \{0, 1\}$ be 1 if ψ_i holds in $\mathcal{A}_{\text{CONST}}$ and 0 otherwise (for $i = 1, \ldots, l$). Notice that the number of tuples satisfying t is the product of the numbers of tuples satisfying φ_i, (for $i = 1 \ldots, n$), times the product of m_i, for $i = 1, \ldots, l$.

Now, if $k > 0$ and some $m_i = 0$, we know that the constraints are inconsistent, so the algorithm can stop. If $k = 0$ and some $m_i = 0$ the constraint is satisfied in every superstructure of $\mathcal{A}_{\text{CONST}}$ so we put $\Gamma_t = \emptyset$. If $k \geq 0$ and for all $i = 1, \ldots, l$ the number $m_i = 1$, then Γ_t consists of triples (φ_i, k_i, j), where the numbers $k_i \leq k + j$ are guessed such that $\prod_i k_i = k + \alpha * j$, for some $\alpha \in \mathbb{N}$.

We define $\Gamma_2 = \text{REPLACE}(\bigcup_{t \in \Gamma_1} \Gamma_t)$.

Lemma 4. *1. Let $\mathcal{A}$ be a database, such that $\mathcal{A} \supseteq \mathcal{A}_{\text{CONST}}$ and $\mathcal{A} \models \Gamma_{\text{CONST}} \cup \Gamma_1$. There exists a choice of the numbers k_i in Γ_2, such that $\mathcal{A} \models \Gamma_2$.*

2. For every database $\mathcal{A}'$ such that $\mathcal{A}' \supseteq \mathcal{A}_{\text{CONST}}$ if $\mathcal{A}' \models \Gamma_{\text{CONST}}$ and $\mathcal{A}' \models \Gamma_2$ then $\mathcal{A}' \models \Gamma$.

3. The size of Γ_2 is exponential w.r.t. the size of Γ.

Step 3. Consider a database $\mathcal{A}$, a formula $\varphi(\bar{x})$ from Γ_2 and a tuple $\bar{a}$ such that $\mathcal{A} \models \varphi(\bar{a})$. The substitution of elements $\bar{a}$ for variables $\bar{x}$ may map several variables from $\bar{x}$ to a single element a in $\bar{a}$. During Step 3 we replace each constraint t with separate constraints for all possible ways in which variables of the formula of t can be identified. We also disallow any further identification of variables in the resulting constraints Γ_3. In other words: if $\mathcal{A} \models \varphi(\bar{a})$ then there is a corresponding homomorphism from elements of the canonical structure $\mathcal{C}_\varphi$ to elements of $\mathcal{A}$. The goal of this step is to obtain a new set Γ_3 which can replace Γ_2, such that all homomorphisms, which correspond to the tuples satisfying formulas in Γ_3, are injective.

The following example explains why we need this step.

Example 4. Let $\varphi_1 = R(x_1, x_2) \wedge R(x_1, x_3)$ and $\varphi_2 = R(x_1, x_2)$. The formula φ_2 is, in fact, equal to the formula φ_1 with variables x_2 and x_3 identified. Let t_1 be a constraint containing the formula φ_1 and t_2 a constraint containing the formula φ_2. Similarly as in Example 3, these two constraints cannot be ordered properly. If we extend a database with the canonical structure $\mathcal{C}_{\varphi_1}$ we change the number of tuples satisfying φ_2 and vice versa.

Let $t = (\varphi, k, j)$ be a constraint in Γ_2. We define Γ_t to be the set of constraints of the form: $(\psi_\theta, k_\theta, j)$ for each $V \subseteq \mathrm{Vars}(\varphi)$ and each $\theta\colon \mathrm{Vars}(\varphi)\setminus V \to V$, where ψ_θ is:

$$\varphi[\theta] \wedge \mathrm{INEQ}(\mathrm{Vars}(\varphi[\theta])).$$

The numbers $k_\theta \leq k + j$ are guessed such that $\sum_\theta k_\theta = k + \alpha * j$, for some $\alpha \in \mathbb{N}$. $\mathrm{INEQ}(\mathrm{Vars}(\varphi[\theta]))$ is a conjunction of inequalities of the form $x \neq y$, for each pair of distinct variables $x, y \in \mathrm{Vars}(\varphi[\theta])$. We introduce the inequalities to ensure that all variables which are not identified during this step have to be substituted with distinct elements of a database.

Finally, we define Γ_3 as $\mathrm{REPLACE}(\bigcup_{t\in\Gamma_2} \Gamma_t)$.

The following lemma states the properties of Γ_3.

Lemma 5.

1. *Let $\mathcal{A}$ be a database, such that $\mathcal{A} \supseteq \mathcal{A}_{\mathrm{CONST}}$ and $\mathcal{A} \models \Gamma_{\mathrm{CONST}} \cup \Gamma_2$ There exists a choice of the numbers k_θ in Γ_3, such that $\mathcal{A} \models \Gamma_3$.*
2. *For every database $\mathcal{A}'$ such that $\mathcal{A}' \supseteq \mathcal{A}_{\mathrm{CONST}}$ if $\mathcal{A}' \models \Gamma_{\mathrm{CONST}}$ and $\mathcal{A}' \models \Gamma_3$ then $\mathcal{A}' \models \Gamma$.*
3. *The size of Γ_3 is exponential w.r.t. Γ.*

4.2 Construction of an exponential database satisfying Γ_3.

In Sect. 4.1 we constructed, for a set Γ of modulo constraints a set Γ_3 of constraints which are satisfiable if and only if Γ are, and such that formulas in the constraints from Γ_3 have the following normal form:

(A) Each formula contains the NotConstants subformula, so that the variables cannot be substituted with elements $\mathrm{dom}(\mathcal{A}_{\mathrm{CONST}})$;
(B) each formula is connected and does not contain ground atoms;
(C) each formula contains the INEQ subformula, so that distinct variables cannot be substituted with the same element of a database.

Define a partial order $\leq_{part}$ on constraints from Γ_3 as follows: $(\varphi_1, k_1, j_1) \leq_{part} (\varphi_2, k_2, j_2)$ if there exists a tuple $\bar{a}$ of elements of $\mathcal{C}_{\varphi_2}$ such that $\mathcal{C}_{\varphi_2} \models \varphi_1(\bar{a})$. In other words: $(\varphi_1, k_1, j_1) \leq_{part} (\varphi_2, k_2, j_2)$ if $\mathcal{C}_{\varphi_1}$ is isomorphic to a substructure of $\mathcal{C}_{\varphi_2}$. We use the word *substructure* in a *positive* sense where $R(a, b)$ is a substructure of $R(a, b), R(b, a), R(a, c)$.

Let $\leq$ be some linear order on constraints from Γ_3 consistent with the partial order $\leq_{part}$. We write $(\varphi_1, k_1, j_1) < (\varphi_2, k_2, j_2)$ if $(\varphi_1, k_1, j_1) \leq (\varphi_2, k_2, j_2)$ and $\varphi_1 \neq \varphi_2$ (recall that formulas in Γ_3 are unique).

Lemma 6. *Let (φ, k, j) be a constraint in Γ_3 such that $\mathcal{C}_\varphi$ has exactly n automorphisms. There are exactly n tuples $\bar{a}$ such that $\mathcal{C}_\varphi \models \varphi(\bar{a})$. Moreover, for every database $\mathcal{B}$ we have $|\{\bar{a} \mid \mathcal{B} \models \varphi(\bar{a})\}| = \alpha * n$, for some $\alpha \in \mathbb{N}$.*

Notice however, that the above lemma would not be true if conjunctive queries with projections were allowed.

Lemma 7. *Let (φ, k, j) be a constraint in Γ_3 and let $\mathcal{B}$ be a database such that $\mathcal{A}_{\text{CONST}} \subseteq \mathcal{B}$ and $\mathcal{B} \models \{t \in \Gamma_3 \mid t > (\varphi, k, j)\}$. If the constraints Γ_3 are satisfiable then there exists a database $\mathcal{B}' \supseteq \mathcal{B}$ such that $\mathcal{B}' \models \{t \in \Gamma_3 \mid t \geq (\varphi, k, j)\}$. The size of the database $\mathcal{B}'$ is at most $|\mathcal{B}| + |\mathcal{C}_\varphi| * \delta$, where $\delta \in \mathbb{N}$ is bounded by the sum of k and j.*

Example 5. Consider constraints $t_1 = (\varphi_1, 2, 2)$ and $t_2 = (\varphi_2, 1, 6)$. Let $\varphi_1 = R(v, z_1) \wedge R(v, z_2) \wedge R'(a, b, z_1) \wedge \text{NotConstants}(v, z_1, z_2) \wedge \text{INEQ}(v, z_1, z_2)$ and $\varphi_2 = R(x, y) \wedge \text{NotConstants}(x, y) \wedge \text{INEQ}(x, y)$. Clearly: $t_1 > t_2$. There are two tuples $\bar{x}$ such that $C_{\varphi_1} \models \varphi_2(\bar{x})$. Let $\mathcal{B}$ be a database presented schematically at Fig. 2 such that $\mathcal{B} \models \{t_1\}$. Our algorithm constructs the database $\mathcal{B}' \models \{t_1, t_2\}$. In order to satisfy t_2 three copies of C_{φ_2} are added.

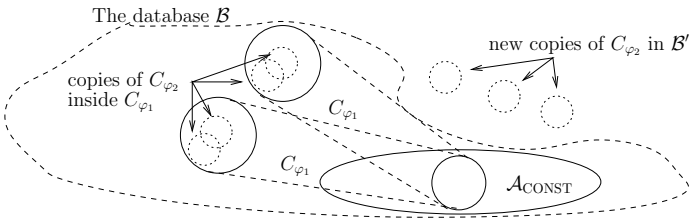


Fig. 2. The database $\mathcal{B}$ satisfying the constraint t_1 from Example 5.

Proof (of Lemma 7). Let n be the number of automorphisms of $\mathcal{C}_\varphi$. According to Lemma 6, the number $m = |\{\bar{b} \in \mathcal{B} \mid \mathcal{B} \models \varphi(\bar{b})\}|$ is a multiple of n.

Let m' be the smallest number such that m' is a multiple of n (including 0) and $m' + m = k + \alpha * j$, for some $\alpha \in \mathbb{N}$. If constraints Γ_3 are satisfiable such number m' exists and its value is bounded by $j * n + k$.

Let $x_1, \ldots, x_{|\text{Vars}(\varphi)|}$ be variables in φ. The database $\mathcal{B}'$ is defined as the union of the database $\mathcal{B}$ and $\frac{m'}{n}$ copies of the canonical structure $\mathcal{C}_\varphi$, with constants $\text{dom}(\mathcal{A}_{\text{CONST}})$ from each copy of $\mathcal{C}_\varphi$ identified with elements of $\mathcal{A}_{\text{CONST}} \subseteq \mathcal{B}$. Formally, elements of $\mathcal{B}'$ are elements of $\mathcal{B}$ and new elements $e_{h,i}$, for $h = 1, \ldots, |\text{Vars}(\varphi)|$ and $i = 1, \ldots, \frac{m'}{n}$. Database $\mathcal{B}'$ is a superstructure of $\mathcal{B}$, and additionally for each conjunct $R(w_1, \ldots, w_l)$ in φ, for each $i = 1, \ldots, \frac{m'}{n}$, the atom $R(v_{1,i}, \ldots, v_{l,i})$ is true in $\mathcal{B}'$, where

$$v_{h,i} = \begin{cases} e_{g,i} \text{ if } w_h = x_g, \text{ where } g \in \{1, \ldots, |\text{Vars}(\varphi)|\} \\ a \quad \text{if } w_h = a, \text{ where } a \text{ is a constant,} \end{cases}$$

for $h = 1, \ldots, l$. Clearly, the size of $\mathcal{B}'$ is at most $|\mathcal{B}| + |\mathcal{C}_\varphi| * \frac{m'}{n}$.

Now, we show that $\mathcal{B}' \models \{t \in \Gamma_3 \mid t \geq (\varphi, k, j)\}$. We will use the following observation:

Observation. Consider a constraint $(\varphi', k', j') \in \Gamma_3$, such that $(\varphi', k', j') \geq (\varphi, k, j)$. Each tuple $\bar{b}$ such that $\mathcal{B}' \models \varphi'(\bar{b})$ is contained in a single connected component of GRAPH($\mathcal{B}'$).

Proof (of the observation). This is since, by (A), the variables in φ' cannot be substituted with elements from $\mathcal{A}_{\text{CONST}}$. and since, by (B), the graph of the canonical structure for φ' consists of a single connected component. □

We show that $\mathcal{B}' \models (\varphi, k, j)$. Let us count the number of tuples $\bar{b}$ such that $\mathcal{B}' \models \varphi(\bar{b})$: there are m tuples consisting of elements of $\mathcal{B}$ and m' new tuples (by Lemma 6), such that the elements of each of them are all contained in some new copy of $\mathcal{C}_\varphi$. Since newly added copies of $\mathcal{C}_\varphi$ are the only new connected components of GRAPH($\mathcal{B}'$), it follows from the above observation that there are no new tuples satisfying φ in $\mathcal{B}'$. So there are exactly $m + m'$ tuples satisfying φ and $\mathcal{B}' \models (\varphi, k, j)$.

Now we need to prove that by extending the structure we did not spoil one of the old constraints. We claim that, for each constraint $(\varphi', k', j') \in \Gamma_3$, such that $(\varphi', k', j') > (\varphi, k, j)$, the number of tuples satisfying (φ', k', j') in $\mathcal{B}'$ is exactly the same as in $\mathcal{B}$: from the above observation it follows that each new tuple $\bar{b}$, such that $\mathcal{B}' \models \varphi'(\bar{b})$, must be contained in some copy of $\mathcal{C}_\varphi$. But this would mean that $\mathcal{C}_\varphi \models \varphi'(\bar{b})$, which would contradict $(\varphi', k', j') > (\varphi, k, j)$. □

Let us now construct a sequence of databases $\mathcal{A}_i$, for $i = 0, \ldots, |\Gamma_3|$, such that the database $\mathcal{A}_i$ satisfies the set of first i (in the order $\leq$) constraints from Γ_3. We start from the database $\mathcal{A}_0 = \mathcal{A}_{\text{CONST}}$. Then, for all $i = 1, \ldots, |\Gamma_3|$, we consider the i-th (in the order $\leq$) constraint from Γ_3 and obtain the database $\mathcal{A}_i$ from the database $\mathcal{A}_{i-1}$ using Lemma 7, which guarantees that finally: $\mathcal{A}' = \mathcal{A}_{|\Gamma_3|}$ satisfies Γ_3, and the size of $\mathcal{A}'$ is at most exponential in $|\Gamma|$.

So far, our nondeterministic algorithm has built a database $\mathcal{A}'$. As its last step it just verifies if $\mathcal{A}' \models \Gamma_3 \cup \Gamma_{\text{CONST}}$. This would almost finish the proof of Theorem 2. The only thing which would still be in doubt is if in the process of satisfying the constraints from Γ_3 we did not spoil anything concerning the constant constraints Γ_{CONST}:

Lemma 8. *Let $\mathcal{A}'$ be the database resulting from the construction in the previous paragraphs. If there exists a database $\mathcal{A}$ such that $\mathcal{A} \supseteq \mathcal{A}_{\text{CONST}}$ and $\mathcal{A} \models \Gamma_{\text{CONST}} \cup \Gamma_3$ then $\mathcal{A}' \models \Gamma_{\text{CONST}}$.*

Notice that Lemma 8 needs an additional assumption: that there exists $\mathcal{A} \supseteq \mathcal{A}_{\text{CONST}}$ satisfying all the constraints $\Gamma_3 \cup \Gamma_{\text{CONST}}$. Otherwise it might happen that new tuples satisfying the queries from Γ_{CONST} would appear in $\mathcal{A}'$, and thus the constraints from Γ_{CONST} would be violated in $\mathcal{A}'$. But if the constraints are satisfiable then our nondeterministic algorithm guessed $\mathcal{A}_{\text{CONST}}$ correctly, and so we can be sure that such $\mathcal{A} \supseteq \mathcal{A}_{\text{CONST}}$ indeed exists.

Observation. Let $(\varphi, k, j) \in \Gamma_3$. If $\mathcal{A}_{\text{CONST}} \cup \mathcal{C}_\varphi$ is a substructure of $\mathcal{A}'$ then $\mathcal{A}_{\text{CONST}} \cup \mathcal{C}_\varphi$ is also a substructure of $\mathcal{A}$. By $\mathcal{A}_{\text{CONST}} \cup \mathcal{C}_\varphi$ we mean here a union of the two structures, with constants from $\mathcal{C}_\varphi$ identified with the respective elements in $\mathcal{A}_{\text{CONST}}$. Again, the word *substructure* is used in a *positive* sense.

Proof (of the observation). Suppose $\mathcal{A}_{\text{CONST}} \cup \mathcal{C}_\varphi$ is a substructure of $\mathcal{A}'$. Then there exists a minimal number i such that $0 < i \leq |\Gamma_3|$ and $\mathcal{A}_{\text{CONST}} \cup \mathcal{C}_\varphi$ is a substructure of $\mathcal{A}_i$.Thus, at the i-th step of the construction of $\mathcal{A}'$, while processing some constraint $(\varphi', k', j') \in \Gamma_3$ we extended the database $\mathcal{A}_{i-1}$ with at least one copy of the canonical structure $\mathcal{C}_\varphi$. This means that $\mathcal{C}_\varphi$ must be a substructure of $\mathcal{C}_{\varphi'}$. Now there are 2 cases:

Case 1: $k' > 0$**.** Then $\mathcal{A}_{\text{CONST}} \cup \mathcal{C}_{\varphi'}$ is a substructure of $\mathcal{A}$, and so also $\mathcal{A}_{\text{CONST}} \cup \mathcal{C}_\varphi$ is a substructure of $\mathcal{A}$.

Case 2: $k' = 0$**.** Recall that we extended $\mathcal{A}_{i-1}$ because the number of tuples $\bar{b}$ such that $\mathcal{A}_{i-1} \models \varphi'(\bar{b})$ was not equal to $\alpha * j'$ for any $\alpha \in \mathbb{N}$, including $\alpha = 0$. Hence, $\mathcal{A}_{\text{CONST}} \cup \mathcal{C}_{\varphi'}$ is a substructure of A_{i-1}, but therefore $\mathcal{A}_{\text{CONST}} \cup \mathcal{C}_\varphi$ is a substructure of A_{i-1}. But this contradicts the minimality of i. □

Proof (of Lemma 9). For each $\mathcal{B}'$ being a substructure of $\mathcal{A}'$, of a form $\mathcal{A}_{\text{CONST}} \cup \mathcal{C}_\varphi$ (where φ is a formula in constraints from Γ_3) and such that GRAPH($\mathcal{B}'$) is a connected component of GRAPH($\mathcal{A}'$), fix one substructure $\mathcal{B}$ of $\mathcal{A}$ of the form $\mathcal{A}_{\text{CONST}} \cup \mathcal{C}_\varphi$ (existence of $\mathcal{B}$ is guaranteed by Observation) and define $h_{\mathcal{B}} \colon \mathcal{B}' \to \mathcal{B}$ as identity (or, to be more precise, as isomorphism).

We define a mapping $h \colon \mathcal{A}' \to \mathcal{A}$. For $c \in \mathcal{A}_{\text{CONST}}$ put $h(c) = c$. For $a \notin \mathcal{A}_{\text{CONST}}$ put $h(a) = h_{\mathcal{B}}(a)$, where $\mathcal{B}$ is such that $a \in \text{dom}(h_{\mathcal{B}})$. Notice that we used the fact that all connected components of GRAPH($\mathcal{A}'$) are of the form GRAPH($\mathcal{C}_\varphi$) for some $(\varphi, k, j) \in \Gamma_3$.

Now, suppose that $\mathcal{A}' \not\models \Gamma_{\text{CONST}}$. So, there exists a tuple $\bar{a}$ of elements of $\mathcal{A}'$, containing some element(s) not in $\mathcal{A}_{\text{CONST}}$ such that for a constraint $(\psi, k, j) \in \Gamma_{\text{CONST}}$ it holds $\mathcal{A}' \models \psi(\bar{a})$. But then the tuple $h(\bar{a})$ contains some element(s) not in $\mathcal{A}_{\text{CONST}}$ and $\mathcal{A} \models \psi(h(\bar{a}))$. (this is since negation and inequality are not allowed in constraints from Γ_{CONST}). The last implies that $\mathcal{A} \not\models \Gamma_{\text{CONST}}$. □

5 From the intermediate result to the main result

In this section we use some of the ideas from [2] to show how Theorem 2 implies Theorem 1. We start with the following definition and lemma from [2].

Definition 6. *Let R be a TreeQL-program and let d be a DTD such that R does not typecheck with respect to d. Then:*

- *there is a path $\bar{v} = v_1, \ldots, v_k$ in the program R where*
  1. *v_1 is a child of the root;*
  2. *$lab(v_i) = (\sigma_i, \varphi_i(\bar{x}_1, \ldots, \bar{x}_i))$, for $i \in \{1, \ldots, k\}$;*
  3. *let $\bar{x} = \bar{x}_1, \ldots, \bar{x}_k$.*
     The node v_k has precisely n children with labels $(\delta_1, \psi_1(\bar{x}, \bar{y}_1))$, …, $(\delta_n, \psi_n(\bar{x}, \bar{y}_n))$;

and

– *there is a database* $\mathcal{A}$ *with elements* $\bar{a} = \bar{a}_1, \ldots, \bar{a}_k$ *such that:*
  1. $\mathcal{A}$ *satisfies* $\varphi_i(\bar{a}_1, \ldots, \bar{a}_i)$*, for each* $i = 1, \ldots, k$*;*
  2. $\delta_1^{j_1} \ldots \delta_n^{j_n} \notin d(\sigma_k)$ *where* $j_i = |\{\bar{b} \,|\, \mathcal{A} \text{ satisfies } \psi_i(\bar{a}, \bar{b})\}|$*, for all* $i = 1, \ldots, n$*.*

We say that $(\bar{v}, \mathcal{A}, \bar{a})$ *is a* breakpoint *for* R *and* d*.*

Lemma 9 ([2]). *Let* $\delta_1, \ldots, \delta_n$ *be symbols and let* $\nu = (k_1, j_1), \ldots, (k_1, j_1)$ *be a vector of* n *pairs of natural numbers. We denote by* L_ν *the language of all words of the form:* $\delta_1^{k_1+\alpha_1*j_1} \ldots \delta_n^{k_n+\alpha_n*j_n}$ *where each* $\alpha_i \in \mathbb{N}$*,* $1 \leq i \leq n$*. For each regular language* r *over alphabet* $\{\delta_1, \ldots, \delta_n\}$*, there exists a finite set Vec(r) of vectors of pairs of natural numbers as above such that* $\neg r \cap \delta_1^* \ldots \delta_n^* = \bigcup_{\nu \in \mathrm{Vec}(r)} L_\nu$*.*

Moreover, the values of the numbers in Vec(r) *are bounded by the number of states of the deterministic automaton recognizing* $\neg r$*.*

We briefly sketch the beginning of the proof of the non-elementary upper bound from [2], using the notation introduced in Definition 6. Assume that the program R does not typecheck w.r.t. d, then there exists a breakpoint $(\bar{v}, \mathcal{A}, \bar{a})$. Let r be the regular language $d(\sigma_k)$ specified by the DTD d. Consider the language $\neg r \cap \delta_1^* \ldots \delta_n^*$, it is the intersection of the language of children of the node v_k (i.e. $\delta_1^* \cdot \ldots \cdot \delta_n^*$) and the complement of r. From lemma 9 it follows that there exists a set of vectors $\mathrm{Vec}_{R,d,\bar{v}}$ such that $\neg r \cap \delta_1^* \ldots \delta_n^* = \bigcup_{\nu \in \mathrm{Vec}_{R,d,\bar{v}}} L_\nu$

Since $(\bar{v}, \mathcal{A}, \bar{a})$ is a breakpoint then there exists a vector $\nu = (k_1, j_1), \ldots, (k_n, j_n)$ in $\mathrm{Vec}_{R,d,\bar{v}}$, such that for each $l \in \{1, \ldots, n\}$ there exists $\alpha_l \in \mathbb{N}$ such that: $|\{\bar{b} \mid \mathcal{A} \models \psi_l(\bar{a}, \bar{b})\}| = k_l + \alpha_l * j_l$.

Then it is shown that it is always possible to find a substructure $\mathcal{A}'$ of $\mathcal{A}$ of a size bounded independently of $\mathcal{A}$ such that elements $\bar{a}$ are in $\mathcal{A}'$ and for each $l \in \{1, \ldots, n\}$ there exists $\alpha'_l \in \mathbb{N}$ such that: $|\{\bar{b} \mid \mathcal{A}' \models \psi_l(\bar{a}, \bar{b})\}| = k_l + \alpha'_l * j_l$.

In our proof we do not require the database $\mathcal{A}'$ to be a substructure of $\mathcal{A}$. This allows us to modify the structure, which makes it possible to achieve an exponential upper bound, however at the cost that we can no longer have universal formulas as integrity constraints.

Again, we use the same notation as in Definition 6. We begin by guessing a path $\bar{v} = v_1, \ldots, v_k$ in the program R, and a vector $\nu = (k_1, j_1), \ldots, (k_n, j_n)$ of as many pairs of natural numbers as the node v_k has children. The numbers in ν are bounded by the number of states of the DFA for $\neg r$. Hence, the sizes of binary representations of the numbers in ν are linear in DTD d. Then we have to check[2] whether the language $\{\delta_1^{k_1+\alpha_1*j_1} \ldots \delta_n^{k_n+\alpha_n*j_n} \mid \alpha_1, \ldots, \alpha_n \in \mathbb{N}\}$ is contained in $\neg r$. This step can be done in PSPACE. It is enough to check words with values of $\alpha_1, \ldots, \alpha_n$ bounded by the number of states of the DFA for $\neg r$. Each of these words (represented with all numbers written in binary) can

[2] This is to guarantee that if for some database $\mathcal{A}$ there exists a node (v_k, θ) in the output tree $R(\mathcal{A})$ such that the concatenation of symbols labeling children of (v_k, θ) is in the language defined by ν then $R(\mathcal{A})$ is not consistent with the DTD d.

be verified to be in $\neg r$ in PSPACE by checking whether a final state of the NFA for r can be reached by reading the word.

Now, we construct a set of modulo constraints Γ over the vocabulary consisting of relational symbols in the program R and constants $\bar{a} = \bar{a}_1, \ldots, \bar{a}_k$:

1. For formulas $\varphi_1, \ldots, \varphi_k$ in nodes $v_1, \ldots, v_k$ we define the constraint t_0: $(\bigwedge_{i=1,\ldots,k} \varphi_i(\bar{a}_1, \ldots, \bar{a}_i), 1, 0)$. Notice that $\mathcal{A} \models t_0$ iff $\mathcal{A}$ satisfies $\varphi_i(\bar{a}_1, \ldots, \bar{a}_i)$, for each $i = 1, \ldots, k$.
2. For the l-th child ($l = 1, \ldots, n$) of the node v_k we define the constraint $t_l = (\psi_l(\bar{a}, \bar{y}_l), k_l, j_l)$. Notice that $\mathcal{A} \models \bigcup_{l=1,\ldots,n}\{t_l\}$ iff $\delta_1^{j_1} \ldots \delta_n^{j_n}$ is in the language defined by ν, where j_l is $|\{\bar{b} \,|\, \mathcal{A} \models \psi_l(\bar{a}, \bar{b})\}|$, for all $l = 1, \ldots, n$.

Finally, Γ is defined as $\{t_0, \ldots, t_n\}$. We conclude with the following lemma, which completes the proof of Theorem 1.

Lemma 10.

1. *If the program R does not typecheck w.r.t. d then there exists a choice of a path $\bar{v}$ in R and a choice of a vector ν, such that there exists a database $\mathcal{A}$ satisfying Γ.*
2. *For every choice of a path $\bar{v}$ in R and every choice of a vector ν, if there exists a database $\mathcal{A}$ satisfying Γ then R does not typecheck w.r.t. d.*
3. *The construction of Γ can be done in polynomial space.*

Acknowledgments. This paper would not have been possible without help of Jurek Marcinkowski, who spent a lot of time on discussions with me and suggested many ideas. Then I would like to thank Tomek Truderung for his suggestions and comments. I really appreciate all the help from both of them. Also, I thank the anonymous referees for their helpful comments.

References

1. S.Abiteboul, R. Hull, V. Vianu Foundations of Databases. Addison-Wesley 1995.
2. N. Alon, T. Milo, F. Neven, D. Suciu, V. Vianu Typechecking XML Views of Relational Databases, *ACM Transactions on Computational Logic*, Vol. 4, No. 3, July 2003, pages 315-354. (preliminary version in *Proceedings of the 16th LICS*, 421-430, 2001).
3. N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. XML with data values: Typechecking revisited. *Journal of Computer and System Sciences*, 66(4) pages 688-727, 2003.
4. M. Fernandez, D. Suciu, W. Tan. SilkRoute: Trading between relations and XML. In *Proceedings of the WWW9 Conference*. 2000, pages 723-746.
5. W. Martens and F. Neven. On the complexity of typechecking top-down XML transformations. *Theoretical Computer Science, 336(1)* pages 153-180, 2005.
6. W. Martens and F. Neven. Frontiers of tractability for typechecking simple XML transformations. *Journal of Computer and System Sciences, 2006*. to appear.
7. T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. *Journal of Computer and System Sciences*, 66(1) pages 66-97, 2003.