# Solutions in XML Data Exchange

Mikołaj Bojańczyk
University of Warsaw
bojan@mimuw.edu.pl

Leszek A. Kołodziejczyk
University of Warsaw
lak@mimuw.edu.pl

Filip Murlak
University of Warsaw
fmurlak@mimuw.edu.pl

## ABSTRACT

The task of XML data exchange is to restructure a document conforming to a source schema under a target schema according to certain mapping rules. The rules are typically expressed as source-to-target dependencies using various kinds of patterns, involving horizontal and vertical navigation, as well as data comparisons. The target schema imposes complex conditions on the structure of solutions, possibly inconsistent with the mapping rules. In consequence, for some source documents there may be no solutions.

We investigate three problems: deciding if all documents of the source schema can be mapped to a document of the target schema (absolute consistency), deciding if a given document of the source schema can be mapped (solution existence), and constructing a solution for a given source document (solution building).

We show that the complexity of absolute consistency is rather high in general, but within the polynomial hierarchy for bounded depth schemas. The combined complexity of solution existence and solution building behaves similarly, but the data complexity turns out to be very low.

In addition to this we show that even for much more expressive mapping rules, based on MSO definable queries, absolute consistency is decidable and data complexity of solution existence is polynomial.

## Categories and Subject Descriptors

H.2.5 [**Database Management**]: Heterogeneous Databases—*Data translation*; I.7.2 [**Document and Text Processing**]: Document Preparation—*XML*

## General Terms

Theory, Algorithms, Languages

## Keywords

XML data exchange, regular queries, patterns, absolute consistency, solution building, solution existence


*ICDT 2011*, March 21–23, 2011, Uppsala, Sweden.


## 1. INTRODUCTION

One of the main challenges of modern data management is dealing with heterogeneous data. A typical scenario is that of data exchange, where one needs to restructure data stored in a source database under a target database schema, following a specification. The specification is given by a so-called *schema mapping*, a collection of logical formulas describing dependencies between the source schema and the target schema. The produced instance of data is called a *solution* for the source data with respect to the schema mapping.

Studies on relational data exchange and schema mappings were initiated several years ago [11, 12], and since then have been a major topic (see recent surveys [5, 6, 15]).

In the XML context, the target schema, a DTD or XML Schema, imposes complex conditions on the structure of the solution. One of the consequences is that in practice schema mappings are often *over-specified*, in the sense that for some source instances there is no solution satisfying the specification and conforming to the target schema. One needs to be able to check if a mapping admits a solution or not.

Nowadays, schema mapping design is usually assisted by specialized software [16]. The ability to issue warnings that a mapping does not admit solutions for some source documents would be a most welcome feature, especially if the warning could also point out the source of the problem.

When a mapping is already there, one has to materialize the appropriate solution for the given source data. This requires checking if a solution exists, and constructing it.

We concentrate on three problems related to solutions:

- *Absolute consistency.* Is there a solution for every possible source document?
- *Solution existence.* Is there a solution for a given source document?
- *Solution building.* Find a solution for a given source document.

So far, absolute consistency was only investigated for very simple mappings, based on so-called tree patterns using only the child and descendant relations [1]. Solution building was investigated for even simpler mappings allowing only a restricted class of DTDs [3]. More practical mapping languages involve sibling order and data comparisons. For such mappings only the *consistency* problem was considered, i.e., whether *some* source document has a solution [1, 3]. Consistency of mappings, though interesting theoretically, seems to have less practical importance as it gives no information on how the mapping will perform on a given source document.

We work with mappings using general XML schemas, formalized as tree automata, DAG-shaped patterns using horizontal and vertical navigation, as well as data comparisons. We also consider a restricted case: *bounded-depth* mappings, where schemas only admit trees of bounded height. As most real-life schemas have small depth, this case is particularly important in practice.

Our main findings for pattern-based mappings can be summarized as follows:

- Absolute consistency is $\Pi_2$EXP-complete[1] in general, and $\Pi_4$P-complete in the bounded depth case.
- Data complexity of solution existence is in LOGSPACE, while combined complexity is NEXP-complete in general, and $\Sigma_3$P-complete for the bounded depth case.
- Solution building can be done in EXPSPACE in general, in PSPACE in the bounded depth case, and in P if the mapping is fixed.

The high combined complexity of the general case is disturbing, but the practically relevant bounded-depth case brings it down to acceptable low levels of polynomial hierarchy, and the polynomial data complexity of solution existence and solution building gives real hope for applications.

We then go on to generalize these results even further. We show that for mappings based on queries definable in MSO, absolute consistency remains decidable, and data complexity of solution existence is still polynomial.

The paper is organized as follows. After recalling the basic notions (Sect. 2), we study the absolute consistency problem for pattern-based mappings (Sect. 3) and its restriction to mappings of bounded depth (Sect. 4). We finish the first part of the paper with the examination of solution building and solution existence (Sect. 5). The second part starts with a definition of regular queries and then develops a representation designed to facilitate deciding absolute consistency and solution existence of mappings using regular queries instead of patterns (Sect. 6). Next, we give the decision procedures (Sect. 7), and finish with some suggestions for future work (Sect. 8).

Due to space limitations some proofs are omitted.

## 2. PRELIMINARIES

***Data trees, DTDs, automata.*** The abstraction of XML documents we use is *data trees*: unranked labelled trees storing in each node a natural number called *data value*. Formally, a *data tree* over a finite labelling alphabet $\Gamma$ is a structure $\langle T, \downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \ell_T, \rho_T\rangle$, where

- the set $T$ is an unranked tree domain, i.e., a prefix-closed subset of $\mathbb{N}^*$ such that $n\cdot i \in T$ implies $n\cdot j \in T$ for all $j < i$;
- the binary relations $\downarrow$ and $\rightarrow$ are the child relation ($n \downarrow n{\cdot}i$) and the next-sibling relation ($n{\cdot}i \rightarrow n{\cdot}(i{+}1)$);
- $\downarrow^*$ and $\rightarrow^*$ are transitive closures of $\downarrow$ and $\rightarrow$;
- the function $\ell_T$ is a labelling from $T$ to $\Gamma$;
- $\rho_T$ is a function from $T$ to $\mathbb{N}$. We say that a node $s \in T$ stores the value $v$ when $\rho_T(s) = v$.

[1] $\Pi_2$EXP is the second level of the exponential hierarchy.

Most often, when the interpretations of $\downarrow, \rightarrow, \ell_T$, and $\rho_T$ are understood, we write just $T$ to refer to a data tree. We use the terms "tree" and "data tree" interchangeably. [2] We write $|T|$ to denote the number of nodes of $T$.

The principal schema language we use are tree automata, abstracting Relax NG [17, 18].

A *nondeterministic word automaton* can be presented as a tuple $\mathcal{B} = \langle \Gamma, Q, q_I, \delta, F\rangle$, where $\Gamma$ is a finite input alphabet, $Q$ is a finite set of states, $q_I \in Q$ is the initial state, $F \subset Q$ is the set of final states, and $\delta \subseteq Q \times \Gamma \times Q$ is the transition relation. A run on a word $\sigma_1\sigma_2\cdots\sigma_n \in \Gamma^*$ is a sequence $q_1q_2\cdots q_{n+1} \in Q^*$ where $q_1$ is the initial state $q_I$, and $(q_i, \sigma_i, q_{i+1}) \in \delta$ for $i = 1, 2, \ldots, n$. A run is accepting if it ends in a final state, $q_{n+1} \in F$. A word is accepted if there is an accepting run for it. The set of accepted words is denoted by $L(\mathcal{B})$.

A *nondeterministic tree automaton* can be presented as a tuple $\mathcal{A} = \langle \Gamma, Q, \delta, F\rangle$ where $\Gamma$ is the input alphabet, $Q$ is a finite set of states, $F \subseteq Q$ is the set of final states, and $\delta$ is a function $Q\times\Gamma \to 2^{Q^*}$ such that $\delta(p, \sigma)$ is a regular language over $Q$ for every $p \in Q$, $\sigma \in \Gamma$. A run of $\mathcal{A}$ on a data tree $T$ is a labelling $\lambda : T \to Q$ such that for every $v \in T$ with $n$ children,

$$\lambda(v0)\lambda(v1)\cdots\lambda(v(n-1)) \in \delta(\lambda(v), \ell_T(v))\,.$$

If $v$ is a leaf, the condition reduces to $\varepsilon \in \delta(\lambda(v), \ell_T(v))$, which explains the lack of initial states in $\mathcal{A}$. A run $\lambda$ is *accepting*, if the root is labelled with a final state, $\lambda(\varepsilon) \in F$. A data tree $T$ is accepted by $\mathcal{A}$ if there is an accepting run for it. The set of trees accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$.

Throughout the paper we assume that all states are reachable, i.e., for each state $p \in Q$ there is a tree which evaluates to $p$. This can be guaranteed by polynomial preprocessing.

In the decision problems involving tree automata we assume that the regular languages $\delta(p, \sigma)$ are given by nondeterministic word automata $\mathcal{B}_{p,\sigma}$, and write $\|\mathcal{A}\|$ for the size of $\mathcal{A}$ including the automata $\mathcal{B}_{p,\sigma}$.

In this case, an *extended run* of $\mathcal{A}$ on $T$ is a run $\lambda$ together with a labelling $\kappa$ of $T$ such that whenever $\ell_T(v) = \sigma$ and $\lambda(v) = p$, and $v$'s children are $v_1, v_2, \ldots, v_k$, it holds that $q_I^{p,\sigma}, \kappa(v_1), \kappa(v_2), \ldots, \kappa(v_k)$ is an accepting run of $\mathcal{B}_{p,\sigma}$ on $\lambda(v_1), \lambda(v_2), \ldots, \lambda(v_k)$. The $\kappa$-label of the root is irrelevant, it can be any state of any $\mathcal{B}_{p,\sigma}$.

A simpler schema language is provided by DTDs. A *document type definition* (DTD) over a labelling alphabet $\Gamma$ is a pair $D = \langle r, P_D\rangle$, where

- $r \in \Gamma$ is a distinguished root symbol;
- $P_D$ is a function assigning regular expressions over $\Gamma - \{r\}$ to the elements of $\Gamma$, usually written as $\sigma \to e$, if $P_D(\sigma) = e$.

A data tree $T$ *conforms to* a DTD $D$, denoted $T \models D$, if its root is labelled with $r$ and for each node $s \in T$ the sequence of labels of children of $s$ is in the language of $P_D(\ell_T(s))$. The set of data trees conforming to $D$ is denoted $L(D)$.

Both DTDs and tree automata define languages of data trees. A DTD $D = \langle r, P_D\rangle$ over $\Gamma$ can be viewed as a tree automaton $\mathcal{A}_D = \langle \Gamma, \Gamma, \delta, \{r\}\rangle$, with $\delta(\sigma, \sigma) = P_D(\sigma)$ and $\delta(\sigma, \tau) = \emptyset$ for $\sigma \neq \tau$. We can think of DTDs as restricted tree automata.

[2] A different abstraction allows several *attributes* in each node, each attribute storing one data value [1, 3]. Attributes can be easily modelled with additional children.

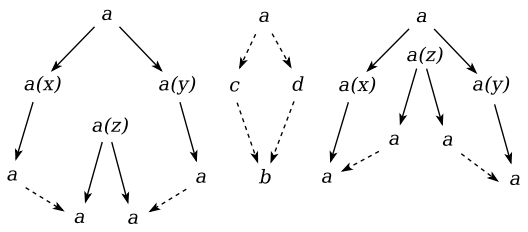


**Figure 1: Typical patterns**

*Patterns.* Patterns were originally invented as convenient syntax for conjunctive queries on trees [7, 8, 13]. While most schema mapping research has concentrated on tree-shaped patterns, definable with an XPath-like syntax [1, 3], the full expressive power of conjunctive queries is only guaranteed by DAG-shaped patterns [7, 8]. Indeed, a tree shaped pattern cannot express "there is a path from $a$ to $b$ via $c$ and $d$", but a DAG-shaped pattern or a conjunctive query can (see the middle pattern in Fig. 1). We base our mappings on DAG-shaped patterns, extending the setting used previously.

A *pattern* $\pi$ over $\Gamma$ can be presented as

$$\pi = \langle V, E_c, E_d, E_n, E_f, \ell_\pi, \xi_\pi \rangle$$

where $\langle V, E_c \cup E_d \cup E_n \cup E_f \rangle$ is a finite DAG whose edges are split into child edges $E_c$, descendant edges $E_d$, next sibling edges $E_n$, and following sibling edges $E_f$, $\ell_\pi$ is a partial function from $V$ to $\Gamma$, and $\xi_\pi$ is a partial function from $V$ to the set of variables. The range of $\xi_\pi$, denoted $\operatorname{Rg} \xi_\pi$, is the set of variables used by $\pi$. By $\|\pi\|$ we mean the size of the underlying DAG.

A data tree $\langle T, \downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \ell_T, \rho_T \rangle$ *satisfies* a pattern $\pi = \langle V, E_c, E_d, E_n, E_f, \ell_\pi, \xi_\pi \rangle$ under a valuation $\theta : \operatorname{Rg} \xi_\pi \to \mathbb{N}$, denoted $T \models \pi\theta$, if there exists a homomorphism

$$h : \langle V, E_c, E_d, E_n, E_f, \ell_\pi, \xi_\pi \circ \theta \rangle \to \langle T, \downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \ell_T, \rho_T \rangle,$$

i.e., a function $h : V \to T$ such that

- $h : \langle V, E_c, E_d, E_n, E_f \rangle \to \langle T, \downarrow, \downarrow^*, \rightarrow, \rightarrow^* \rangle$ is a homomorphism of relational structures;
- $\ell_T(h(v)) = \ell_\pi(v)$ for all $v \in \operatorname{Dom} \ell_\pi$; and
- $\rho_T(h(u)) = \theta(\xi_\pi(u))$ for all $u \in \operatorname{Dom} \xi_\pi$.

We write $\pi(\bar{x})$ to express that $\operatorname{Rg} \xi_\pi \subseteq \bar{x}$. For $\pi(\bar{x})$, instead of $\pi\theta$ we usually write $\pi(\bar{a})$, where $\bar{a} = \theta(\bar{x})$. We say that $T$ satisfies $\pi$, denoted $T \models \pi$, if $T \models \pi\theta$ for some $\theta$.

Note that we use the usual non-injective semantics, where different vertices of the pattern can be witnessed by the same tree node, as opposed to injective semantics, where each vertex is mapped to a different tree node [10]. Under the adopted semantics patterns are closed under conjunction: $\pi_1 \wedge \pi_2$ can be expressed by the disjoint union of $\pi_1$ and $\pi_2$.

Without loss of generality we assume throughout the paper that *siblings have a common parent*, i.e., whenever two vertices of a pattern are connected by a next-sibling or a following sibling edge, there is a vertex connected to both of them by a child edge.

Examples of patterns are given in Fig. 1. Solid and dashed arrows represent $E_c$ and $E_d$ respectively. Figure 2 shows examples of homomorphisms witnessing that the left pattern is satisfied for $x = y = z = 1$ and the right one is satisfied for $x = 3$, $y = 4$, $z = 0$. Note that in the left pattern, $x$ and $y$ always take the same value.

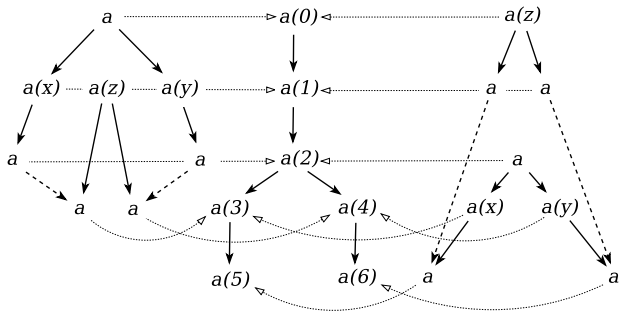


**Figure 2: Homomorphisms witness satisfaction**

*Pattern-based mappings.* A *(pattern-based) schema mapping* $\mathcal{M} = \langle \mathcal{S}_s, \mathcal{S}_t, \Sigma \rangle$ consists of a source schema $\mathcal{S}_s$, a target schema $\mathcal{S}_t$, and a set $\Sigma$ of *source-to-target dependencies (stds)* that relate source and target instances. The source and target schemas are given as tree automata or as DTDs. Stds are expressions of the form:

$$\pi(\bar{x}), \eta(\bar{x}) \;\longrightarrow\; \pi'(\bar{x}, \bar{y}), \eta'(\bar{x}, \bar{y}), \tag{1}$$

where $\pi, \pi'$ are patterns, and $\eta, \eta'$ are conjunctions of equalities and inequalities among $\bar{x}$ and $\bar{x}, \bar{y}$ respectively. We assume the usual safety condition: each variable used in $\eta$ is also used in $\pi$.

A pair of trees $(T, T')$ satisfies the std (1) if whenever $T \models \pi(\bar{a})$ and $\eta(\bar{a})$ holds, there is $\bar{b}$ such that $T' \models \pi'(\bar{a}, \bar{b})$ and $\eta'(\bar{a}, \bar{b})$ holds. Given a source $T \in L(\mathcal{S}_s)$, a target $T' \in L(\mathcal{S}_t)$ is called a *solution* for $T$ under $\mathcal{M}$ if $(T, T')$ satisfies all the stds in $\Sigma$. We let $\mathcal{M}(T)$ stand for the set of all solutions for $T$.

Suppose that the source schema is given as a DTD $r \to a^*b^*$, and the target schema is $r \to a\,b^*$. A mapping might be given, e.g, by a single std


where solid and dashed arrows represent $E_n$ and $E_f$, respectively. Observe that for this mapping every source tree has a solution. On the other hand, if we replace the source DTD with $r \to (a + b)^*$, some source trees will have no solutions.

In the complexity analysis of computational problems we use various restrictions on mappings. These include imposing tree-structure on patterns (instead of DAG structure), forbidding some axes in patterns, forbidding equality or inequality in stds, and restricting schemas to DTDs.

*Evaluation and satisfiability.* The evaluation problem is: given a pattern $\pi(\bar{x})$, a tuple $\bar{a}$, and a tree $T$, decide if $T \models \pi(\bar{a})$. The complexity of this problem is folklore.

PROPOSITION 2.1. *Data complexity of evaluating patterns is in* LOGSPACE *and combined complexity is in P.*

PROOF. A pattern is an FO query using relations $\rightarrow^*, \downarrow^*$, which can be computed in LOGSPACE (for $\downarrow^*$ move against the arrows). In consequence, a fixed pattern can be evaluated in LOGSPACE. Dynamic programming gives an algorithm with polynomial combined complexity. □

A pattern $\pi$ is satisfiable with respect to an automaton $\mathcal{A}$ if there is a tree $T \in L(\mathcal{A})$ such that $T \models \pi$. The *satisfiability problem* is: given a pattern $\pi$ and an automaton

$\mathcal{A}$, decide if $\pi$ is satisfiable with respect to $\mathcal{A}$. This problem was shown to be NP-complete for many variants of patterns [2, 4, 8, 14]. As we need the details of the NP-algorithm in some arguments, we sketch it here briefly.

Let $h$ be a homomorphism from $\pi$ to $T$. The *support* of $h$, denoted $\operatorname{supp} h$, is the subtree of $T$ obtained by removing all nodes that cannot be reached from $\operatorname{Rg} h$ by going up, left, and right.

LEMMA 2.2. *For each pattern $\pi$ satisfiable wrt an automaton $\mathcal{A}$, there exists $T \in L(\mathcal{A})$ and a homomorphism $h : \pi \to T$ with $|\operatorname{supp} h| \leq 12\|\pi\| \cdot \|\mathcal{A}\|^2$.*

PROOF. Take a tree $T \in L(\mathcal{A})$ satisfying $\pi$ and let $h$ be a homomorphism from $\pi$ to $T$. Divide the nodes of $\operatorname{supp} h$ into four categories: the nodes from the image of $h$ are *red*, the nodes that are not red and have more than one child that is an ancestor of a red node (or is red itself) are *green*, the others are *yellow* if they are ancestors of red nodes, and *blue* otherwise. Let $N_{\text{red}}$, $N_{\text{green}}$, $N_{\text{yellow}}$, and $N_{\text{blue}}$ be the numbers of red, green, yellow, and blue nodes.

By definition, $N_{\text{red}} \leq \|\pi\|$. Also $N_{\text{green}} \leq \|\pi\|$: when going bottom-up, each green node decreases the number of subtrees containing a red node by at least one, and since in the root we arrive with one subtree containing a red node, $N_{\text{green}} \leq N_{\text{red}}$. By a pumping argument we may assume that all yellow $\downarrow$-paths and all blue $\rightarrow$-paths in $\operatorname{supp} h$ are not longer than $\|\mathcal{A}\|$, The number of (maximal) yellow $\downarrow$-paths is at most $N_{\text{red}} + N_{\text{green}}$. Hence there are at most $2\|\pi\| \cdot \|\mathcal{A}\|$ yellow nodes. Since all blue nodes are siblings of nodes of other colours, the number of (maximal) blue $\rightarrow$-paths is at most $2(N_{\text{red}} + N_{\text{green}} + N_{\text{yellow}}) \leq 4\|\pi\| \cdot (\|\mathcal{A}\| + 1)$ and so $N_{\text{blue}} \leq 4\|\pi\| \cdot (\|\mathcal{A}\| + 1)\|\mathcal{A}\|$. Altogether we have at most $2\|\pi\|(\|\mathcal{A}\| + 1)(2\|\mathcal{A}\| + 1) \leq 12\|\pi\| \cdot \|\mathcal{A}\|^2$ nodes. □

PROPOSITION 2.3. *The satisfiability of patterns is in* NP.

PROOF. By Lemma 2.2, we can guess a homomorphism into a polynomial tree $T$ together with an "almost" run on $T$: in every leaf we additionally guess a sequence of states to which the missing subtrees should evaluate (by the pumping lemma, the sequence can be linear in the size of the automaton). Since all states are reachable, $T$ can be extended to a tree accepted by the automaton. Checking correctness of the "almost" run, and of the homomorphism is polynomial in the size of the pattern and the tree $T$, hence it is polynomial. □

# 3. ABSOLUTE CONSISTENCY FOR PATTERN-BASED MAPPINGS

As we have mentioned, a schema mapping $\mathcal{M} = \langle \mathcal{S}_s, \mathcal{S}_t, \Sigma \rangle$ is called *absolutely consistent* if every source tree has a solution, i.e., $\mathcal{M}(T) \neq \emptyset$ for every $T \in L(\mathcal{S}_s)$. We are interested in the following decision problem.

| | |
|---|---|
| PROBLEM: | ABCONS |
| INPUT: | Mapping $\mathcal{M} = \langle \mathcal{A}_s, \mathcal{A}_t, \Sigma \rangle$ |
| QUESTION: | Is $\mathcal{M}$ absolutely consistent? |

Assume for a while that the mapping $\mathcal{M}$ contains a single dependency $\pi(\bar{x}) \to \pi'(\bar{x}, \bar{y})$. The logical structure of the condition we need to check is: *for every* source tree $T$ *there exists* a target tree $T'$ such that *for every* $\bar{a}$ satisfying $T \models \pi(\bar{a})$ *there exists* $\bar{b}$ satisfying $T' \models \pi'(\bar{a}, \bar{b})$. To turn this into an algorithm we would need to show a bound on the size of trees that need to be considered.

Instead, we will try to change the order of quantifiers to the following: "for every $T$ and every $\bar{a}$, there exists $T'$ and some witnessing $\bar{b}$". The modified condition can be checked easily in $\Pi_2 P$. Indeed, what really matters is the equality type of $\bar{a}$ and $\bar{b}$, so it is enough to chose their entries from a fixed set of linear size. Furthermore, one does not need to guess $T$ and $T'$ explicitly, it is enough to witness their existence. By Lemma 2.2, there exists a polynomial witness.

To justify the reordering of the quantifiers, we would need to show that for every two target trees $T_1$ and $T_2$ there is a $\pi'$-*union* tree $T$, such that whenever $T_1 \models \pi'(\bar{a}, \bar{b})$ or $T_2 \models \pi'(\bar{a}, \bar{b})$, then $T \models \pi'(\bar{a}, \bar{b})$. Without a target schema a $\pi'$-union can be obtained by combining $T_1$ and $T_2$ under a fresh root node. In the presence of a target schema, however, a $\pi'$-union need not exist. For instance, consider the schema $r \to a\, b^*$, two trees $r[a(1), b(1), b(2)]$ and $r[a(2), b(3), b(4)]$, and $\pi'(x_1, x_2)$ saying "there is an $a$-node storing $x_1$ with a following sibling storing $x_2$". The first tree satisfies $\pi'(1, 2)$, and the second satisfies $\pi'(2, 4)$, but clearly no tree conforming to the schema can satisfy both $\pi'(1, 2)$ and $\pi'(2, 4)$.

We resolve this difficulty by showing that one can split every schema-defined language of data trees into subsets which are closed under $\pi'$-union, and have a description exponential in the size of the schema and $\pi'$. Those descriptions will be called *kinds*. In the example above, let $L_d$ consist of all trees conforming to the schema $r \to a\, b^*$, which store $d$ in the unique $a$-node. It is easy to see that each $L_d$ is closed under $\pi'$-union: the union tree simply needs to include all the $b$-nodes of $T_1$ and $T_2$.

A single kind usually cannot provide solutions to all source trees. For instance, in the example from the previous section the source schema is $r \to a^* b^*$ and $\pi(x_1, x_2)$ says "there is an $a$-node storing $x_1$ whose next sibling is a $b$-node, and some following sibling stores $x_2$". Clearly, a tree from $L_d$ can only be solution to source trees that store $d$ in the last $a$-node. Thus source documents have to be split into subsets admitting solutions of a single kind. It turns out that the latter is guaranteed by closure under pattern unions, which means that we can also use kinds to split the source documents.

Based on this we reformulate absolute consistency condition as "for every source kind $K$ there exists a target kind $K'$ such that for every $T$ of kind $K$ and every $\bar{a}$, there exists $T'$ of kind $K'$ and witnessing $\bar{b}$", which ultimately leads to a $\Pi_2$EXP-algorithm.

***Kinds.*** Recall the schema $r \to ab^*$. As we have noticed, the $a$-node is the critical area of every tree conforming to this schema. One can easily collect $b$-nodes from two trees in one tree, but there is always only one $a$-node. The main idea behind kinds is to distinguish the critical and the non-critical areas of trees. A kind will specify the critical areas entirely, including data values. The non-critical areas will be represented as "holes" in the tree, associated with the corresponding parts of the schema.

We find it most convenient to incorporate into each kind an extended run of the automaton representing the original schema. The run will have holes, just like the underlying tree. Each hole will be associated with a strongly connected component of the automaton, reflecting the intuition that the non-critical areas cover those parts of trees that can be repeated.

With a tree automaton $\mathcal{A} = \langle \Gamma, Q, \delta, F \rangle$ we associate a graph $G_{\mathcal{A}} = (Q, E)$, where $(p, p') \in E$ iff $Q^* p Q^* \cap \delta(p', \sigma)$ is nonempty for some $\sigma \in \Gamma$. We speak of strongly connected components (SCCs) of $\mathcal{A}$ meaning SCCs of $G_{\mathcal{A}}$. We say that an SCC is *non-trivial* if it contains an edge (it might have only one vertex though). A non-trivial SCC $X$ is *branching* if there exist $p, p_1, p_2 \in X$ such $Q^* p_1 Q^* p_2 Q^* \cap \delta(p, \sigma)$ is nonempty for some $\sigma$. If this is not the case, $X$ is *non-branching*. We also work with SCCs of word automata, defined in the natural way.

Let $\mathcal{A} = \langle \Gamma, Q, \delta, F \rangle$ and let the automaton recognizing $\delta(p, \sigma)$ be $\mathcal{B}_{p,\sigma} = \langle Q^{\mathcal{A}}, Q^{p,\sigma}, q_I^{p,\sigma}, \delta^{p,\sigma}, F^{p,\sigma} \rangle$ for $p \in Q, \sigma \in \Gamma$. Let $\mathcal{C}_{p,\sigma}$ and $\mathcal{D}$ be the sets of nontrivial SCCs of $\mathcal{B}_{p,\sigma}$ and $\mathcal{A}$, respectively. An *$\mathcal{A}$-kind* is a tree $K$ labelled with elements of

$$\mathcal{D} \cup \bigcup_{p,\sigma} \mathcal{C}_{p,\sigma} \cup \left( Q \times \Gamma \times \mathbb{N} \times \bigcup_{p,\sigma} Q^{p,\sigma} \right)$$

according to certain rules. The rules simply say that the partial run information is consistent. For technical convenience we also demand that $\mathcal{D}$-labelled nodes have no siblings, and SCC-labelled nodes are leaves, unless their label is a non-branching SCC of $\mathcal{A}$. The formal rules are as follows:

1. the root is labelled with $(p, \sigma, a, q)$ such that $p \in F$ (the state $q$ is irrelevant);
2. each internal node $v$ is either labelled with $(p, \sigma, a, q)$ and has a single child, labelled with an SCC $X \in \mathcal{D}$ containing $p$, or its sequence of children is labelled with
$$\ell_1, \ell_2, \ldots, \ell_k \in \mathcal{C}_{p,\sigma} \cup (Q \times \Gamma \times \mathbb{N} \times Q^{p,\sigma})$$
for some $p$ and $\sigma$ where
   (a) $v$ is labelled with $(p, \sigma, a, q)$ for some $a, q$ or with a non-branching $X \in \mathcal{D}$ such that $p \in X$ and $\ell_i = (p_i, \sigma_i, a_i, q_i)$ with $p_i \in X$ for some $1 \leq i \leq k$,
   (b) $\ell_1 = (p_1, \sigma_1, a_1, q_1)$ and $(q_I^{p,\sigma}, p_1, q_1) \in \delta^{p,\sigma}$,
   (c) for $1 \leq i < k$ either $\ell_i = (p_i, \sigma_i, a_i, q_i)$, $\ell_{i+1} = (p_{i+1}, \sigma_{i+1}, a_{i+1}, q_{i+1})$ and $(q_i, p_{i+1}, q_{i+1}) \in \delta^{p,\sigma}$, or one of $\ell_i, \ell_{i+1}$ is $X \in \mathcal{C}_{p,\sigma}$ and the other is $(p', \sigma', a', q')$ with $q' \in X$,
   (d) $\ell_k = (p_k, \sigma_k, a_k, q_k)$ and $q_k$ is accepting;
3. each leaf is labelled with a branching SCC from $\mathcal{D}$, an element of $\bigcup_{p,\sigma} \mathcal{C}_{p,\sigma}$, or $(p, \sigma, a, q)$ such that $\mathcal{B}_{p,\sigma}$ accepts the empty word.

To gain better understanding of this notion, let us examine a special case of DTDs admitting only trees of height 1. Recall that we can view DTDs as automata, whose state space is equal to the labelling alphabet $\Gamma$, the only final state is the root label, $r$, and only $\delta(\sigma, \sigma)$ are nonempty. If only trees of height 1 are admitted, for each $\sigma \neq r$, $\delta(\sigma, \sigma) = \{\varepsilon\}$. The only interesting transition is $\delta(r, r) = L(e)$, where $r \to e$ is the production for the root symbol in the DTD. Since no internal node can have label $r$, the tree automaton has only trivial SCCs. In consequence, each kind is a tree of height 1, whose root is labelled with $(r, r, d, p)$ for some meaningless $p$, and some $d \in \mathbb{N}$. The sequence of labels of root's children in $K$ is of the form

$$\alpha_0 X_1 \alpha_1 X_2 \alpha_2 X_3 \alpha_3 \ldots X_n \alpha_n \,,$$

where $X_i$ are SCCs of a word automaton $\mathcal{B}$ recognizing $\delta(r, r)$, $\alpha_i$ are essentially data words decorated with runs of $\mathcal{B}$ starting in $X_i$ (or the initial state if $i = 0$) and finishing in $X_{i+1}$ (or a final state if $i = n$).

The purpose of kinds is to define languages of data trees. A data tree $T$ *agrees with* $K$ if there exists an extended run $\lambda, \kappa$ of $\mathcal{A}$ on $T$ and a surjection $f \colon T \to K$ such that

- $T$'s root is mapped to $K$'s root
- if $v \to v'$, then $f(v) \to f(v')$ or $f(v) = f(v') = w$ and $w$ is labelled with an element of $\mathcal{D} \cup \bigcup_{p,\sigma} \mathcal{C}_{p,\sigma}$, and analogously for $v \downarrow v'$,
- $f(v)$ is labelled with $(\lambda(v), \ell_T(v), \rho_T(v), \kappa(v))$, or with an element of $\mathcal{D} \cup \bigcup_{p,\sigma} \mathcal{C}_{p,\sigma}$.

This definition formalizes the intuition of filling the holes: for $v$ labelled with an SCC $X$, $f^{-1}(v)$ is the area of the tree $T$ filling the hole represented by $v$ in $K$. If $X$ is a branching SCC of $\mathcal{A}$ or an SCC of $\mathcal{B}_{p,\sigma}$, this area is a subforest of $T$ consisting of a set of subtrees rooted at subsequent children of the same node (the inverse image of the parent of $v$). If $X$ is a non-branching SCC of $\mathcal{A}$, then $f^{-1}(v)$ is a subforest like above, minus the subforest formed by the inverse image of $v$'s descendants.

Let $L(K)$ denote the set of trees agreeing with $K$. From the definition above and the consistency conditions satisfied by the labelling of each kind it follows that $L(K) \subseteq L(\mathcal{A})$.

Recall once again the example discussed in the previous subsection. The languages $L_d$ cannot be expressed by kinds, but a similar effect can be obtained by taking kinds $K_{d_1,d_2,d_3}$ with the root labelled with $(r, r, d_1, q_I)$ and the root's children labelled with

$$(a, a, d_2, q)\{q\}(b, b, d_3, q) \,,$$

where $q_I, q$ are the states of the two-state automaton recognizing $a\, b^*$ (transitions are $q_I, a \to q$ and $q, b \to q$, the only final state is $q$).

The language $L(K_{d_1,d_2,d_3})$ consists of trees that store $d_1$ in the root, $d_2$ in the $a$-node, $d_3$ in the rightmost $b$-node, and have at least two $b$-nodes (something has to be mapped to the node labelled with $\{q\}$ in $K_{d_1,d_2,d_3}$). In order to cover all trees conforming to $r \to a\, b^*$, we need to add kinds whose root's children are labelled with $(a, a, d_2, q)(b, b, d_3, q)$ or $(a, a, d_2, q)$.

The outline of the algorithm presented at the end of the previous subsection suggests that we should be able to provide a small object witnessing that a pattern is satisfied in a tree agreeing with a kind. An argument similar to the proof of Lemma 2.2 and Proposition 2.3 leads to the following bound.

LEMMA 3.1. *For a kind $K$, a pattern $\pi$, and a tuple $\bar{a}$, satisfiability of $\pi(\bar{a})$ in a tree agreeing with $K$ can be witnessed by an object polynomial in the size of $\pi$, the height of $K$, and the branching of $K$.*

Recall that our aim is to show that each regular language can be covered by exponential kinds, each defining a language closed under $\pi'$-unions. Obviously, the choice of kinds has to take $\pi'$ into account. For instance, in our running example the kinds with the root's children labelled with $(a, a, d_2, q)\{q\}(b, b, d_3, q)$, $(a, a, d_2, q)(b, b, d_3, q)$ or $(a, a, d_2, q)$ are closed under union for $\pi'(x_1, x_2)$ saying "there is an $a$-node storing $x_1$ with a following sibling storing $x_2$", but not

for $\pi''(x_1, x_2)$ saying "there is an $a$-node storing $x_1$ whose next sibling stores $x_2$". For $\pi''$ one should use kinds given by

$$(a,a,d_2,q)(b,b,d_3,q)\{q\}(b,b,d_4,q)\,,$$
$$(a,a,d_2,q)(b,b,d_3,q)(b,b,d_4,q)\,,$$
$$(a,a,d_2,q)(b,b,d_3,q)\,,\text{ and }(a,a,d_2,q)\,,$$

where $d_2, d_3, d_4$ range over $\mathbb{N}$. Intuitively, to ensure closure under $\pi'$-union, we need to specify large enough areas of the tree around the holes. This is formalized by the following notion.

For $m \in \mathbb{N}$, we say that $K$ *has horizontal margins of size* $M$ if whenever a node $v$ is labelled with $X \in \mathcal{C}_{p,\sigma}$, there is a sequence of siblings $v_{-M}, v_{-M+1}, \ldots, v_M$ with $v_0 = v$ such that for all $0 < |j| \leq M$, $v_j$ is labelled with $(p_j, \sigma_j, a_j, q_j)$ such that $q_j \in X$.

Similarly, $K$ *has vertical margins of size* $M$ if

- whenever $v$ is labelled with a non-branching SCC $X$, there is a $\downarrow$-path $v_{-M}, v_{-M+1}, \ldots, v_M$ with $v_0 = v$ such that for all $0 < |j| \leq M$, $v_j$ is labelled with $(p_j, \sigma_j, a_j, q_j)$ where $q_j \in X$,
- whenever $v$ is labelled with a branching SCC $X$, there is a $\downarrow$-path $v_{-M}, v_{-M+1}, \ldots, v_0$ with $v_0 = v$ such that for all $-M \leq j < 0$, $v_j$ is labelled with $(p_j, \sigma_j, a_j, q_j)$ where $q_j \in X$,

Note that margins of size 1 are enforced already by the definition of kinds.

LEMMA 3.2. *For all $\mathcal{A}$, $T \in L(\mathcal{A})$ and $m, n \in \mathbb{N}$ there is an $\mathcal{A}$-kind $K$ with vertical margins of size $m$, horizontal margins of size $n$, height at most $(2m+1)\|\mathcal{A}\|$, branching at most $(2n+1)\|\mathcal{A}\|$, such that $T \in L(K)$.*

PROOF. Fix an extended accepting run $\lambda, \kappa$ of $\mathcal{A}$ on $T$. Let $\tilde{T}$ be a tree obtained from $T$ by labelling each node $v$ with $(\lambda(v), \ell_T(v), \rho_T(v), \kappa(v))$. Prune $\tilde{T}$ putting SCC labelled nodes as stubs according to the following rules.

First process the SCCs of $\mathcal{B}_{p,\sigma}$. Each $X$ in which the run stays for at least $2n+1$ steps becomes a $\mathcal{C}_{p,\sigma}$-node, and the first and last $n$ steps of the run spent in $X$ become the horizontal margins around this node. The subtrees rooted at the margin nodes are kept, but the subtrees rooted at the remaining nodes in the middle are lost together with their roots.

Next, deal with the SCCs of $\mathcal{A}$. Each maximal path $P$ on which the automaton stays in a branching SCC $X$ is cut off at depth $m+1$; under the freshly obtained leaf put a single child labelled with $X$.

For non-branching $X$ only consider paths of length at least $2m+1$. Cut off at depth $m+1$, add a fresh $X$ node, and under it put all subtrees originally rooted at the children of the $(m+1)$st node of the path, counting from the bottom.

Let $K$ be the resulting tree. By construction $T \in L(K)$. The bounds on the height and branching of $K$ follow from the observation that no SCC occurs more than once in a sequence of children, or a branch. □

The purpose of margins is to isolate the regions of the tree filling the holes from each other, and from the fixed part beyond the margins. In this way realizations of patterns touching the filling regions can be rearranged according to our needs. We use this in the lemma below, which constitutes the technical core of the upper bounds for pattern-based mappings.

LEMMA 3.3. *Let $\pi(\bar{x})$ be a pattern, and let $K$ be an $\mathcal{A}$-kind with horizontal and vertical margins of length $\|\pi\|$. For all $T_1, T_2, \ldots, T_n \in L(K)$ there is $T \in L(K)$ such that $T \models \pi(\bar{a})$ whenever $T_i \models \pi(\bar{a})$ for some $i \in \{1, 2, \ldots, n\}$.*

The proof of this lemma can be found in the on-line version of the paper [9]. To have a sample of the techniques used, let us consider a simple word example. Fix a word automaton $\mathcal{A}$ with the state space $Q$. We work with data words consistently decorated with states of $\mathcal{A}$. Abusing notation, we write $\alpha \in L(K)$ and $\alpha \models \pi(\bar{a})$ for $\alpha \in (\Gamma \times \mathbb{N} \times Q)^*$, when in fact those relations hold for the underlying data words.

Let $K$ be an $\mathcal{A}$-kind with margins of length $\|\pi\|$, and just one occurrence of an SCC, say $X$. Hence, $K = \alpha\beta X \beta'\alpha'$ where $\beta, \beta' \in (\Gamma \times \mathbb{N} \times X)^{\|\pi\|}$. Take $\gamma_1, \gamma_2 \in (\Gamma \times \mathbb{N} \times Q)^*$ such that $\gamma_1, \gamma_2 \in L(K)$. Then $\gamma_i = \alpha\beta\delta_i\beta'\alpha'$ for some $\delta_i \in (\Gamma \times \mathbb{N} \times X)^*$. It is easy to find $\eta \in (\Gamma \times \mathbb{N} \times X)^*$ such that $\gamma = \alpha\beta\delta_1\beta'\eta\tilde{\beta}\delta_2\beta'\alpha'$ is in $L(K)$, where $\tilde{\beta}$ is $\beta$ with the first letter chopped off.

Assume that $\gamma_i \models \pi(\bar{a})$ and let $h$ be a witnessing homomorphism from $\pi(\bar{a})$ to $\gamma_i$. If $h(\pi)$ is disjoint from $\delta_i$, partition the vertices of $\pi$ into those mapped to $\alpha\beta$ and those mapped to $\beta'\alpha'$. Observe that only $E_f$ edges can exist between those two parts of $\pi$. It follows easily that $\gamma \models \pi(\bar{a})$. Suppose that $h(\pi)$ is not disjoint from $\delta_i$. Since $|\beta| = |\beta'| = \|\pi\|$, by the pigeonhole principle there exist positions $j, j'$ of the words $\beta$, $\beta'$ which are not in $h(\pi)$. Partition the vertices of $\pi$ into those mapped to the left of $j$, between $j$ and $j'$, and to the right of $j'$. Again, only $E_f$ edges can connect different parts. It follows that $\gamma \models \pi(\bar{a})$. For example, for $i = 1$ a witnessing homomorphism is obtained by composing $h$ with the injection $g : \alpha\beta\gamma_1\beta'\alpha' \to \alpha\beta\gamma_1\beta'\eta\gamma_2\beta'\alpha'$ which maps $\alpha\beta\gamma_1\beta'_{\leq j'}$ to the prefix $\alpha\beta\gamma_1\beta'_{\leq j'}$ and $\beta'_{>j'}\alpha'$ to the suffix $\beta'_{>j'}\alpha'$, where $\beta'_{\leq j'}$ is the prefix of $\beta'$ of length $j'$ and $\beta'_{>j'}$ is the remaining suffix.

***Algorithm.*** We start with a simple observation that lets us concentrate on mappings with a single std.

LEMMA 3.4. *Let $\mathcal{M} = \langle \mathcal{S}_s, \mathcal{S}_t, \{\pi_i, \eta_i \longrightarrow \pi'_i, \eta'_i \mid i = 1, 2, \ldots, n\}\rangle$ such that each std uses different variables. For $I \subseteq \{1, 2, \ldots, n\}$, let*

$$\mathcal{M}_I = \Big\langle \mathcal{S}_s, \mathcal{S}_t, \Big\{ \bigwedge_{i\in I} \pi_i, \bigwedge_{i\in I} \eta_i \longrightarrow \bigwedge_{i\in I} \pi'_i, \bigwedge_{i\in I} \eta'_i \Big\} \Big\rangle\,.$$

*For all $T, T'$ it holds that $T' \in \mathcal{M}(T)$ iff $T' \in \mathcal{M}_I(T)$ for all $I \subseteq \{1, 2, \ldots, n\}$.* □

THEOREM 3.5. ABCONS *is in* $\Pi_2$EXP.

PROOF. We present the algorithm as a two-round game between two players, $\forall$ and $\exists$. In each round, $\forall$ moves first. Moves are made by the choice of an object of size exponential in $\|\mathcal{M}\|$ during the first round, and polynomial in $\|\mathcal{M}\|$ during the second round. The winning condition, a polynomial time property of the moves, is defined so that $\exists$ wins exactly if $\mathcal{M}$ is absolutely consistent. In the first round, $\forall$ states what *kind* of tree (in the sense defined above) is a counterexample to absolute consistency, while $\exists$ chooses the *kind* of tree that gives solutions to the purported counterexamples. In the second round, $\forall$ picks a tree of the declared kind and a tuple witnessing that the solutions fail, and $\exists$ tries to respond with a tree and a tuple that would prove $\forall$ wrong.

By Lemma 3.4 we can assume that $\mathcal{M}$ has a single std: $\forall$ can choose $\mathcal{M}_I$ as a part of the first move. Let an instance of AbCons be $\mathcal{M} = \langle \mathcal{A}_s, \mathcal{A}_t, \{\pi(\bar{x}), \eta(\bar{x}) \to \pi'(\bar{x}, \bar{y}), \eta'(\bar{x}, \bar{y})\}\rangle$, where $\mathcal{A}_s = \langle \Gamma, Q^s, \delta^s, F^s\rangle$ is the source automaton and $\mathcal{A}_t = \langle \Gamma, Q^t, \delta^t, F^t\rangle$ is the target automaton.

In the first round $\forall$ plays an $\mathcal{A}_s$-kind $K_\forall$ with horizontal and vertical margins $\|\pi\|$, height at most $(2\|\pi\|+1)\|\mathcal{A}_s\|$, and branching at most $(2\|\pi\|+1)\|\mathcal{A}_s\|$. The data values used in $K$ are to represent an equality type, so it is enough to choose them from $\{1, 2, \ldots, |K_\forall|\}$.

The response $K_\exists$ of $\exists$ is similar except that $\mathcal{A}_s$ is replaced by $\mathcal{A}_t$, $\pi$ is replaced by $\pi'$, and some of the nodes can store nulls taken from a fixed set $\{\bot_1, \bot_2, \ldots, \bot_{|K_\exists|}\}$. Each null can appear more than once in $K_\exists$. It is intended to represent "data distinct from whatever appears in the source tree". (Formally, we partition $\mathbb{N}$ into two infinite sets, $\mathbb{N}_s$ containing data values that are allowed in the source and $\mathbb{N}_t$ containing the data values allowed only on the target side.)

In the second round, $\forall$ chooses a data tuple $\bar{a}$ (*without* nulls) such that $\eta(\bar{a})$ holds, together with a polynomial object witnessing that $\pi(\bar{a})$ can be realized in a tree agreeing with $K_\forall$ (Lemma 3.1)

$\exists$ then responds with a tuple $\bar{b}$ (possibly *including* nulls) such that $\eta'(\bar{a}, \bar{b})$ holds, and a polynomial witness that $\pi'(\bar{a}, \bar{b})$ can be realized in a tree agreeing with $K_\exists$.

A player loses if he fails to make a move complying with the rules. If all moves are made, $\exists$ wins.

It remains to show that $\exists$ has a winning strategy if and only if $\mathcal{M}$ is absolutely consistent.

If $\mathcal{M}$ is not absolutely consistent, $\forall$'s strategy in the first round is to choose a data tree $T$ for which no solution exists and play $K_\forall$ such that $T \in L(K_\forall)$ (see Lemma 3.2).

If $|\mathcal{M}|$ is absolutely consistent, $\exists$'s strategy in the first round is to choose $K_\exists$ so that for every tree agreeing with $K_\forall$ there exists a solution agreeing with $K_\exists$. If such a $K_\exists$ cannot be produced, then there exists a tree agreeing with $K_\forall$ for which there is no solution at all, contradicting absolute consistency. To see this, reason as follows. For each possible response $K$ to $K_\forall$, let $T_K$ be a tree agreeing with $K_\forall$ for which there is no solution agreeing with $K$. By Lemma 3.3 there is a tree $T \in L(K_\forall)$ that satisfies $\pi(\bar{a})$ whenever one of the $T_K$'s satisfies $\pi(\bar{a})$. Since every $T' \in L(\mathcal{A}_t)$ agrees with one of $K$'s (Lemma 3.2), there is no solution for $T$.

In the second round, if $\mathcal{M}$ is not absolutely consistent, there is some $T$ agreeing with $K_\forall$ for which there is no solution. $\forall$'s strategy now is to choose a tuple $\bar{a}$ such that such that $T \models \pi(\bar{a}), \eta(\bar{a})$, but $\pi'(\bar{a}, \bar{y}), \eta'(\bar{a}, \bar{y})$ cannot be realized in a tree agreeing with $K_\exists$. Some suitable $\bar{a}$ exists, as otherwise a solution for $T$ could be obtained by an argument similar to the one above.

If $|\mathcal{M}|$ is absolutely consistent, then whatever $\bar{a}$ was played, $\pi'(\bar{a}, \bar{b}), \eta'(\bar{a}, \bar{b})$ can be realized in a tree from $L(K_\exists)$. $\exists$'s strategy is simply to choose suitable $\bar{b}$ and a witness. □

***Hardness.*** Consider the following problem:

| | |
|---|---|
| Problem: | $2^n$-universality |
| Input: | nondeterministic Turing machine $M$, number $n$ in unary |
| Question: | Does $M$ accept every word of length $2^n$ in at most $2^n$ steps? |

The problem is obviously in $\Pi_2$EXP. It can be also shown that it is hard for this class.

Lemma 3.6. $2^n$-universality *is* $\Pi_2$EXP*-complete.*

Proof. Take a $\Pi_2$EXP language $L$. There is a nondeterministic machine $M$ and $k \le l$ such that $x \in L$ iff for all $y$ such that $|y| \le 2^{|x|^k}$, $M$ accepts $\langle x, y\rangle$ ($M$ always stops after at most $2^{|x|^l}$ steps). Let $M_{x,k}$ write $x$ on the tape, mark the first $2^{|x|^k}$ positions of the input as relevant, and simulate $M$ on $x$ and the relevant part of the input. Since $k \le l$, the machine stops after at most $2^{C|x|^l}$ steps for some $C$ independent of $x$. Hence, $x \in L$ iff $\langle M_{x,k}, 1^{C|x|^l}\rangle$ is a positive instance of $2^n$-universality. □

We give a reduction from $2^n$-universality to AbCons for a restricted class of mappings, based on tree-shaped patterns using only vertical navigation. For such patterns we use the following syntax [3]:

$$\begin{array}{lcll} \pi & ::= & \sigma(x)[\lambda] \mid \sigma[\lambda] & \text{patterns} \\ \lambda & ::= & \varepsilon \mid \pi \mid /\!/\pi \mid \lambda, \lambda & \text{lists} \end{array} \tag{2}$$

where $\sigma \in \Gamma \cup \{\_\}$. That is, a tree pattern is given by its root node and a listing of its subtrees. A subtree can be rooted at a child of the root (corresponding to $\pi$ in the definition of $\lambda$), or its descendant (corresponding to $/\!/\pi$). The wildcard symbol _ is used to denote a vertex without a label. We also write $\sigma$ for $\sigma[\,]$, $\sigma/\pi$ for $\sigma[\pi]$, $\sigma/\!/\pi$ for $\sigma[/\!/\pi]$, $\sigma[\lambda, \bigwedge_{i=1}^k \pi_i, \lambda']$ for $\sigma[\lambda, \pi_1, \pi_2, \ldots, \pi_k, \lambda']$, and similarly for $\sigma(x)$.

Theorem 3.7. AbCons *is* $\Pi_2$EXP*-hard, even if schemas are non-recursive DTDs, and the only relations available are child and equality on the target side.*

Proof. Let an instance of $2^n$-universality be $n$, and a Turing machine $M$ with states $q_0, q_1, \ldots, q_f$. W.l.o.g. we assume that $q_f$ is the only final accepting state. For simplicity we assume that the tape alphabet is $0, 1, \flat$.

The idea of the reduction is to encode the input word in the source tree, and the run in the target tree. The run will be encoded as a sequence of $2^n$ configurations of length $2^n$. Additionally, in the source tree we store a linear order of length $2^n$, which will be used to address configurations and their cells, and in the target tree we store a specially preprocessed transition relation of $M$.

The source DTD is given as

$$\begin{aligned} r &\to \mathit{ord}\, q_0\, q_1\, \ldots\, q_f \perp \mathit{zero}\, \mathit{one}\, \mathit{blank}\,, \\ \mathit{ord} &\to a_1\, b_1\,, \\ a_i, b_i &\to a_{i+1}\, b_{i+1}\,, \\ a_n, b_n &\to c\,, \\ c &\to \mathit{zero} \mid \mathit{one} \end{aligned}$$

with $i = 1, 2, \ldots, n-1$ and the target DTD given as

$$\begin{aligned} r &\to a_1\, b_1\, (tr)^d\,, \\ tr &\to \mathit{st}_1\, \mathit{sym}_1\, \mathit{st}_2\, \mathit{sym}_2\, \cdots\, \mathit{st}_6\, \mathit{sym}_6, \\ a_i, b_i &\to a_{i+1}\, b_{i+1}\,, \\ a_{2n}, b_{2n} &\to \mathit{confnum}\, \mathit{cellnum}\, \mathit{st}\, \mathit{sym}\,, \end{aligned}$$

where $i = 1, 2, \ldots, 2n-1$, and $d$ will be defined shortly. Under $a_{2n}$ and $b_{2n}$ nodes we store a configuration number, a cell number, a state, and a tape symbol. The $tr$ nodes store

$\hat{\delta}$, the *extended transition relation* of $M$, describing possible transitions in a window of three consecutive tape cells. Formally, $\hat{\delta} \subseteq (\{q_0, q_1, \ldots, q_f, \bot\} \times \{0, 1, \flat\})^6$, where $\bot$ means "the head is elsewhere", and $(p_1, \sigma_1, p_2, \sigma_2, \ldots, p_6, \sigma_6) \in \hat{\delta}$ iff at most one of $p_1, p_2, p_3$ is not equal to $\bot$, and $p_4\sigma_4 p_5\sigma_5 p_6\sigma_6$ is obtained from $p_1\sigma_1 p_2\sigma_2 p_3\sigma_3$ by performing a transition of $M$. In particular, if $p_1 = p_2 = p_3 = \bot$, it is possible that $p_4 \neq \bot$ or $p_6 \neq \bot$. Note that $\hat{\delta}$ can be computed in P. The constant $d$ used in the target DTD is equal to $|\hat{\delta}|$.

Assume for a while that we only need to handle source trees in which all data values are distinct. The $c$-nodes encode a linear order of length $2^n$, and their leaves labelled with *zero* or *one* encode the input word (the data values they store are ignored). The children of the root, $q_i$, $\bot$, *zero*, *one*, and *blank*, store data values encoding the states and tape symbols of $M$. To ensure that the extended transition relation is stored properly on the target side, for each $(q_1, \sigma_1, q_2, \sigma_2, \ldots, p_6, \sigma_6) \in \hat{\delta}$ add an std

$$r[q_1(x_1), \hat{\sigma}_1(y_1), q_2(x_2), \hat{\sigma}_2(y_2), \ldots, q_6(x_6), \hat{\sigma}_6(y_6)] \longrightarrow$$
$$\longrightarrow r/tr\left[\bigwedge_{i=1}^{6} st_i(x_i), \bigwedge_{i=1}^{6} sym_i(y_i)\right]$$

where $\hat{0} = zero$, $\hat{1} = one$, and $\hat{\flat} = blank$. Note that $d$ different stds are introduced, so all $tr$-nodes in the target tree are filled according to $\hat{\delta}$.

Now we need to ensure that the target tree encodes an accepting run of $M$. In the stds we use auxiliary patterns $Begin(x)$, $Begin_\sigma(x)$, $End(x)$, $Succ(x, y)$, and $Succ_3(x, y, z)$, which we define after describing the stds, as well as

$$Cell(x, y, u, v) = \_[confnum(x), cellnum(y), st(u), sym(v)]\,.$$

We use $/\!/$ as abbreviation only. Each use can be replaced with a sequence of $\_$ and $/$ operators of suitable length.

To build the first configuration we copy the input word with the head in state $q_0$ over the first cell,

$$r[Begin_\sigma(x), q_0(u), \sigma(v)] \longrightarrow r/\!/Cell(x, x, u, v)\,,$$
$$r[Begin(x), /\!/c(y)/\sigma, \bot(u), \sigma(v)] \longrightarrow r/\!/Cell(x, y, u, v)\,,$$

with $\sigma \in \{zero, one\}$. Transition correctness is ensured by

$$r[Succ(x_0, x_1),\ Succ_3(y_1, y_2, y_3)] \longrightarrow$$
$$\longrightarrow r\left[\bigwedge_{i,j} /\!/Cell(x_i, y_j, u_{3i+j}, v_{3i+j}), tr\left[\bigwedge_{i=1}^{6} st_i(u_i), \bigwedge_{i=1}^{6} sym_i(v_i)\right]\right].$$

Finally, the last configuration needs to be accepting (w.l.o.g. we assume that the accepting state is looping),

$$r[End(x), q_f(u)] \longrightarrow r/\!/Cell(x, y, u, v)\,.$$

It remains to define the auxiliary patterns. With every $c$-node $v$ we associate the sequence of $a$'s and $b$'s on the path leading from the root to $v$. This sequence is interpreted as a binary number ($a$ read as 0, $b$ read as 1), which is the position of $v$ in the order. The first three patterns are defined easily:

$$Begin(x) = ord/a_1/a_2/\cdots/a_n/c(x)\,,$$
$$Begin_\sigma(x) = ord/a_1/a_2/\cdots/a_n/c(x)/\sigma\,,$$
$$End(x) = ord/b_1/b_2/\cdots/b_n/c(x)\,.$$

The remaining two cannot be defined as single patterns, but can be expressed as disjunctions of patterns. As we only use the auxiliary patterns on the source side of stds, disjunction can be easily eliminated at the cost of multiplying stds.

$$Succ(x, y) = \bigvee_{i=1}^{n} /\!/_-\big[a_i/b_{i+1}/b_{i+2}/\cdots/b_n/c(x),$$
$$b_i/a_{i+1}/a_{i+2}/\cdots/a_n/c(y)\big]\,,$$
$$Succ_3(x, y, z) =$$
$$\bigvee_{i=1}^{n-1} /\!/_-\Big[a_i/b_{i+1}/b_{i+2}/\cdots/b_{n-1}\big[a_n/c(x), b_n/c(y)\big],$$
$$b_i/a_{i+1}/a_{i+2}/\cdots/a_n/c(z)\Big] \ \vee$$
$$\vee \bigvee_{i=1}^{n-1} /\!/_-\Big[a_i/b_{i+1}/b_{i+2}/\cdots/a_n/c(x),$$
$$b_i/a_{i+1}/a_{i+2}/\cdots/a_{n-1}\big[a_n/c(y), b_n/c(z)\big]\Big]\,.$$

We claim that the mapping we have just defined is absolutely consistent iff the answer to $n$-UNIVERSALITY is "yes". Assume that the mapping is absolutely consistent. Every input word $w$ can be encoded in a source tree using distinct data values. An inductive argument shows that a solution to such a tree encodes an accepting run of $M$ on $w$. Conversely, if the answer is "yes", for each source tree $S$ using distinct data values, a solution is obtained from the run of $M$ on the word encoded in the sequence of *zero* and *one* leaves of $S$. What if $S$ uses some data values more than once? For a function $h : \mathbb{N} \to \mathbb{N}$ and a tree $U$, let $h(U)$ be the tree obtained from $U$ by replacing each data value $a$ with $h(a)$. Now, let $S'$ be a tree with the structure identical as $S$, but using distinct data values, and let $h$ be a function on data values such that $h(S') = S$. By the previously considered case, there is a solution $T'$ for $S'$. Since our mapping does not use inequality on the target side, nor equality on the source side, $h(T')$ is a solution for $h(S') = S$. □

*Remark.* In the reduction above we can remove disjunction from the DTDs at a cost of relaxing restrictions on patterns. We only need to modify the encoding of the 0s and 1s of the input word. One way is to set $c \to c_1\, c_2$, and replace $c/zero$ with $c[c_1(z), c_2(z')], z \neq z'$ and $c/one$ with $c[c_1(z), c_2(z)]$. Another way is to set $c \to c_1\, c_2^*\, c_3$ and use next-sibling to distinguish between $c_1c_2$ and $c_1c_2^+c_3$.

# 4. BOUNDED DEPTH MAPPINGS

We have seen that absolute consistency is highly untractable even for tree-shaped patterns using only vertical axes and non-recursive DTDs with very simple productions. In this section we show that the complexity can be lowered substantially if the height of trees is bounded by a constant. We say that a mapping $\mathcal{M}$ *has depth at most* $d$ if the source and target schema only admit trees of height at most $d$.

THEOREM 4.1. ABCONS *for mappings of bounded depth is in* $\Pi_4$P.

PROOF. We claim that the general algorithm presented in Theorem 3.5 has the desired complexity for mappings of bounded depth.

Assume that some $\mathcal{A}$ only accepts trees of height at most $d$ and let $K$ be an $\mathcal{A}$-kind. Obviously $K$ has height at most $d$. (In fact, $K$ contains no nodes labelled with SCCs of $\mathcal{A}$, as otherwise arbitrarily high trees would agree with $K$.)

In consequence, the kinds played in the first round have polynomial branching, and bounded depth, hence are polynomial. In the second round, polynomial objects are played. As the correctness of the moves is polynomial, this gives a $\Pi_4$P-algorithm. □

*Remark.* A small modification of our techniques makes it possible to prove Theorem 4.1 for a more general definition of boundedness: $\mathcal{M}$ *has depth at most* $d$ if every pattern it uses can only be realized within the initial $d$ levels of every tree conforming to the schema. This includes mappings using patterns starting at the root, that do not use descendant, nor $E_c$ paths of length greater than $d$.

We now show that the absolute consistency is $\Pi_4$P-hard even for word schema mappings, which can be viewed as depth 1 tree mappings. In the word case schemas are regular expressions (or word automata), and stds only use word patterns, i.e., patterns with $E_c = E_d = \emptyset$. In fact, we only need patterns that are disjoint unions of $E_n$-paths. We write them as sequences of data words with variables, e.g., $a(x)b(y), a(x)$ is a pattern consisting of two $E_n$-paths.

THEOREM 4.2. ABCONS *is* $\Pi_4$P*-hard for bounded-depth mappings, even if depth is 1, and the only relations available are next sibling, and equality on the target side.*

PROOF. We provide a reduction from TAUTOLOGY for $\Pi_4$ quantified propositional formulas. Let

$$\varphi = \forall x_1, x_2, \ldots, x_n \exists y_1, y_2, \ldots, y_n \\ \forall u_1, u_2, \ldots, u_n \exists v_1, v_2, \ldots, v_n \bigwedge_{i=1}^{m} X_i \vee Y_i \vee Z_i$$

with $X_i, Y_i, Z_i \in \{x_j, y_j, u_j, v_j, \bar{x}_j, \bar{y}_j, \bar{u}_j, \bar{v}_j \mid j = 1, \ldots, n\}$. Let the source and target schemas be

$$\mathcal{S}_s = (a_1|a_1')(a_2|a_2') \ldots (a_n|a_n')ee\,,$$
$$\mathcal{S}_t = eeea_1a_1a_2a_2 \ldots a_na_nb_1b_1b_2b_2 \ldots b_nb_n(\#ggg)^7\,.$$

The source word encodes a valuation of $x_1, x_2, \ldots, x_n$, $a_i$ means that $x_i$ is *true*, $a_i'$ means it is *false*. In $e$-positions we store values representing *true* and *false*. On the target side, we want to keep a copy of the valuation of $x_i$'s and a guessed valuation of $y_i$'s, except this time we use a different coding. The first position labelled with $a_i$ stores the value of $x_i$, *true* or *false*, and the following position stores the value of the negation of $x_i$. Similarly, $b_i$'s store values of $y_i$. We also want in the target word two copies of *true* and a copy of *false* arranged so as to enable nondeterministic choice between a pair (*true*, *false*) or (*false*, *true*), as well as all triples with at least one entry *true*, which will help us to check that each clause of $\varphi$ is satisfied.

Let us now describe the stds. First we make sure that values representing true and false are copied properly,

$$e(x)e(y) \longrightarrow e(x)e(y)e(x)\,,$$

for each $i$ translate the $a_i/a_i'$ coding of values of $x_i$ into true/false coding,

$$a_i, e(t)e(f) \longrightarrow a_i(t)a_i(f)\,,$$
$$a_i', e(t)e(f) \longrightarrow a_i(f)a_i(t)\,,$$

and enforce in the target word all triples with at least one entry *true*,

$$e(t)e(f) \longrightarrow \#g(f)g(f)g(t)\#g(f)g(t)g(f)\# \ldots \#g(t)g(t)g(t)\,.$$

Next, we guess a value of $y_i$ for each $i$,

$$e(t)e(f) \longrightarrow b_i(t), b_i(f)\,,$$

and ensure that it makes the internal $\Pi_2^p$ part of $\varphi$ true for $x_1, x_2, \ldots, x_n$ encoded in the source word:

$$e(u_1), e(u_2), \ldots, e(u_n), e(t)e(f) \longrightarrow \\ e(u_1)e(\bar{u}_1), e(u_2)e(\bar{u}_2), \ldots, e(u_n)e(\bar{u}_n), \\ e(v_1)e(\bar{v}_1), e(v_2)e(\bar{v}_2), \ldots, e(v_n)e(\bar{v}_n), \\ a_1(x_1)a_1(\bar{x}_1)a_2(x_2)a_2(\bar{x}_2) \ldots a_n(x_n)a_n(\bar{x}_n), \\ b_1(y_1)b_1(\bar{y}_1)b_2(y_2)b_2(\bar{y}_2) \ldots b_n(y_n)b_n(\bar{y}_n), \\ g(X_1)g(Y_1)g(Z_1), g(X_2)g(Y_2)g(Z_2), \ldots, g(X_m)g(Y_m)g(Z_m)$$

(the literals $X_j, Y_j, Z_j$ are taken from $\varphi$).

The obtained mapping is absolutely consistent iff $\varphi$ is a tautology. Indeed, if the mapping is absolutely consistent, in particular it has a solution for each source word that uses two different data values in $e$-positions. By construction, such words have solutions iff $\varphi$ is a tautology. If the data values in $e$-positions are equal, the stds are satisfied trivially. □

*Remark.* Disjunction can be eliminated from the source schema at a cost of allowing data comparisons on the source side. To achieve this, replace $(a_i|a_i')$ with $a_ia_i$, and encode the truth value as (in)equality of the two data values.

# 5. BUILDING SOLUTIONS

In previous sections we examined existence of solutions for all source trees. The task of this section is to build a solution for a given tree. Techniques used in previous sections give us a procedure with polynomial data complexity; the combined complexity is higher.

THEOREM 5.1. *For a mapping* $\mathcal{M}$ *and a source tree* $T$ *one can build a solution (or determine that it does not exist) in* EXPSPACE *in general, in* PSPACE *if the mapping has bounded depth, and in* P *if the mapping is fixed.*

PROOF. As a first step we remove stds whose source sides are not satisfied in $T$. (Checking if $\pi$ is satisfied in $T$ can be done in NP, and for fixed $\pi$ in P.) The remaining stds can be merged into one just like in Lemma 3.4.

By Lemma 3.2, if there is a solution to $T$, it agrees with one of the possible kinds which $\exists$ can play in the first round of the game described in the proof of Theorem 3.5. Note that the size of those kinds only depends on $\mathcal{M}$ (single exponential and polynomial for bounded depth), and so can be viewed as fixed. The kinds only use data values from $T$ and nulls from a set independent of $T$ (single exponential in $\|\mathcal{M}\|$, polynomial for bounded depth). Therefore, the number of possible kinds is polynomial in $|T|$ and the problem amounts to finding a solution agreeing with a given kind $K$.

First, for each tuple $\bar{a}$ such that $\eta(\bar{a})$ holds and $T \models \pi(\bar{a})$, we need to find a tuple $\bar{b}$ satisfying $\eta'(\bar{a}, \bar{b})$, whose entries are data values used in $T$ or $K$, or nulls from $\{\perp_1', \perp_2', \ldots, \perp_{|\bar{y}|}'\}$, and a tree $T_{\bar{a}}$ agreeing with $K$ such that $T_{\bar{a}} \models \pi'(\bar{a}, \bar{b})$. Clearly, the size of $T_{\bar{a}}$ does not depend on $T$. (A pumping argument similar to the one in Lemma 2.2 shows that each part of $T_{\bar{a}}$ that gets mapped into a node of $K$ labelled with an SCC can have size single exponential in $\|\mathcal{M}\|$, and polynomial for bounded depth.) In consequence, we can find $\bar{b}$ and $T_{\bar{a}}$ by exhaustive search. If for some $\bar{a}$ the search fails, there is no solution to $T$ agreeing with $K$.

The last step is to merge $T_{\bar{a}}$'s into a single solution satisfying all $\pi'(\bar{a}, \bar{b})$. By the proof of Lemma 3.3, this can be done in time polynomial in the total size of the trees and the kind $K$. □

As we have seen in the proof, the solution is at most polynomial in the size of the source tree and exponential in the size of the mapping. Those bounds are tight: the smallest tree conforming to the target DTD (or accepted by the target automaton) might need to be exponential, and a simple copying rule $r/\!\!/_{-}(x) \longrightarrow r/\!\!/_{-}(x)$ makes the solution at least as large as the source tree. This however does not give matching complexity lower bounds for solution building.

To give more precise bounds let us consider solution existence, a decision version of solution building:

| | |
|---|---|
| PROBLEM: | SOLEX |
| INPUT: | mapping $\mathcal{M}$, source tree $T$ |
| QUESTION: | Is $\mathcal{M}(T)$ nonempty? |

We also examine data complexity of solution existence, i.e., the complexity of the following problem:

| | |
|---|---|
| PROBLEM: | SOLEX($\mathcal{M}$) |
| INPUT: | source tree $T$ |
| QUESTION: | Is $\mathcal{M}(T)$ nonempty? |

THEOREM 5.2. SOLEX *is* NEXP*-complete in general, and* $\Sigma_3$P*-complete for mappings of bounded depth.* SOLEX($\mathcal{M}$) *is in* LOGSPACE.

PROOF. To get the upper bounds for SOLEX proceed just like in the proof of Theorem 5.1, only instead of examining every possible kind played by $\exists$, choose it nondeterministically. For $\Sigma_3$P multiple stds are eliminated during the game: $\exists$ guesses the stds with source sides satisfied, merges them, and plays a kind for the new mapping; $\forall$ either displays an std omitted by $\exists$ and wins immediately, or continues with the new mapping according the old rules.

The NEXP lower bound can be obtained by a modification of the reduction described in the proof of Theorem 3.7. The problem we reduce from is: given a nondeterministic Turing machine $M$ and $n \in \mathbb{N}$, does $M$ accept $0^{2^n}$ in at most $2^n$ steps. We keep the same mapping, and for the source tree we take a tree whose all $c$-nodes have a *zero*-child, and whose data values are distinct.

Similarly, modifying the reduction from the proof of Theorem 4.2, we get $\Sigma_3$P-hardness for word mappings.

Let us move to SOLEX($\mathcal{M}$). Since $\mathcal{M}$ has constant size, by Lemma 3.4 we can assume that $\mathcal{M}$ is has a single std, $\mathcal{M} = \langle \mathcal{A}_s, \mathcal{A}_t, \{\pi(\bar{x}), \eta(\bar{x}) \to \pi'(\bar{x}, \bar{y}), \eta'(\bar{x}, \bar{y})\}\rangle$. The kind played by $\exists$ is constant size and takes data values from a set $A$ of linear size. In consequence, it can be stored in logarithmic space. Instead of guessing it, we can iterate over all possibilities. It remains to check in LOGSPACE if there is a solution of a given kind $K$.

By Lemma 3.3 it is enough to check if for each $\bar{a}$ such that $\eta(\bar{a})$ holds and $T \models \pi(\bar{a})$ there is a tuple $\bar{b}$ and a tree $T'$ of the kind $K$ such that $\eta'(\bar{a}, \bar{b})$ holds and $T' \models \pi'(\bar{a}, \bar{b})$. Again, we can iterate over all $\bar{a}$ and $\bar{b}$ with entries from $A$. For fixed $\bar{a}, \bar{b}$, $T \models \pi(\bar{a})$ can be checked in LOGSPACE by Proposition 2.1. The remaining tests can be carried out in constant time, modulo suitable encoding of data values in $K$, $\bar{a}$, $\bar{b}$, which can be prepared in LOGSPACE. □

# 6. MSO QUERIES

In this section we extend the language of schema mappings, by replacing patterns with formulas of monadic second-order logic (MSO). We prove that for this richer language, all the problems are still decidable. We do not establish the precise complexities.[3]

The general setup is similar to the one in the previous sections. A single source-to-target dependency is also an implication of the form

$$\pi(\bar{x}), \eta(\bar{x}) \longrightarrow \pi'(\bar{x}, \bar{y}), \eta'(\bar{x}, \bar{y}).$$

The formulas are $\eta$ and $\eta'$ are boolean combinations of equality constraints on their arguments (this is slightly more general than before, since previously we had conjunctions of equalities and inequalities). The main generalization concerns $\pi$ and $\pi'$. Previously, the tuples of data values selected by $\pi$ and $\pi'$ came from patterns. In this section, we allow a richer syntax for $\pi$ and $\pi'$, where formulas of MSO are used.

To avoid confusion, we will explicitly distinguish here between queries that select tuples of nodes of a data tree (which we call *node-selecting queries*) and queries that select tuples of data values (which we call *data-selecting queries*). We use letters $\alpha, \beta$ for node-selecting queries and $\pi$ for data-selecting queries. Every boolean combination $\eta$ of equalities and inequalities on data values can be seen as a data-selecting query. There is also a natural transformation from node-selecting queries to data-selecting queries, defined as follows. When $\alpha(x_1, \ldots, x_k)$ is a node-selecting query, then $\pi_\alpha(x_1, \ldots, x_k)$ is a data-selecting query that selects a $k$-tuple of data values $d_1, \ldots, d_k$ in a data tree if the query $\alpha$ selects some $k$-tuple of nodes $x_1, \ldots, x_k$ in the data tree such that node $x_i$ has data value $d_i$ for $i = 1, \ldots, k$. The idea is that in this section, we will study stds where $\pi$ is not a pattern, but of the form $\pi_\alpha$, where $\alpha$ is a node-selecting query of MSO that does not talk about data values.

***MSO formulas.*** In the previous sections, we used patterns to describe $\pi$ and $\pi'$ in the stds. There are some limitations of patterns, however. For instance, every tuple of nodes that matches a pattern will still match it if new nodes are added to the tree. In some natural queries, this is no longer true. For example, we might be interested in node pairs $x, y$ where $y$ is a descendant of $x$ and all nodes on the path from $x$ to $y$ have label $a$:

$$a(x) \wedge a(y) \wedge x \downarrow^* y \wedge \forall z\big(x \downarrow^* z \downarrow^* y \Rightarrow a(y)\big).$$

In this section, we use MSO formulas to define node-selecting queries. We use the following definition of MSO formulas: they can quantify over individual nodes, they can quantify over sets of nodes, they can use predicates to test the label of a node, and they can use binary predicates for the ancestor and following sibling relations. A formula of this logic with $k$ free individual variables defines a $k$-ary node-selecting query. (We do not use free set variables.) Note that we *do not allow* predicates for data values in MSO logic, and therefore whether or not a tuple of nodes gets selected depends only on the structure of the tree, the labels of the tree, but not

[3]The questions of complexity are most interesting when the queries are given not by MSO formulas, but by automata. If the input contains MSO formulas, then any algorithm will be nonelementary, because satisfiability for MSO formulas is nonelementary.

the data values. This restriction is very important. (If we allow predicates for data values in MSO logic, satisfiability becomes undecidable.) We also use the name *regular queries* for node-selecting queries defined by MSO formulas.

***Regular schema mappings.*** A *regular schema mapping* $\mathcal{M}$ is a tuple $\langle \mathcal{S}_s, \mathcal{S}_t, \Sigma \rangle$, where $\mathcal{S}_s$ is a source schema, $\mathcal{S}_t$ is a target schema, and $\Sigma$ is a set of stds of the form

$$\pi_\alpha(\bar{x}), \eta(\bar{x}) \longrightarrow \pi_{\alpha'}(\bar{x}, \bar{y}), \eta'(\bar{x}, \bar{y}) \tag{3}$$

where $\pi_\alpha, \pi_{\alpha'}$ are data-selecting queries obtained from MSO formulas $\alpha, \alpha'$, and $\eta, \eta'$ are boolean combinations of equalities among $\bar{x}$ and $\bar{x}, \bar{y}$ respectively. (The assumption that every variable of $\alpha$ is used by $\alpha'$ is not restrictive, as $\alpha'$ can ignore some variables.) A pair of trees $(T, T')$ satisfies the std (3) if for every tuple of data values $\bar{a}$ selected by both $\pi_\alpha$ and $\eta$ in $T$, there exists a tuple of data values $\bar{b}$ such that $\bar{a}\bar{b}$ is selected by both $\pi_{\alpha'}$ and $\eta'$ in $T'$. Given $T \in L(\mathcal{S}_s)$, a tree $T'$ is a *solution* to $T$ iff $T' \in L(\mathcal{S}_t)$ and $(T, T')$ satisfies every std in $\Sigma$.

Once again, we would like to emphasize that the MSO formulas $\alpha, \alpha'$ do not depend on the data values in the trees $T$ and $T'$. The data values are only inspected by $\eta$ and $\eta'$ in a very restrictive way: by boolean combinations of equalities.

The goal of this section is to show that the decision problems studied in this paper are decidable also for regular schema mappings.

THEOREM 6.1. *For regular schema mappings* AbCons *is decidable,* SolEx *is decidable, and* SolEx($\mathcal{M}$) *is in* P.

LEMMA 6.2. *Without loss of generality, we can assume that in the regular schema mapping*

$$\mathcal{M} = \langle \mathcal{S}_s, \mathcal{S}_t, \Sigma \rangle,$$

*the schemas $\mathcal{S}_s$ and $\mathcal{S}_t$ are trivial (select all documents) and the set $\Sigma$ contains exactly one std.*

***Potential.*** By Lemma 6.2, we assume that in the regular schema mapping there are no source and target schemas and that there is only a single std

$$\pi_\alpha(\bar{x}), \eta(\bar{x}) \longrightarrow \pi_{\alpha'}(\bar{x}, \bar{y}), \eta'(\bar{x}, \bar{y}).$$

Let us write $\pi$ for the conjunction of queries $\pi_\alpha$ and $\eta$, likewise let us write $\pi'$ for the conjunction of queries $\pi_{\alpha'}$ and $\eta'$. These are both data-selecting queries. Let $X$ be the set of variables in the tuple $\bar{x}$, and let $X'$ be the set of variables in the tuple $\bar{x}\bar{y}$.

If $X$ is a set of variables, we use the name $X$-answer set for a (finite) set of tuples of data values indexed by variables from $X$. Since we assume that our data values are natural numbers, an $X$-answer set is a finite subset of $\mathbb{N}^X$. For a document, we use the name *source answer set* for the set of tuples selected by $\pi$ in the document (this is an $X$-answer set). Likewise we define the *target answer set*, using the query $\pi'$ (this is an $X'$-answer set). A document $T'$ is a solution to document $T$ if and only if the target answer set in $T'$, when the tuples are restricted to coordinates from $X$, includes the source answer set of $T$.

A *potential* $\mathcal{P}$ over a set of variables $X$ is a family of $X$-answer sets, i.e. a family of finite subsets of $\mathbb{N}^X$. If $\mathcal{P}$ is a potential over variables $X'$ and $X \subseteq X'$, then we write $\mathcal{P}|X$ for the potential where the tuples in the answer sets are restricted to $X$. If $\mathcal{P}$ and $\mathcal{Q}$ are potentials over the same variables, then we write $\mathcal{P} \leq \mathcal{Q}$ if every set of tuples in $\mathcal{P}$ is a subset of some set of tuples in $\mathcal{Q}$.

For our fixed std, define the *source potential* (call it $\mathcal{P}$) as the family of its source answer sets (ranging over all possible documents), likewise define the *target potential* (call it $\mathcal{Q}$). The absolute consistency problem is equivalent to deciding if

$$\mathcal{P} \leq \mathcal{Q}|X.$$

The solution existence problem can also be expressed using potentials. For a given source document $T$, there exists a solution to $T$ if and only if

$$\mathcal{P}_T \leq \mathcal{Q}|X$$

where $\mathcal{P}_T$ is the potential that contains only one set, namely the source answer set of $T$.

Our approach to both the absolute consistency and solution existence problem is as follows. First, we develop a language for describing potentials. We show that the $\leq$ relation is decidable for potentials presented in the language. Finally, we prove that the potentials $\mathcal{P}, \mathcal{Q}, \mathcal{P}_T$ can all be expressed in the language.

***Potential expressions.*** Fix a finite set $X$ of variables and a finite set $C$ of constants. A *potential expression* with variables $X$ and constants $C$ is an expression of the form

$$e = \mu_1 | \mu_2 | \ldots | \mu_n \,,$$

where each $\mu_i$ is a boolean combination of equalities over variables and constants. We assume that each variable and each constant is used at least once. An example of a potential expression over variables $\{x, y, z\}$ and constants $\{c\}$ is

$$x = c \wedge y = z | (x = y = z)$$

A potential expression $e$ defines a potential $\mathcal{P}_e$ over variables $X$ as follows. This potential contains an answer set $A \subseteq \mathbb{N}^X$ if and only if there is some $\mu_i \in \{\mu_1, \ldots, \mu_n\}$, some finite set $D \subseteq \mathbb{N}$ of data values, and some constant valuation $C \to D$ such that $A$ contains exactly the tuples in $D^X$ that satisfy $\mu_i$. The potential defined by expression in the example contains exactly the answer sets of the form

$$A_{D,c} = \{(c, d, d) : d \in D\}$$
$$B_D = \{(d, d, d) : d \in D\}$$

where $D$ is a finite set of data values and $c \in D$.

Our proof of Theorem 6.1 uses two lemmas. The first lemma says that the order $\leq$ is decidable on potentials given by expressions. The proof is similar to the one of Theorem 3.5.

LEMMA 6.3. *The following problem is decidable (in* $\Pi_4$P*): Given $e, f$, is $\mathcal{P}_e \leq \mathcal{P}_f$?*

The second lemma says that potential expressions can be computed for the potentials that appear when solving the absolute consistency and solution existence problems.

LEMMA 6.4. *Let $\pi_\alpha(\bar{x})$ be a data-selecting query obtained from an MSO query $\alpha$, and let $\eta(\bar{x})$ be a boolean combination of equalities on data values. Let $\mathcal{P}$ be the potential of the conjunction of $\pi_\alpha$ and $\eta$. One can compute a potential expression $e$ such that $\mathcal{P}_e \equiv \mathcal{P}$.*

The proof of Lemma 6.4 is long and included in the appendix available online [9]. Actually, our proof can be extended to show a more general result. Namely, for every potential $\mathcal{P}$ there exists a potential expression $e$ with $\mathcal{P}_e \equiv \mathcal{P}$ if and only if $\mathcal{P}$ is a potential such that for any answer set $A \in \mathcal{P}$ and any permutation of data values $\rho : \mathbb{N} \to \mathbb{N}$, the potential $\mathcal{P}$ contains also the answer set

$$\rho A = \{\rho \circ v : v \in A\}.$$

However, we do not need the more general result. On the other hand, we need to be able to compute the pattern expression (and not just know that it exists), and for this we use the assumptions of Proposition 6.4.

From the above two lemmas we get an algorithm for deciding the absolute consistency problem. First, we compute a potential expression $e$ that represents the source potential. Then, we compute a potential expression that represents the target potential. Finally, we test the order. For the solution existence problem, the approach is the same. However, instead of computing the source potential, we only use the subset of the source potential that contains a single answer set, namely the source answer set in the given document. Any potential that contains a single answer set is easily seen to be described by a potential expression, e.g. by using constants for all data values in the answer set. Finally, once the schema mapping is fixed, the solution existence problem is in P thanks to the following lemma.

LEMMA 6.5. *Fix a potential expression $e$ with variables $X$. The following problem is in P: for an $X$-answer set $A$, decide if there exists an $X$-answer set $B \in \mathcal{P}_e$ with $A \subseteq B$.*

# 7. FUTURE WORK

A summary of complexity results is shown in Fig. 3. Notably lacking are lower bounds for regular mappings. As we have mentioned, more specialized techniques are likely to bring algorithms with better complexity.

An open direction of more theoretical interest is to explore the limits of decidability. Which kinds of logic in stds guarantee decidability of absolute consistency? Or even more generally, which kinds of dependencies? We believe that our notion of potential can be helpful.

A fundamental concept in data exchange is a universal solution, i.e., a solution that can be mapped homomorphically into every other solution. Universal variants of the problems we have considered are worth exploring. On the other hand, homomorphisms of ordered trees are injective, which makes a universal solution a rare bird. Developing relaxed versions of universal solutions seems an interesting research topic.

*Acknowledgments.* The authors thank Leonid Libkin for inspiring discussions, and Alin Deutsch and the anonymous referees for helpful comments.

The first author was supported by the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599.

The second and the third author were supported by the *Querying and Managing Navigational Databases* project realized within the Homing Plus programme of the Foundation for Polish Science, cofinanced by the European Union from the Regional Development Fund within the Operational Programme Innovative Economy ("Grants for Innovation").

The second author was also supported by Polish Ministry of Science and Higher Education grant no. N N201 382234.

| Mappings | AbCons | SolEx | SolEx($\mathcal{M}$) |
|---|---|---|---|
| bounded-depth | $\Pi_4$P | $\Sigma_3$P | in LOGSPACE |
| pattern-based | $\Pi_2$EXP | NEXP | in LOGSPACE |
| regular | decidable | decidable | in P |

| Mappings | Solution building |
|---|---|
| fixed | in P |
| bounded-depth | in PSPACE |
| pattern-based | in EXPSPACE |

**Figure 3: Summary of complexity results**

# 8. REFERENCES

[1] S. Amano, L. Libkin, and F. Murlak. XML schema mappings. In *PODS 2009*, pages 33–42.
[2] S. Amer-Yahia, S. Cho, L. Lakshmanan, D. Srivastava. Tree pattern query minimization. *VLDB J.* 11 (2002), 315–331.
[3] M. Arenas and L. Libkin. XML data exchange: consistency and query answering. *JACM*, 55(2):7:1–72, 2008.
[4] M. Benedikt, W. Fan, F. Geerts. XPath satisfiability in the presence of DTDs. *J. ACM* 55(2): (2008).
[5] P. Barceló. Logical foundations of relational data exchange. *SIGMOD Record*, 38(1):49–58, 2009.
[6] P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD*, 2007.
[7] H. Björklund, W. Martens, T. Schwentick. Conjunctive query containment over trees. *DBPL'07*, pages 66–80.
[8] H. Björklund, W. Martens, T. Schwentick. Optimizing conjunctive queries over trees using schema information. *MFCS'08*, pages 132–143.
[9] M. Bojańczyk, L. Kołodziejczyk, F. Murlak. Solutions for XML Data Exchange. Available at `www.mimuw.edu.pl/~fmurlak/papers/bkm10.pdf`.
[10] C. David. Complexity of data tree patterns over XML documents. In *MFCS'08*, pages 278–289.
[11] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Date exchange: semantics and query answering. *TCS*, 336:89–124, 2005.
[12] R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Composing schema mappings: second-order dependencies to the rescue. *ACM TODS*, 30(4):994–1055, 2005.
[13] G. Gottlob, C. Koch, K. Schulz. Conjunctive queries over trees. *J.ACM* 53(2):238–272 (2006).
[14] J. Hidders. Satisfiability of XPath expressions. In *DBPL'03*, pages 21–36.
[15] P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In *PODS 2005*, pages 61–75.
[16] R. Miller, M. Hernandez, L. Haas, L. Yan, C. Ho, R. Fagin, and L. Popa. The Clio project: managing heterogeneity. *SIGMOD Record*, 30:78–83, 2001.
[17] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. ACM Transactions on Internet Technology, 5(4):1-45, 2005.
[18] F. Neven. Automata Theory for XML Researchers. *SIGMOD Record* 31(3): 39-46 (2002).