Containment of Pattern-Based Queries over Data Trees

Claire David
Université Paris-Est Marne-la-Vallée
claire.david@univ-mlv.fr

Amélie Gheerbrant
University of Edinburgh
agheerbr@inf.ed.ac.uk

Leonid Libkin
University of Edinburgh
libkin@inf.ed.ac.uk

Wim Martens
University of Bayreuth
wim.martens@uni-bayreuth.de

ABSTRACT

We study static analysis, in particular the containment problem, for analogs of conjunctive queries over XML documents. The problem has been studied for queries based on arbitrary patterns, not necessarily following the tree structure of documents. However, many applications force the syntactic shape of queries to be tree-like, as they are based on proper tree patterns. This renders previous results, crucially based on having non-tree-like features, inapplicable. Thus, we investigate static analysis of queries based on proper tree patterns. We go beyond simple navigational conjunctive queries in two ways: we look at unions and Boolean combinations of such queries as well and, crucially, all our queries handle data stored in documents, i.e., we deal with containment over data trees.

We start by giving a general Π_2^p upper bound on the containment of conjunctive queries and Boolean combinations for patterns that involve all types of navigation through documents. We then show matching hardness for conjunctive queries with all navigation, or their Boolean combinations with the simplest form of navigation. After that we look at cases when containment can be witnessed by homomorphisms of analogs of tableaux. These include conjunctive queries and their unions over child and next-sibling axes; however, we show that not all cases of containment can be witnessed by homomorphisms. We look at extending tree patterns used in queries in three possible ways: with wildcard, with schema information, and with data value comparisons. The first one is relatively harmless, the second one tends to increase complexity by an exponential, and the last one quickly leads to undecidability.

EDBT/ICDT '13, March 18 - 22 2013, Genoa, Italy.

Categories and Subject Descriptors

H.2.3 [**Database management**]: Languages—*Query Languages*; F.2 [**Analysis of algorithms and problem complexity**]: General

General Terms

Algorithms, Theory

1. INTRODUCTION

Static analysis of queries and specifications has been actively investigated in the context of XML, not only due to its importance in tasks such as query optimization but also due to a very different nature of the results brought to the fore by the hierarchical structure of XML documents [1, 3, 7, 10, 13, 14, 16, 17, 18, 19, 28, 30, 33, 35]. Typical reasoning problems include consistency of queries or constraints with respect to schema information, typechecking of transformations, security of views, and crucially, query containment, with or without schema information. The latter is the problem we deal with. Starting from the relational case, we know that query containment is often the technical core of many query optimization [15]. In recent years query containment found multiple applications not only in query answering and optimization, but also in data integration, exchange, and provenance among others [22, 23, 27].

Already in the relational case, we know that, by and large, containment is decidable for conjunctive queries and relatives, and undecidable for expressive queries, such as those coming from the full relational algebra. In the XML case, static analysis of queries has been largely restricted to queries in various fragments of XPath [35] and analogs of conjunctive queries [10, 11, 21], primarily describing the structure of documents. The latter classified the complexity of query containment depending on the list of used axes, providing a seemingly complete picture.

Nonetheless, the picture depicted by [10, 11, 21] is not as complete as it seems. Firstly, this work basically took *relational conjunctive queries* on top of XML documents, and considered containment problems for them. Queries like that in a way do not follow tree patterns. An example of such

a query is one saying that we have two nodes s and s', so that s' is a descendant of s, and we have other nodes $s_1, \ldots, s_n$ so that each s_i is a descendant of s and an ancestor of s'. This says that the s_is appear in some order on the unique path from s and s'. But note that the query itself is DAG-shaped rather than tree-shaped (i.e., if we consider its tableau, it is a DAG rather than a tree). Secondly, results in those papers mainly concentrated on navigational features and much less on the data that documents carry. For instance, containment of queries with data values remained practically unexplored (as these papers concentrated on satisfiability).

And yet many applications demand proper *XML conjunctive queries* which are tree-based and can return data. Such queries are naturally induced by *XML patterns* which appear in multiple applications including XML data exchange, data integration, and query optimization [4, 5, 6, 8, 9, 24]. Such patterns are given by grammars or formation rules that naturally induce a tree structure. As a simple example, we can say that a label a is a pattern, and if $\pi_1, \ldots, \pi_n$ are patterns, then $a[\pi_1, \ldots, \pi_n]$ is a pattern. It will be matched by an a-labeled node that has n (not necessarily distinct) children matching $\pi_1, \ldots, \pi_n$. Such patterns (and more complex ones including other axes) form the basis for describing XML schema mappings and incompleteness in XML documents. And yet they cannot generate the DAG-like "confluence" behavior explained earlier; because for any two nodes appearing on a descendant path, it is always specified which one appears first. But it is precisely this behavior that is behind many of the complexity results applied to graph-based pattern queries in XML.

So our question is: what are the costs of static analysis problems for proper *tree-based pattern queries*?

A key feature of the main applications of such patterns is that they are not purely about the *structure* of documents, but they also collect *data*. For instance, a pattern $a(x)[b(x), c(y)]$ collects values x and y of data so that a document has an a-node with value x, that in turn has a b-child with the same value x and a c-child with value y. Thus, queries return sets of tuples, rather just a yes/no answer for the existence of a purely structural match.

Furthermore, we do not look solely at analogs of conjunctive queries. In the relational case, it is well known that containment of both conjunctive queries and their unions has the same complexity, namely NP-complete [15, 34], and the complexity of containment of arbitrary Boolean combinations of conjunctive queries moves one level up in the polynomial hierarchy to Π_2^p-complete. So we deal with analogs of such queries too, built from tree-based patterns.

Thus, our revised goal is to investigate the containment problems for analogs of conjunctive queries and relatives (unions, Boolean combinations) based on tree-based XML patterns over both structural information and data that XML documents carry.

Overview of the results. As an abstraction of XML documents with data values we use, as is common, *data trees*. In those trees, each node carries both a label and a data value. This is also sufficient to model XML documents whose nodes may have multiple attributes, simply by creating an extra child of a node for each attribute name.

We start by defining tree patterns upon which our queries are based. The simplest are patterns based on child navigation. An example is the pattern $a(x)[b(x), c(y)]$ that we explained before. Note that variables corresponding to data values are free: we view this pattern as $\pi(x, y)$, returning all pairs (x, y) such that a match occurs with x and y being the data values witnessing it. We can add horizontal navigation too, for instance we can have a pattern $a(x)[b(x) \rightarrow c(y)]$, stating that the c-witness is the sibling that follows the b-witness. More generally, we can have transitive closure axes too: for instance $a(x)[c(x)]//[b(x) \rightarrow^* c(y)]$ says that the a-node has a c-child with the same data value and two descendants, with labels b and c and data values x and y, so that they are siblings and the b-node occurs earlier in the sibling order. These types of patterns occur, for instance, in integration and exchange tasks for defining XML schema mappings [4, 6, 8] or in descriptions of XML with incompleteness features [2, 9].

Based on such patterns, we define conjunctive queries (CQs) by closing them under conjunction and existential quantification, i.e., as queries $q(\bar{x}) = \exists \bar{y} \bigwedge_i \pi_i(\bar{x}, \bar{y})$. We look at unions of conjunctive queries, or UCQs, which are of the form $\bigcup_i q_i(\bar{x})$, where each $q_i(\bar{x})$ is a CQ, and their Boolean combinations, or BCCQs, obtained by applying operations $q \cap q'$, $q \cup q'$, and $q - q'$ to CQs.

We present a general upper bound showing that containment of BCCQs that use all the axes is in Π_2^p. We show two matching lower bounds: either for BCCQs with the simplest navigation (only child relation), or CQs with all the axes.

In the relational case, CQ containment is tested by tableaux homomorphisms. The Π_2^p-hardness for general CQs precludes the possibility of such a test in general, but we show that with restricted sets of axes it is still possible (in fact we just have to exclude the horizontal $\rightarrow^*$ relation).

We then look at adding features to patterns and queries. First, we add wildcard and show that the Π_2^p-upper bound continues to hold. We also show that, for some classes of queries, the complexity of containment can jump from NP-complete to Π_2^p-complete if wildcard is added to patterns. Furthermore, we identify a 'safe' case of using wildcard, namely everywhere except at roots of patterns, that preserves homomorphism characterizations of containment.

The next addition we consider is containment under schema information (abstracted as a tree automaton, which capture many schema formalisms for XML). Here we show that the upper bound increases to double-exponential, and a matching lower bound can be shown for CQs with all axes. Finally, we look at adding data-value comparisons to queries, in particular the disequality ($\neq$) comparisons. This addition has a much more dramatic effect on the complexity of containment: it becomes undecidable for BCCQs, and for CQs when both comparisons and schemas are present (even with severe restrictions on available navigation).

Comparison with non-tree pattern queries. As we already mentioned, a number of results exist on CQs based on graph-shaped, rather than tree-shaped patterns [10, 11, 21]. None of those results extend to handle unions and differences of queries (i.e., UCQs and BCCQs) and they handle data values in a very limited way. Below we contrast them with our results.

For CQs that may contain arbitrary graph patterns, the Π_2^p upper bound continues to hold, but hardness requires less: for instance, purely navigational queries with non-tree patterns are already Π_2^p-complete for vertical navigation [11] (for tree patterns they stay in NP, as we show). Under schema, containment of CQs jumps to doubly-exponential too [10]. With $\neq$ comparisons, even CQ containment becomes undecidable [10].

Those results cover only a part of the landscape that we study here (i.e., CQs, concentrating mainly on pure navigation), and, crucially, under different assumptions on the shape of queries. Such assumptions make most existing proofs inapplicable for us (as they often rely heavily on non-tree features, such as the confluence, explained earlier, and the bidirectionality of axes).

Organization. We give key definitions in Section 2. The Π_2^p upper bound for BCCQs is shown in Section 3. In Section 4 we prove two matching lower bounds. In Section 5 we investigate cases when containment can be witnessed by the homomorphism of tableaux. Section 6 studies the effect of adding wildcard to queries. In Section 7 we investigate static analysis under schema constraints. Adding data-value comparison is studied in Section 8. Concluding remarks are given in Section 9. Due to space limitations, proofs are only sketched.

2. TREES, PATTERNS, AND QUERIES

Unranked trees and data trees. We start with the standard definitions of unranked finite trees which serve as an abstraction of XML documents when one deals with their structural properties. A finite unranked tree domain is a non-empty, prefix-closed finite subset D of $\mathbb{N}^*$ (words over $\mathbb{N}$) such that $s \cdot i \in D$ implies $s \cdot j \in D$ for all $j < i$ and $s \in \mathbb{N}^*$. We refer to elements of finite unranked tree domains as *nodes*. We assume a countably infinite set $\mathcal{L}$ of possible labels that can be used to label tree nodes. An unranked tree is a structure $\langle D, \downarrow, \rightarrow, \lambda \rangle$, where

- D is a finite unranked tree domain,
- $\downarrow$ is the child relation: $s \downarrow s \cdot i$ for $s \cdot i \in D$,
- $\rightarrow$ is the next-sibling relation: $s \cdot i \rightarrow s \cdot (i+1)$ for $s \cdot (i+1) \in D$, and
- $\lambda : D \rightarrow \mathcal{L}$ is the labeling function assigning a label to each node.

We denote the reflexive-transitive closure of $\downarrow$ by $\downarrow^*$ (descendant-or-self), and the reflexive-transitive closure of $\rightarrow$ by $\rightarrow^*$ (following-sibling-or-self).

Data trees are a standard abstraction of XML documents when one deals with both structural properties and data. Suppose we have a domain $\mathcal{D}$ of *data values*, such as strings, numbers, etc. A *data tree* is a structure $t = \langle D, \downarrow, \rightarrow, \lambda, \rho \rangle$, where $\langle D, \downarrow, \rightarrow, \lambda \rangle$ is an unranked tree, and $\rho : D \rightarrow \mathcal{D}$ assigns each node a data value. In XML documents, nodes may have multiple attributes, but this is easily modeled with data trees. For instance, to model a node with attributes $a_1, \ldots, a_n$ having values $v_1, \ldots, v_n$, we pick special labels $\ell_1, \ldots, \ell_n$, and create n extra children labeled $\ell_1, \ldots, \ell_n$ carrying values $v_1, \ldots, v_n$.

Patterns. As already explained, the patterns that we use are naturally tree-shaped. To explain how they are introduced, let us consider the reducts of data trees to the child relation, i.e., structures $\langle D, \downarrow, \lambda, \rho \rangle$. Trees of this form can be defined by recursion. That is, a node labeled with $a \in \mathcal{L}$ and carrying a data value $v \in \mathcal{D}$ is a data tree, and if $t_1, \ldots, t_n$ are trees, we can form a new tree by making them children of a node with label a and data value v.

In patterns we use also variables; the intention for them is to match data values in data trees. Thus, they are essentially partial tree descriptions with variables appearing in place of some data values. We assume a countable infinite set $\mathcal{V}$ of variables, disjoint from the domain of values $\mathcal{D}$. So the previous inductive definition gives rise to the definition of the simplest patterns we consider here:

$$\pi := a(x)[\pi, \ldots, \pi] \tag{1}$$

with $a \in \mathcal{L}$ and $x \in \mathcal{V} \cup \mathcal{D}$. Here the sequence in $[\ldots]$ could be empty. In other words, if $\pi_1, \ldots, \pi_n$ is a sequence of patterns (perhaps empty), $a \in \mathcal{L}$ and $x \in \mathcal{V} \cup \mathcal{D}$, then $a(x)[\pi_1, \ldots, \pi_n]$ is a pattern. If $\bar{x}$ is the list of all the variables used in a pattern π, we write $\pi(\bar{x})$.

We denote patterns from this class by $\mathbf{\Pi}(\downarrow)$. The semantics of $\pi(\bar{x})$ is defined with respect to a data tree $t = \langle D, \downarrow, \rightarrow, \lambda, \rho \rangle$, a node $s \in D$, and a valuation $\nu : \bar{x} \rightarrow \mathcal{D}$ as follows: $(t, s, \nu) \models a(x)[\pi_1(\bar{x}_1), \ldots, \pi_n(\bar{x}_n)]$ iff

- $\lambda(s) = a$ (the label of s is a);
- $\rho(s) = \begin{cases} \nu(x) & \text{if } x \text{ is a variable} \\ x & \text{if } x \text{ is a data value;} \end{cases}$
- there exist not necessarily distinct children $s \cdot i_1, \ldots, s \cdot i_n$ of s so that $(t, s \cdot i_j, \nu) \models \pi_j(\bar{x}_j)$ for each $j \leq n$ (recall that n could be 0, in which case this last item is not needed).

We write $(t, \nu) \models \pi(\bar{x})$ if there is a node s so that $(t, s, \nu) \models \pi(\bar{x})$ (i.e., a pattern is matched somewhere in the tree). Also if $\bar{v} = \nu(\bar{x})$, we write $t \models \pi(\bar{v})$ instead of $(t, \nu) \models \pi(\bar{x})$.

A natural extension for these simple patterns is to include both vertical and horizontal navigation. Again the intuition comes from defining data trees as follows: a node labeled with $a \in \mathcal{L}$ and carrying a data value $v \in \mathcal{D}$ is a data tree, and if $t_1, \ldots, t_n$ are trees, we can form a new tree by making them children of a node with label a and data value v, so that their roots are connected in the order $t_1 \rightarrow t_2 \rightarrow \ldots \rightarrow t_n$.

This leads to the definition of patterns in the class $\mathbf{\Pi}(\downarrow, \rightarrow)$:

$$\pi := a(x)[\pi \rightarrow \ldots \rightarrow \pi] \tag{2}$$

with $a \in \mathcal{L}$ and $x \in \mathcal{V} \cup \mathcal{D}$. Again the sequence in $[\ldots]$ could be empty. In other words, if $\pi_1, \ldots, \pi_n$ is a sequence of patterns (perhaps empty), $a \in \mathcal{L}$ and $x \in \mathcal{V} \cup \mathcal{D}$, then $a(x)[\pi_1 \rightarrow \pi_2 \rightarrow \ldots \rightarrow \pi_n]$ is a pattern. The last clause in the definition of the semantics of $\mathbf{\Pi}(\downarrow)$ is modified as follows:

- there exists a child $s \cdot i$ of s so that $(t, s \cdot i, \nu) \models \pi_1(\bar{x}_j)$, $(t, s \cdot (i+1), \nu) \models \pi_2(\bar{x}_2)$, ..., $(t, s \cdot (i+n-1), \nu) \models \pi_n(\bar{x}_n)$. In other words, it is consecutive children that witness the satisfaction of subpatterns.

Patterns in $\mathbf{\Pi}(\downarrow)$ and $\mathbf{\Pi}(\downarrow, \rightarrow)$ completely specify the structure of a tree (depending on the available axes) and, in particular, only express local properties of trees. We therefore also consider their more expressive versions with transitive closure axes $\downarrow^*$ (descendant) and $\rightarrow^*$ (following sibling). More precisely, following [4, 20], we define general patterns by the rules:

$$\begin{array}{lcl} \pi & := & a(x)[\mu, \ldots, \mu]//[\mu, \ldots, \mu] \\ \mu & := & \pi \rightsquigarrow \ldots \rightsquigarrow \pi \end{array} \tag{3}$$

Here a, x and π are as before, and μ stands for a *sequence*, i.e., a forest such that the roots of its trees are sequential siblings in a tree, and each $\rightsquigarrow$ is either $\rightarrow$ or $\rightarrow^*$.

The class of such patterns is denoted by $\mathbf{\Pi}(\Downarrow, \Rightarrow)$, with $\Downarrow$ we use both types of downward navigation ($\downarrow$ and $\downarrow^*$) and $\Rightarrow$ meaning that we use both types of horizontal navigation ($\rightarrow$ and $\rightarrow^*$). The semantics is extended as follows.

- $(t, s, \nu) \models \pi_1 \rightsquigarrow \ldots \rightsquigarrow \pi_m$ if there is a sequence $s = s_1, \ldots, s_m$ of nodes so that $(t, s_i, \nu) \models \pi_i$ for each $i \leq m$ and $s_i \rightarrow s_{i+1}$ whenever the ith $\rightsquigarrow$ is $\rightarrow$, and $s_i \rightarrow^* s_{i+1}$ whenever the ith $\rightsquigarrow$ is $\rightarrow^*$.
- $(t, s, \nu) \models a(x)[\mu_1, \ldots, \mu_n]//[\mu'_1, \ldots, \mu'_k]$ if the satisfaction of $a(x)$ in node s is as before, and there exist n not necessarily distinct children $s_1, \ldots, s_n$ of s such that $(t, s_i, \nu) \models \mu_i$ for each $i \leq n$, and there exist k not necessarily distinct descendants $s'_1, \ldots, s'_k$ of s such that $(t, s'_i, \nu) \models \mu'_i$ for each $i \leq k$.

Notice that the semantics of patterns allows different μ_i to be mapped into the same nodes in a tree.

Finally, we consider a class $\Pi(\Downarrow)$ of patterns which is a restriction of the most general patterns to downward navigation only. These are defined by the grammar

$$\pi := a(x)[\pi, \ldots, \pi]//[\pi, \ldots, \pi] \tag{4}$$

where each of the sequences of patterns can be empty. That is, a pattern $a(x)[\pi_1, \ldots, \pi_n]//[\pi'_1, \ldots, \pi'_k]$ is witnessed in an a-labeled node assigning its data value to x if it has n children (not necessarily distinct) witnessing $\pi_1, \ldots, \pi_n$ and k descendants (again not necessarily distinct) witnessing $\pi'_1, \ldots, \pi'_k$.

Shorthands. We shall be using standard shorthand notations: $a(x)/\pi$ stands for $a(x)[\pi]$, while $a(x)//\pi$ denotes $a(x)//[\pi]$, and $a(x)/\pi//\pi'$ stands for $a(x)[\pi]//[\pi']$.

Conjunctive queries, their unions, and Boolean combinations. Pattern-based conjunctive XML queries are obtained by closing patterns by conjunction and existential quantification. Since we have different classes of patterns $\mathbf{\Pi}(\sigma)$, for σ being $\downarrow$, or $\downarrow, \rightarrow$, or $\Downarrow$, or $\Downarrow, \Rightarrow$, we have different classes of conjunctive queries denoted by $\mathrm{CQ}(\sigma)$. More precisely, $\mathrm{CQ}(\sigma)$ queries are of the form:

$$q(\bar{x}) \;=\; \exists \bar{y} \bigwedge_{i=1}^{n} \pi_i(\bar{z}_i) \tag{5}$$

where each π_i is a $\mathbf{\Pi}(\sigma)$ pattern, and each $\bar{z}_i$ is contained in $\bar{x}, \bar{y}$. The semantics is standard: $(t, \nu) \models q(\bar{x})$ if there is an extension ν' of valuation ν to variables $\bar{y}$ such that $(t, \nu') \models \pi_i(\bar{z}_i)$ for every $i \leq n$.

As is standard, we also write $t \models q(\bar{v})$ if $(t, \nu) \models q(\bar{x})$ with $\nu(\bar{x}) = \bar{v}$.

Of course conjunctive queries are closed under conjunction. Standard ways of enriching their power include considering unions of conjunctive queries (or UCQs, which, in the relational case, capture the positive fragment of relational algebra) and more generally, Boolean combinations of conjunctive queries (or BCCQs, which, while possessing some form of negation, still retain many nice properties that relational algebra as a whole loses).

Formally, a query from $\mathrm{UCQ}(\sigma)$ is of the form $q(\bar{x}) = q_1(\bar{x}) \cup \ldots \cup q_m(\bar{x})$, where each $q_i(\bar{x})$ is a $\mathrm{CQ}(\sigma)$ query. It returns the union of answers to the q_i's, i.e., $(t, \nu) \models q(\bar{x})$ iff $(t, \nu) \models q_i(\bar{x})$ for some $i \leq m$.

Queries in the class BCCQ are obtained as follows: take some queries $q_1(\bar{x}), \ldots, q_m(\bar{x})$ from $\mathrm{CQ}(\sigma)$ and consider a Boolean combination of them, i.e., close them under operations $q \cap q'$, $q \cup q'$, and $q - q'$. The semantics is extended naturally, with those interpreted as intersection, union, and set difference, respectively.

The answer to a query $q(\bar{x})$, from any of the above classes, on a data tree t is defined as $q(t) = \{\nu(\bar{x}) \mid (t, \nu) \models q(\bar{x})\}$. Note that our definitions of query classes ensure that $q(t)$ is always finite.

Containment. The main problem we study here is the containment problem. Given two queries $q(\bar{x}), q'(\bar{x}')$ with tuples of free variables of the same length, we write $q \subseteq q'$ iff $q(t) \subseteq q'(t)$ for every data tree t. So the problem we look at is the following.

PROBLEM:	$\mathrm{CQ}_{\subseteq}(\sigma)$
INPUT:	queries $q(\bar{x}), q'(\bar{x}')$ in $\mathrm{CQ}(\sigma)$;
QUESTION:	is $q \subseteq q'$?

If instead of queries in $\mathrm{CQ}(\sigma)$ we use queries in $\mathrm{UCQ}(\sigma)$, we refer to the problem $\mathrm{UCQ}_{\subseteq}(\sigma)$ and, if we use queries from $\mathrm{BCCQ}(\sigma)$, we refer to the problem $\mathrm{BCCQ}_{\subseteq}(\sigma)$.

In the *relational* case, these problems are among the basic problems of database theory. The complexity of $\mathrm{CQ}_{\subseteq}$ and $\mathrm{UCQ}_{\subseteq}$ over relational databases is NP-complete [15, 34] (under the representation of UCQs that we use here), and the complexity of $\mathrm{BCCQ}_{\subseteq}$ is Π_2^p-complete [34].

3. AN UPPER BOUND

A priori, there is no upper bound that is immediate for the containment problem. In fact, in the presence of negation (even a limited form of it) combined with XML hierarchical structure, some reasoning problems can become undecidable (see, e.g., [17, 4]). In the relational case, we know that containment for BCCQs is Π_2^p-complete, but this does not imply the same bounds for XML pattern-based queries, especially those that might use transitive closure axes $\rightarrow^*$ and $\downarrow^*$.

Nevertheless, we can show that for all such queries, the containment problem remains not only decidable, but the upper bound on its complexity continues to match that for the simplest relational queries. In fact we show the following.

THEOREM 3.1. *The problem* $\mathrm{BCCQ}_{\subseteq}(\Downarrow, \Rightarrow)$ *is decidable in* Π_2^p.

In other words, for each of the classes of queries — CQ, UCQ, BCCQ— and for each of the classes of patterns seen so far, the containment problem is in Π_2^p, as all of these problems are subsumed by the containment problem of BCCQs with $\mathbf{\Pi}(\Downarrow, \Rightarrow)$-patterns.

Proof sketch. Checking whether $q_1 \subseteq q_2$ is the same as checking $q_1 - q_2 = \emptyset$, so it will suffice to give a Σ_2^p algorithm for checking if a BCCQ q returns a nonempty result on some data tree. We assume that q is a Boolean combination of CQs $q_1, \ldots, q_m$. For the sketch we assume they are Boolean (free variables do not change anything). To check satisfiability it suffices to guess an assignment $\chi : \{1, \ldots, m\} \rightarrow \{0, 1\}$ so that for

$$q' = \bigwedge\{q_i \mid \chi(i) = 1\} \text{ and } q'' = \bigvee\{q_j \mid \chi(j) = 0\}$$

we have a tree t such that $q'(t)$ is true and $q''(t)$ is false. Note that q' is a CQ, and q'' is a UCQ. The idea of the proof is to turn this into a certain answer problem in XML data exchange [8]. We let schemas of XML documents be arbitrary and the mapping consist of a single rule $\_ \rightarrow q'$, forcing the patterns of q' in every target tree. Then we check whether the certain answer to q'' is false: this happens iff there is a tree satisfying q' and the negation of q''.

The latter requires two steps in the proof. One is a modification of the proof of the coNP *data* complexity of certain answers in [8]. The problem is that the latter proof produces a witnessing tree whose size is exponential in q'', which is too large for our purposes. So we show how to encode the exponential witness by a data structure whose size is polynomial in q', q'' and which allows checking for satisfiability of UCQs. The second step is making sure that all the guesses are combined in the right order to yield a Σ_2^p algorithm. □

4. LOWER BOUNDS FOR CONTAINMENT

Now that we know that all the containment problems are in Π_2^p, it is natural to ask when we have matching lower bounds. Note that in all the variations of containment problems, we have two parameters: the class of queries (going from the simplest, CQs, to UCQs, and to BCCQs), and the set of axes (again, starting with the simplest, just $\downarrow$, and then going to more complex $\downarrow, \rightarrow$, as well as $\Downarrow$ and $\Downarrow, \Rightarrow$).

What we show in this section is that each of the combination simplest/hardest leads to Π_2^p-hardness. That is, the containment problem with the simplest of axes, just $\downarrow$, is Π_2^p-complete if we allow Boolean combinations of queries. If we have just CQs, the containment becomes Π_2^p-complete when we have all the axes, i.e. $\downarrow, \downarrow^*, \rightarrow$, and $\rightarrow^*$.

Note that the first result on the surface is rather similar to Π_2^p-completeness of containment of relational BCCQs [34]. Indeed, the standard representation of relations in XML only needs the $\downarrow$ axis, and shallow documents. However, the result does not follow from the results in [34], as we demand containment over all XML documents, not only those that properly represent relational databases of a given schema. In particular, if we have two relational BCCQs q and q', and their natural XML codings as $\mathrm{BCCQ}(\downarrow)$ queries q_{XML} and q'_{XML}, then $q_{\text{XML}} \subseteq q'_{\text{XML}}$ implies $q \subseteq q'$ (as each relational database can be coded as an XML tree), but under the same coding $q \subseteq q'$ need not imply $q_{\text{XML}} \subseteq q'_{\text{XML}}$.

Even though we cannot use results on [34], we can modify reductions to apply to all XML documents and obtain the following.

THEOREM 4.1. *The problem* $\mathrm{BCCQ}_{\subseteq}(\downarrow)$ *is* Π_2^p*-complete.*

Next, we move to the other extreme case: CQs with all the axes. Of course relational containment of CQs is NP-complete, so to get hardness for a larger class, one has to use, in an essential way, the hierarchical structure of XML. In fact we provide a rather elaborate reduction showing that the navigational abilities of all the axes are sufficient to increase the complexity even of conjunctive query containment.

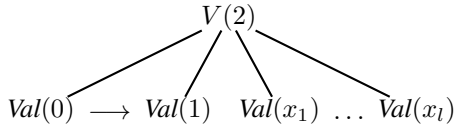
THEOREM 4.2. *The problem* $CQ_{\subseteq}(\Downarrow, \Rightarrow)$ *is* Π_2^p*-complete.*

Proof sketch. The upper bound was shown in the previous section. To show hardness, we proceed by reduction from $\forall\exists$3CNF. Given such a formula $\varphi := \forall p_1 \ldots \forall p_l \exists r_1 \ldots \exists r_m \bigwedge_i (\ell_{i1} \vee \ell_{i2} \vee \ell_{i3})$, where the ℓ_{ij}s could be positive or negative literals, we associate with it two Boolean queries $q, q' \in \mathrm{CQ}(\Downarrow, \Rightarrow)$ such that φ is true if and only if $q \subseteq q'$.

We construct q and q' so that for every possible valuation v of the p_is, two conditions hold. First, there exists a tree t_v satisfying q which encodes v. Second, such a tree t_v satisfies q' iff there is a valuation v^+ extending v to the r_is and for

which φ evaluates to true. The key idea behind the construction is encoding possible valuations for quantified variables, and we explain it now. The encoding of the CNF formula itself is standard.

In order to encode every possible valuation of the p_is using one single query q, we associate a variable x_i to each p_i and then take full advantage of navigational features to model assignments. Specifically, we use a tree pattern $V(2)/[\mathit{Val}(0) \to \mathit{Val}(1), \mathit{Val}(x_1), \ldots, \mathit{Val}(x_l)]$. Its root has $l+2$ children, among which the ordering is specified for two ($\mathit{Val}(0) \to \mathit{Val}(1)$). The remaining l children carry the x_is, but note that their exact positions as children of the $V(2)$ node are not specified. This is illustrated below:



Now on every complete tree t witnessing this pattern via some homomorphism h, the image of every x_i will either be on the left, or on the right of 0, i.e., either

$$t \models V(2)[\mathit{Val}(h(x_i)) \to^* \mathit{Val}(0)],$$

or

$$t \models V(2)[\mathit{Val}(0) \to^* \mathit{Val}(h(x_i))].$$

This allows us to associate a valuation v of the p_is to any tree satisfying this pattern by letting $v(p_i)$ be false if the image of x_i occurs on the left of $\mathit{Val}(0)$, and by letting $v(p_i)$ be true otherwise. The rest of the encoding consists of the standard encoding of a CNF formula, and ensuring, for q', that the extended valuation makes that formula true. □

Remark Note that letting one omit a complete specification of the sibling ordering has the effect of encoding 2^n possible valuations with n different nodes. This is similar to the effect of using "confluence" features in [11]. In both cases, such a concise encoding of exponentially many valuations led to Π_2^p lower bounds.

5. CONTAINMENT VIA HOMOMORPHISMS

A classical result of relational database theory says that containment of relational CQs is NP-complete and containment is witnessed by the existence of a *homomorphism* of tableaux: if T_i is the tableau of a query q_i, for $i = 1, 2$, then $q_1 \subseteq q_2$ iff there is a homomorphism from T_2 to T_1 [15]. However, the results of the previous section indicate that such a characterization of containment via homomorphisms cannot be extended to all classes of CQs we consider here. Indeed, testing for the existence of a homomorphism is a classical NP-complete problem and we saw in Theorem 4.2 that containment of CQ($\Downarrow, \Rightarrow$) queries is Π_2^p-complete.

So the question is: for what types of queries, if any, can we characterize containments via homomorphisms of their tableaux? And even before answering this question, we need to ask: what are the tableaux of XML-based CQs?

Since tableaux for relational queries are essentially incomplete databases (more precisely, naïve tables with a distinguished row of variables), it is natural to define tableaux of XML CQs as incomplete XML trees. Indeed, patterns forming a query are essentially incompletely specified trees, so we can view each query as an incomplete tree (more precisely, a forest). The theory of incompleteness of XML has been developed [2, 9] and thus we can borrow a notion of an incomplete tree.

Incomplete trees and homomorphism. An *incomplete tree* is defined as a structure $t = (N, V, \downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \lambda, \rho)$, where

- N and V are disjoint finite sets of the nodes of t and its data values, respectively; we assume that $V \subset \mathcal{D} \cup \mathcal{V}$, i.e., values could be either data values or variables;
- all of $\downarrow, \downarrow^*, \rightarrow, \rightarrow^*$ are binary relations on N;
- λ is a partial function from N to $\mathcal{L}$; and
- ρ is a function from N to V.

Note that in an incomplete tree, the relations $\downarrow, \downarrow^*, \rightarrow, \rightarrow^*$ may be interpreted arbitrarily. In particular, some incomplete trees cannot be extended to a complete tree. The issue is discussed in details in [9]. The labeling function is partial, reflecting the fact that labels of some nodes may not be known. The data assigning function ρ is not partial since some data values could be variables, just like in patterns.

Given two incomplete trees $t = \langle N, V, \downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \lambda, \rho\rangle$ and $t' = \langle N', V', \downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \lambda', \rho'\rangle$, a *homomorphism* from t to t' is a map $h : N \cup V \to N' \cup V'$ such that:

- $h(N) \subseteq N'$ and $h(V) \subseteq V'$;
- if wRw' in t, with $w, w' \in N$ and R one of the relations $\downarrow, \downarrow^*, \rightarrow, \rightarrow^*$, then $h(w)Rh(w')$ in t';
- if $\lambda(w)$ is defined in t, then $\lambda'(h(w)) = \lambda(w)$;
- h is the identity on elements of $\mathcal{D}$; and
- $h(\rho(w)) = \rho'(h(w))$ for all $w \in N$.

Note that each tree can be viewed as an incomplete tree (with the natural interpretations of the binary relations) and thus it makes sense to speak of a homomorphism from an incomplete tree to a complete tree.

Our plan is now as follows. We show how to associate, to a CQ q, an incomplete tree t_q. If q is a Boolean query, then $t \models q$ iff there is a homomorphism from t_q into t. If q has free variables $\bar{x}$, then $t \models q(\bar{v})$ iff there is a homomorphism from $t_{q(\bar{x})}$ to t that sends $\bar{x}$ to $\bar{v}$.

We then show that, for some classes σ of axes and queries $q, q' \in \mathrm{CQ}(\sigma)$, we have $q \subseteq q'$ iff there is a homomorphism from the σ-restriction of $t_{q'}$ to the σ-restriction of t_q.

Incomplete trees of CQs. We now define analogs of tableaux of relational CQs; these will be incomplete trees.

We first define an incomplete tree t_π for each pattern π. To carry the inductive construction, we shall need to define both trees t_π and t_μ for sequences μ. Note that even though we use the name 'incomplete tree', such a structure need not be a tree (due to incompleteness); in fact t_μs will be forest-like. Each incomplete tree t of the form t_π or t_μ will have a set $\text{RT}(t)$ of roots associated with it in such a way that $\text{RT}(t_\pi)$ is always a singleton. The inductive construction is as follows.

- If $\pi = a(x)$, then $t_\pi = \langle \{s\}, \{x\}, \downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \lambda, \rho \rangle$, where s is a single node, all the binary relations are empty, $\lambda(s) = a$ and $\rho(s) = x$. Furthermore, $\text{RT}(t_\pi) = \{s\}$.
- Let $\pi = a(x)[\mu_1, \ldots, \mu_n]//[\mu'_1, \ldots, \mu'_k]$. Suppose we already have t_{μ_i}s and $t_{\mu'_j}$s defined. Let N_i and V_i be the sets of nodes and values in t_{μ_i}s and N'_j and V'_j be the sets of nodes and values in $t_{\mu'_j}$s. By renaming nodes in those incomplete trees, we may assume that all the sets N_is and N'_js are disjoint. Then
$$t_\pi = (N, V, \downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \lambda, \rho)$$
where $N = \{s\} \cup \bigcup_i N_i \cup \bigcup_j N'_j$, with s being a new node, and $V = \bigcup_i V_i \cup \bigcup_j V'_j$. The binary relations are the unions of those relations in the t_{μ_i}s and $t_{\mu'_j}$s. In addition, we put:
  - $s \downarrow s'$ for each $s' \in \text{RT}(\mu_i)$, for $i \leq n$; and
  - $s \downarrow^* s'$ for each $s' \in \text{RT}(\mu'_j)$, for $j \leq k$.

  The functions λ and ρ are the same as in the t_{μ_i}s and $t_{\mu'_j}$s; in addition $\lambda(s) = a$ and $\rho(s) = x$. Furthermore, $\text{RT}(t_\pi) = \{s\}$.
- Let $\mu = \pi_1 \rightsquigarrow \ldots \rightsquigarrow \pi_n$. Let t_{π_i} be an incomplete tree $\langle N_i, V_i, \downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \lambda_i, \rho_i \rangle$. As before, assume that by renaming nodes, all the N_is are disjoint. Let $\text{RT}(t_{\pi_i}) = \{s_i\}$.

  Then $t_\mu = \langle N, V, \downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \lambda, \rho \rangle$, where $N = \bigcup_i N_i$ and $V = \bigcup_i V_i$; the binary relations are unions of those in the t_{π_i}s, and in addition we put:
  - $s_i \rightarrow s_{i+1}$ if μ contains $\pi_i \rightarrow \pi_{i+1}$; and
  - $s_i \rightarrow^* s_{i+1}$ if μ contains $\pi_i \rightarrow^* \pi_{i+1}$.

  The functions λ and ρ coincide with λ_i and ρ_i on N_i. Moreover, $\text{RT}(\mu) = \{s_1, \ldots, s_n\}$.

With a query
$$q(\bar{x}) = \exists \bar{y_1} \ldots \exists \bar{y_n} \pi_1(\bar{x}, \bar{y_1}) \wedge \ldots \wedge \pi_n(\bar{x}, \bar{y_n})$$
we associate an incomplete data tree
$$t_q = (N, V, \downarrow, \downarrow^*, \rightarrow, \rightarrow^*, \lambda, \rho)$$
which is the node-disjoint union of all the t_{π_i}s; that is, we rename nodes so that their sets are disjoint (but not the values), and take the union of structures $t_{\pi_1}, \ldots, t_{\pi_n}$.

The incomplete trees t_q indeed play the role of tableaux of CQs. Recall that in the relational case, we have $D \models q(\bar{v})$ iff there is a homomorphism from the tableau of $q(\bar{x})$ to D that sends $\bar{x}$ to $\bar{v}$. The same is true here. The result is very similar to one in [9], adapted to the definitions given here.

PROPOSITION 5.1. *Let t be a data tree, and $q(\bar{x})$ a query from CQ$(\Downarrow, \Rightarrow)$. Then $t \models q(\bar{v})$ iff there is a homomorphism $h : t_q \rightarrow t$ so that $h(\bar{x}) = \bar{v}$.*

Containment and homomorphisms. We already mentioned that a classical result of relational database theory states that relational CQ containment $q \subseteq q'$ holds iff the tableau of q' can be homomorphically mapped into the tableau of q. Furthermore, an analog of this cannot possibly hold for queries in CQ$(\Downarrow, \Rightarrow)$ unless some complexity classes collapse. Nonetheless, it will work for queries that do not use all the axes.

Suppose we have a query q from CQ$(\downarrow)$. Then its incomplete tree t_q records no information about $\downarrow^*, \rightarrow$, and $\rightarrow^*$. So for two such queries q and q', a homomorphism of the $\downarrow$-reducts of t_q and $t_{q'}$ (that only keep information about $\downarrow, \lambda$, and ρ) is the same as a homomorphism t_q and $t_{q'}$. Hence, even for queries that use reduced sets of axes, e.g., CQ$(\downarrow)$ or CQ$(\downarrow, \rightarrow)$, we can still meaningfully talk about homomorphisms of their incomplete trees, in place of homomorphisms of reducts of incomplete trees.

We next show that without transitive closure axes, we have an analog of relational containment.

THEOREM 5.2. *Let $q(\bar{x})$ and $q'(\bar{x}')$ be two queries from either CQ$(\downarrow)$, or CQ$(\downarrow, \rightarrow)$. Then $q \subseteq q'$ iff there is a homomorphism $h : t_{q'} \rightarrow t_q$ so that $h(\bar{x}') = \bar{x}$.*

Since testing homomorphism existence is done in NP, and NP-hardness bound for relational CQs trivially applies to CQ$(\downarrow)$ queries, we obtain the following.

COROLLARY 5.3. *The containment problems for CQ$(\downarrow)$ and CQ$(\downarrow, \rightarrow)$, i.e., CQ$_\subseteq(\downarrow)$ and CQ$_\subseteq(\downarrow, \rightarrow)$, are* NP*-complete.*

In fact, we prove an even more general result, that shows the applicability of the homomorphism technique to queries in CQ$(\Downarrow, \rightarrow)$, i.e., queries using all forms of vertical navigation, but only the next-sibling form of horizontal navigation. Formally, they are CQs based on patterns from $\mathbf{\Pi}(\Downarrow, \rightarrow)$ defined as
$$\begin{array}{lcl} \pi & := & a(x)[\mu, \ldots, \mu]//[\mu, \ldots, \mu] \\ \mu & := & \pi \rightarrow \ldots \rightarrow \pi \end{array} \quad (6)$$
That is, they extend $\mathbf{\Pi}(\downarrow, \rightarrow)$ patterns by allowing descendants, and prohibiting only $\rightarrow^*$.

Given a query $q \in$ CQ$(\Downarrow, \rightarrow)$, we define an incomplete tree $(t_q)^*$ by replacing the interpretation of $\downarrow^*$ in t_q by the reflexive-transitive closure of the union of $\downarrow$ and $\downarrow^*$ in t_q. Then containment can be tested by the existence of homomorphisms between such extended tableaux. As an example, consider queries $q \subseteq q'$, where $q = \exists x\, a(x)//b(x)[c(x)]$ and $q' = \exists x\, a(x)//c(x)$. While there is no homomorphism from

$t_{q'}$ to t_q, there is one from $(t_{q'})^*$ to $(t_q)^*$. Indeed, in both structures there is a descendant axis going from the a-labeled node to the c-labeled node.

THEOREM 5.4. *Let $q(\bar{x})$ and $q'(\bar{x}')$ be two queries from CQ$(\Downarrow, \rightarrow)$. Then $q \subseteq q'$ iff there is a homomorphism $h : (t_{q'})^* \rightarrow (t_q)^*$ so that $h(\bar{x}') = \bar{x}$.*

Proof sketch. The right to left direction of the equivalence is immediate. To show the other direction, we assume $q \subseteq q'$ and we turn the forest t_q into some "canonical" complete tree T such that there is a natural one to one homomorphism $h_1 : (t_q)^* \rightarrow T$ and such that for all $w, w' \in (t_q)^*$, for all $R \in \{\rightarrow, \downarrow, \downarrow^*\}$, we have wRw' iff $h_1(w)Rh_1(w')$. To this end, we create new nodes labeled with a fresh label $\heartsuit$ and a fresh data value $\sharp$. One of these nodes becomes the common parent of each of the roots of the tree patterns in t_q and thus becomes the root of T. We also define a recursive procedure replacing descendant axis $w_1 \downarrow^* w_2$ occurring in t_q by child paths $w_1 \downarrow w_3 \downarrow w_2$, where w_3 is one of the new nodes labeled $\heartsuit(\sharp)$. We proceed in a similar way with sequences of siblings which are given as mere unions. We order them arbitrarily using the next-sibling relation, but we always take care of inserting one of the new $\heartsuit(\sharp)$-labeled nodes in between two siblings which were not previously related by a $\rightarrow$-arrow. We finally substitute fresh distinct constants for every distinct variable, thus obtaining a complete tree. From $q \subseteq q'$, we then infer that there exists another homomorphism $h_2 : t_{q'} \rightarrow T$. Relying on the special properties of h_1, we finally construct the homomorphism $h : (t_{q'})^* \rightarrow (t_q)^*$ from h_1 and h_2 by letting $h(x) = h_1^{-1}(h_2(x))$. □

As before, we immediately obtain the following.

COROLLARY 5.5. *The problem CQ$_\subseteq(\Downarrow, \rightarrow)$ is* NP*-complete.*

As mentioned earlier, replacing $\rightarrow$ by $\Rightarrow$ and obtaining an analog of Theorem 5.4 is impossible without an unlikely collapse of complexity classes.

COROLLARY 5.6. *Assume that there is a polynomial-time algorithm that associates with each query $q \in$ CQ$(\Downarrow, \Rightarrow)$ an incomplete tree $t(q)$ so that $q \subseteq q'$ iff there is a homomorphism $t(q') \rightarrow t(q)$. Then* NP =coNP.

Indeed, since containment of CQ$(\Downarrow, \Rightarrow)$ is Π_2^p-hard and testing homomorphism existence is NP-complete, the existence of such a containment test would imply $\Pi_2^p \subseteq$ NP from which NP =coNP follows easily.

Polynomial-time cases. Our characterization of containment via homomorphisms immediately shows how to obtain polynomial-time cases of containment. Indeed, since containment is now reduced to the existence of homomorphisms, it is effectively cast as a constraint satisfaction (or conjunctive query evaluation) problem. Thus, we can use multiple known results classifying tractable cases of those and apply them to structures representing incomplete data trees. As all of these are quite routine, we leave the complete treatment to the full version (due to space limitations here), and now give just a couple of examples. One is the containment $q \subseteq q'$ for any of the classes CQ$(\downarrow)$, CQ$(\downarrow, \rightarrow)$, and CQ$(\Downarrow, \rightarrow)$ if the query q' is fixed. The other is containment for the classes CQ$(\downarrow)$ and CQ$(\downarrow, \rightarrow)$ when q' mentions each variable at most once (since in this case containment can be reduced to the combined complexity of evaluating conjunctive queries of fixed treewidth). More results will be provided in the full version.

Extension to unions of CQs. A classical result in relational theory says that for unions of relational conjunctive queries, $q = q_1 \cup \ldots \cup q_m$ and $q' = q'_1 \cup \ldots \cup q'_k$, we have $q \subseteq q'$ iff for every $i \leq m$, there exists $j \leq k$ so that $q_i \subseteq q'_j$ [34]. We call this the *SY-criterion* (for Sagiv/Yannakakis) for containment of UCQs. In particular, the SY-criterion implies that the complexity of containment of relational UCQs remains NP-complete (assuming, of course, that they are represented in the above way, as unions of CQs; for other syntactic representations, in which the union is not the outermost operation, the complexity is Π_2^p-complete [34]).

Note that we have defined XML queries in UCQ(σ) to be syntactically of the form $q_1 \cup \ldots \cup q_m$, where each q_i is a CQ(σ)-query. It turns out that for the classes which permit testing containment by means of homomorphisms between incomplete trees t_q, a similar extension to unions continues to be true.

PROPOSITION 5.7. *Queries in* UCQ$(\downarrow)$ *and* UCQ$(\downarrow, \rightarrow)$ *satisfy the SY-criterion for containment.*

This immediately gives us the following.

COROLLARY 5.8. *The problems* UCQ$_\subseteq(\downarrow)$ *and* UCQ$_\subseteq(\downarrow, \rightarrow)$ *are* NP*-complete.*

Indeed, for queries $q = q_1 \cup \ldots \cup q_m$ and $q' = q'_1 \cup \ldots \cup q'_k$ we simultaneously guess a map $f : \{1, \ldots, m\} \rightarrow \{1, \ldots, k\}$, and m maps h_i from $t_{q'_{f(i)}}$ to t_{q_i} for each $i \leq m$, and check, in polynomial time, if the h_is satisfy conditions of Theorem 5.2.

6. THE EFFECT OF WILDCARD

A standard feature of most XML formalisms is the use of *wildcard*, i.e., a special symbol in place of a label that matches every label in a tree. We normally use _ for wildcard. So patterns can be extended in the following way: instead of a pattern that starts with $a(x)$, we can have a pattern that starts with $\_(x)$. It will be witnessed in a node s of a data tree t even if we drop the requirement that labels match. When we deal with classes of patterns $\mathbf{\Pi}(\sigma)$ extended with wildcard, we write $\mathbf{\Pi}(\sigma, \_)$.

For instance, patterns in $\mathbf{\Pi}(\downarrow, \_)$ are given by

$$\pi := a(x)[\pi, \ldots, \pi], \quad a \in \mathcal{L} \cup \{\_\}, \quad x \in \mathcal{V} \cup \mathcal{D}. \qquad (7)$$

The semantics is extended, compared to (1), as follows. For a data tree $t = \langle D, \downarrow, \rightarrow, \lambda, \rho \rangle$, a node $s \in D$,

and a valuation $\nu : \bar{x} \to \mathcal{D}$, we have $(t, s, \nu) \models a(x)[\pi_1(\bar{x}_1), \ldots, \pi_n(\bar{x}_n)]$ iff

- $\lambda(s) = a$ if $a \in \mathcal{L}$;
- $\rho(s)$ is $\nu(x)$ if x is a variable, and x if x is a constant data value;
- there exist not necessarily distinct children $s \cdot i_1, \ldots, s \cdot i_n$ of s so that $(t, s \cdot i_j, \nu) \models \pi_j(\bar{x}_j)$ for each $j \leq n$.

Likewise we define all other classes of patterns extended with wildcard, e.g., $\mathbf{\Pi}(\downarrow, \to, \_)$ and $\mathbf{\Pi}(\Downarrow, \Rightarrow, \_)$, and classes of CQs, UCQs, and BCCQs based on them. For those queries we define the containment problem: for instance, $\mathrm{BCCQ}_{\subseteq}(\Downarrow, \Rightarrow, \_)$ is the problem of checking containment of BCCQs based on patterns from $\mathbf{\Pi}(\Downarrow, \Rightarrow, \_)$.

The question is then whether the use of wildcard increases the cost of testing containment. The first instance of that question is whether we can preserve the Π_2^p upper bound for all containment cases. The answer to this is positive. In fact, our proof of Theorem 3.1 already shows how to handle wildcard.

PROPOSITION 6.1. *The problem* $\mathrm{BCCQ}_{\subseteq}(\Downarrow, \Rightarrow, \_)$ *is in* Π_2^p.

Hence, all other containment problems are in Π_2^p in the presence of wildcard.

What does change, however, is the lower bounds. Recall that we saw in Corollary 5.8 that $\mathrm{UCQ}_{\subseteq}(\downarrow, \to)$ is in NP. The presence of wildcard makes the complexity jump: adding wildcards to $\mathbf{\Pi}(\downarrow, \to)$ patterns makes the complexity of containment of UCQs Π_2^p-hard, rather than being in NP.

THEOREM 6.2. *The problem* $\mathrm{UCQ}_{\subseteq}(\downarrow, \to, \_)$ *is* Π_2^p*-complete.*

Proof sketch. To show hardness, we adapt the lower bound proof of Theorem 4.2 by constructing queries q_{rigid}, q'_{rigid} instead of q, q'. Recall that the query q was encoding all the possible valuations of the p_is using a special pattern over $\mathbf{\Pi}(\Downarrow, \Rightarrow)$. Additionally we used another pattern in q to encode the clauses in φ. We did not describe this pattern in the sketch of Theorem 4.2, but it is enough for the current sketch to note that it can alternatively be represented as a $\mathbf{\Pi}(\downarrow, \to)$-pattern π_φ. We define q_{rigid} by adding to π_φ two new nodes as first and second child of its root. These new nodes are respectively labeled *Val*(0) and *Val*(1). Let π_φ^{01} be the resulting pattern. For every $1 \leq i \leq l$, we also create a single node pattern labeled *Val*(x_i) and we form q_{rigid} by existentially quantifying the x_i's and taking the conjunction of these $l + 1$ patterns. Now we define q'_{rigid} as a disjunction whose first member slightly adapts q', while its second member π^- is the disjunction of all $\mathbf{\Pi}(\downarrow, \to)$ patterns extending π with one single node labeled with wildcard and with a fresh variable over data values. The key idea is now that if a complete tree t does not satisfy π^- but satisfies q_{rigid} via some homomorphism h, then for every x_i, either $h(x_i) = 0$, or $h(x_i) = 1$, i.e., t encodes one particular valuation of the p_is. □

Since wildcard can lead to an increase in complexity of the containment problem, it is natural to ask then when we can match the previously established complexity results in the presence of wildcard. For $\mathbf{\Pi}(\downarrow)$ and $\mathbf{\Pi}(\downarrow, \to)$ patterns the answer to this is surprisingly simple: we can allow wildcard everywhere except at the root of the pattern. Recall that in Section 5 we associated with each pattern π an incomplete tree t_π with a unique root. The requirement is basically that the label of the root of t_π is $a \in \mathcal{L}$; other nodes of t_π can be labeled either by $a \in \mathcal{L}$ or by $\_$.

For instance, the following rules define such patterns based on child-only navigation:

$$\begin{array}{rcll} \pi & := & a(x)[\pi', \ldots, \pi'] & a \in \mathcal{L} \\ \pi' & := & a(x)[\pi', \ldots, \pi'] & a \in \mathcal{L} \cup \{\_\} \end{array} \tag{8}$$

That is, the π's define patterns that can use wildcard, and π is the top-level pattern, whose root label comes from $\mathcal{L}$.

When we have this restriction on patterns with wildcard, we write $\mathbf{\Pi}(\sigma, \_^{\neg r})$, where σ, as before, is a set of axes. Likewise we define classes of queries – e.g., $\mathrm{CQ}(\downarrow, \to, \_^{\neg r})$ – and containment problems – e.g., $\mathrm{CQ}_{\subseteq}(\downarrow, \to, \_^{\neg r})$.

Obviously the addition of wildcard preserves lower bounds. We have already seen that containment of BCCQs with wildcard is in Π_2^p, and hence all three versions of BCCQ containment – $\mathrm{BCCQ}_{\subseteq}(\Downarrow, \Rightarrow)$, $\mathrm{BCCQ}_{\subseteq}(\Downarrow, \Rightarrow, \_)$, and $\mathrm{BCCQ}_{\subseteq}(\Downarrow, \Rightarrow, \_^{\neg r})$ – are Π_2^p-complete.

Now we show that the NP bounds established via homomorphisms are also preserved when wildcard is used everywhere except the root.

PROPOSITION 6.3. *The problems* $CQ_{\subseteq}(\downarrow, \_^{\neg r})$ *and* $CQ_{\subseteq}(\downarrow, \to, \_^{\neg r})$ *are* NP*-complete.*

Proof sketch. We adapt the proof of the corresponding result in Section 5. We now turn t_Q into a complete tree using a slightly different procedure. We just add to it one new root node labeled $\heartsuit(\sharp)$ and we decide arbitrarily on a sibling ordering when none is specified. We finally substitute fresh distinct constants for every distinct variable, thus obtaining a complete tree T. The remainder of the proof is almost as before. Whenever the next sibling relation is available, we only need to notice that the homomorphism $h_2 : t_{Q'} \to T$ cannot map any node in $t_{Q'}$ to the root of T. As tree patterns are rooted, this entails that nothing can be said in $t_{Q'}$ about the relative sibling orderings of the preimages of the children of the root of T. □

Note that such a procedure would not work when both unions of siblings and next sibling are allowed. For instance, let $q = \exists x, y, z\ a(x)[a(y), b(z)]$ and $q' = \exists x, y, z\ a(x)[\_(y) \to \_(z)]$ with $a \neq b$. Obviously $q \subseteq q'$, as q forces the tree to have an a-labeled node with at least two children. On the other hand, it is easy to see that there is no homomorphism from $t_{q'}$ to t_q.

Similarly, the method cannot be applied to queries in

$\mathrm{CQ}(\Downarrow, \_^{\neg r})$. Consider $q = \exists x, y\ a(x)//b(y)$ and $q' = \exists x, y, z\ a(x)/\_(z)//b(y) \wedge \exists x, y, z\ a(x)//\_(z)/b(y)$, with $a \neq b$. Again $q \subseteq q'$, as q forces the tree to have an a-labeled node which has at least one child and a b-labeled descendant which has a parent. But here again, it is obvious that there is no homomorphism from $t_{q'}$ to $(t_q)^*$.

Observe finally that by allowing wildcard to appear everywhere in patterns we also lose the homomorphism criterion that let us establish the NP upper bound. For instance, let $q = \exists x, y\ \big(a(x) \wedge b(y)\big)$, with $a \neq b$, and let $q' = \exists x, y\ (\_(x)/\_(y))$. Since q forces each tree to have at least two nodes, we have the containment $q \subseteq q'$; however there is no homomorphism from $t_{q'}$ to t_q.

As the last result of this section, we show that combining unions of queries even with the restricted use of wildcard can increase the complexity of containment.

PROPOSITION 6.4. *Containment of UCQs that use downward navigation and wildcard except at the root, i.e., the problem* $\mathrm{UCQ}_{\subseteq}(\Downarrow, \_^{\neg r})$*, is* Π_2^p*-complete.*

Proof sketch. We adapt the proof of Theorem 4.2 along the same lines as in the proof of Theorem 6.2. We define queries $q_{\Downarrow}, q'_{\Downarrow}$ as follows. We keep all the $\downarrow$ paths patterns which were actually used in q to encode the clauses of φ, but we now encode the valuation of the p_i's using a pattern $\pi_1/\ldots/\pi_l$ where for each $1 \leq i \leq l$, $\pi_i = \mathit{Val}(0)//\mathit{Val}(x_i)//\mathit{Val}(1)$. We can now construct $q'_{\Downarrow}$ almost as in the proof of Theorem 6.2, except that we replace π_- with a CQ $\exists x_1 \ldots \exists x_{2l+1} \mathit{Val}(0)/\_(x_1)/\ldots/\_(x_{2l+1})$. □

7. THE EFFECT OF SCHEMAS

So far we have not assumed any schema information, such as a DTD or a more general schema description, under which we perform static analysis of queries. However, such assumptions are fairly common, as many XML documents are required to satisfy schema descriptions. Schemas are very well known to affect static analysis of XML. In fact containment of queries can easily behave differently under schemas, even such simple ones as specifying the label of the root of a document. For instance, if $q = \exists x, y\ \big(a(x) \wedge b(y)\big)$ and $q' = \exists x, y\ \big(c(x)/\_(y)\big)$, then in general $q \not\subseteq q'$, but if we state that roots must be labeled c, then $q \subseteq q'$.

In addition, the presence of schemas is known to affect the complexity of static reasoning tasks, generally by increasing it, sometimes even making it undecidable [7, 10, 17, 18, 21, 33, 35]. The main observation of this section is that under schema information, we preserve decidability of query containment for those classes we have encountered so far, but at the cost of an exponential blow-up.

Abstraction of XML schemas. There are many formalisms for describing XML schemas (see, e.g., [29] for a survey), but most of them are subsumed by the notion of an unranked tree automaton. To define it, fix a finite alphabet $\Sigma \subset \mathcal{L}$. A *non-deterministic unranked tree automaton (NTA)* [32, 36] over Σ-labeled trees is a tuple $\mathcal{A} = (Q, \Sigma, \delta, F)$, where Q is a finite set of states, $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Sigma \to 2^{(Q^*)}$ is a transition function. We require that the $\delta(q, a)$'s be regular languages over Q for all $q \in Q$ and $a \in \Sigma$. When we deal with complexity results involving automata, we assume that these regular languages are represented by NFAs (or by regular expressions, since those can be converted into NFAs in polynomial time).

A run of $\mathcal{A}$ over a tree t with domain D and labeling function λ is a function $r_{\mathcal{A}} : D \to Q$ such that for each node s with n children $s \cdot 0, \ldots, s \cdot (n-1)$, the word $r_{\mathcal{A}}(s{\cdot}0) \cdots r_{\mathcal{A}}(s{\cdot}(n{-}1))$ is in the language $\delta(r_{\mathcal{A}}(s), \lambda(s))$. So, for a leaf s labeled a this means that s could be assigned state q iff the empty word ϵ is in $\delta(q, a)$. A run is accepting on tree t if the root of t is assigned an accepting state (formally, $r_{\mathcal{A}}(\epsilon) \in F$. A tree t is accepted by $\mathcal{A}$ if there exists an accepting run of $\mathcal{A}$ on t. The set of all trees accepted by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$.

We then define the containment problem under schemas as follows. Let $\mathcal{Q}$ be one of the classes CQ, UCQ, or BCCQ, and σ a set of axes.

PROBLEM:	$\mathcal{Q}_{\subseteq}(\sigma)$ under schemas
INPUT:	queries $q(\bar{x}), q'(\bar{x}')$ in $\mathcal{Q}(\sigma)$ and NTA $\mathcal{A}$;
QUESTION:	is $q(t) \subseteq q'(t)$ for every $t \in \mathcal{L}(\mathcal{A})$?

A general upper bound. We show that all the versions of $\mathcal{Q}_{\subseteq}(\sigma)$ remain decidable under schemas, but the upper bound is one exponent higher than it was without schemas.

THEOREM 7.1. $\mathrm{BCCQ}_{\subseteq}(\Downarrow, \Rightarrow, \_)$ *under schemas is* 2EXPTIME*-complete.*

Proof sketch. The idea is to prove that we can reduce $\mathrm{BCCQ}_{\subseteq}(\Downarrow, \Rightarrow, \_)$ under schemas to a similar problem over finite alphabets and that we can encode a $\mathrm{CQ}(\Downarrow, \Rightarrow, \_)$ into an exponential-size unranked tree automaton. The 2EXPTIME upper bound then follows from tree automata techniques. The lower bound is immediate from Theorem 7.2.

Lower bounds. Since $\mathrm{BCCQ}_{\subseteq}(\Downarrow, \Rightarrow, \_)$ without the presence of schemas is in Π_2^p (and therefore in single-exponential time), it is natural to ask to whether the jump to double-exponential time is unavoidable. It turns out that it is, even for conjunctive queries, as we can prove the following.

THEOREM 7.2. $\mathrm{CQ}_{\subseteq}(\Downarrow, \Rightarrow)$ *under schemas is* 2EXPTIME*-complete.*

Proof sketch. The upper bound is immediate from Theorem 7.1. The lower bound is obtained in two steps. First we show that we can transfer lower bounds for $\mathrm{UCQ}_{\subseteq}(\Downarrow, \Rightarrow)$ under schemas to lower bounds for $\mathrm{CQ}_{\subseteq}(\Downarrow, \Rightarrow)$ under schemas by adapting a technique from [31]. Then we prove the lower bound for $\mathrm{UCQ}_{\subseteq}(\Downarrow, \Rightarrow)$ by a reduction from the acceptance problem for alternating exponential space bounded Turing machines. This is done by adapting the

proof of the 2EXPTIME lower bound for query containment from Theorem 6 in [10]. Two difficulties arise as that proof used queries with node equalities and wildcards. We handle node equalities by using data value equality constraints in our setting. We show how we can enforce all nodes from a tree to have different data values and then we simulate node equality by data equality. We further provide a modification of the encoding that avoids the use of wildcard. □

We do not yet have a complete classification of what happens for all of the classes of queries under schemas, but we do have an indication very little is needed to make their complexity considerably higher than in the schema-less scenario. In fact one can use results from [12] to prove that even for very simple classes of queries (child relation only; no branching), containment under schemas *provably* requires exponential time.

8. THE EFFECT OF DATA VALUE COMPARISONS

The last feature we are going to consider is data value comparisons, specifically disequalities $\neq$. This is a standard addition that has been considered in the study of relational conjunctive queries. In fact it is one of the mildest ways of adding a limited form of negation to positive queries in a way that preserves their nice properties, such as the decidability of static analysis. The other such extension, also considered here, is allowing Boolean combinations of CQs.

The relational case of CQs with $\neq$ comparisons has been settled in [25, 26, 37]: the containment problem is Π_2^p-complete. From this we can derive some hardness results, for instance, containment of $\mathrm{CQ}(\downarrow)$ with disequalities under schema is Π_2^p-hard (note that the schema assumption is necessary here to ensure documents code relational databases, as was already explained in Section 4). As for upper bounds, for relational BCCQs, even with disequalities, containment is decidable. In fact it is easily seen that such containment reduces to the complement of satisfiability for the Bernays-Schönfinkel class.

However, relational results do not give us any *upper* bounds on the containment problem for XML queries. We show in this section that there is a reason for it: such problems are, by and large, *undecidable*. In fact we show two undecidability results: for XML BCCQs with data comparisons, and even for CQs in the presence of schema information.

Queries with data comparisons. We now formally define classes of queries with $=$ and $\neq$ data comparisons. Suppose we start with a class $\mathbf{\Pi}(\sigma)$ of patterns. Then *CQs with data comparisons* over σ are defined as

$$q(\bar{x}) \;=\; \exists \bar{y}\,\big(\bigwedge_{i=1}^{n} \pi_i(\bar{z}_i) \;\wedge\; \alpha(\bar{x},\bar{y})\big), \qquad (9)$$

where all the π_is are patterns from $\mathbf{\Pi}(\sigma)$ and α is a conjunction of formulae of the form $u = v$ and $u \neq v$, where the variables u and v come from $\bar{x}$ and $\bar{y}$. For instance, $q(x) = \exists y\,(a[b(x), c(y)] \wedge x \neq y)$ is such a query.

The class of such queries will be denoted by $\mathrm{CQ}(\sigma, \sim)$ (using the common XML literature notation of $\sim$ for data value comparisons). We then define the class $\mathrm{UCQ}(\sigma, \sim)$ as unions of queries in $\mathrm{CQ}(\sigma, \sim)$, and $\mathrm{BCCQ}(\sigma, \sim)$ as Boolean combinations of such queries.

Before we present our results, notice that in (9), the formula α allows explicit equalities. Normally in CQs these can be avoided simply by collapsing two variables. However, in the case of pattern-based queries, we may actually need explicit equalities, at least for UCQs. Consider, for example, a Boolean query $q(x, y) = a(x) \wedge a(y)$. Then this query implies the following UCQ $q'(x, y) = (x = y) \vee \_/\_$. Indeed, if $q(x, y)$ is witnessed by two data values that are different, then they must occur in different nodes and hence the $\_/\_$ pattern is true.

Containment without schemas. Without schemas, the containment problem for BCCQs behaves drastically differently from the relation case, as we show below.

THEOREM 8.1. *Containment of BCCQs with data comparisons, i.e., the problem* $\mathrm{BCCQ}_{\subseteq}(\Downarrow, \Rightarrow, \_, \sim)$*, is undecidable.*

In fact one needs either $\downarrow, \downarrow^*, \rightarrow$ or $\downarrow, \rightarrow, \rightarrow^*$ to establish undecidability.

Proof sketch. The proof shows that satisfiability for a BCCQ is undecidable by reduction from Post's Correspondence Problem (PCP). The proof is rather technical. It may be tempting to think that, since $\mathrm{BCCQ}(\Downarrow, \Rightarrow, \_, \sim)$ can express certain key constraints, one can simulate the node equality tests from [10] in our setting by data equalities, and then we can adapt undecidability results from there as well. However, under such a key constraint, it is not clear at all how then the data equalities and inequalities from [10] can be correctly simulated. The reduction from PCP consists of a series of encoding steps that state that (1) all trees satisfying the BCCQ must be string-shaped and of a certain form; and (2) that they somehow encode a PCP solution. The proof can be done in two flavors: either we say that the tree does not branch, in which case we need the negation of the $\rightarrow$ predicate to express (1) as well as both $\downarrow$ and $\downarrow^*$ for (2). Alternatively, we say that the root has no grandchildren, in which case we need $\downarrow$ for (1) and $\rightarrow$ and $\rightarrow^*$ for (2). □

Containment with schemas. As in the previous section, for each containment problem of the form $\mathcal{Q}_{\subseteq}(\sigma, \sim)$, with $\mathcal{Q}$ being CQ, or UCQ, or BCCQ, we can associate an analogous containment problem *under schemas* which, in addition, will take as an input a schema, represented as an automaton.

The combination of data value comparisons and schemas has an even more severe effect on the complexity of the containment problem: it becomes undecidable already for CQs using only downward navigation.

THEOREM 8.2. *The containment problem for* $\mathrm{CQ}(\Downarrow, \sim)$ *queries under schema is undecidable.*

Proof sketch. As in the proof of Theorem 7.2, we first notice that we can transfer lower bounds for $\mathrm{UCQ}_{\subseteq}(\Downarrow, \sim)$ under schemas to lower bounds for $\mathrm{CQ}_{\subseteq}(\Downarrow, \sim)$. After that we prove undecidability for $\mathrm{UCQ}_{\subseteq}(\Downarrow, \sim)$ by reduction from the halting problem of two-counter machines. □

We conclude with the following remark. We noticed earlier that relational results give us Π_2^p-hardness for containment of $\mathrm{CQ}(\downarrow, \sim)$ queries under schemas (to enforce relational encoding). While the precise complexity of the problem $\mathrm{CQ}_{\subseteq}(\downarrow, \sim)$ remains open (see concluding remarks), we can at least eliminate the need for schemas from the hardness result, i.e., we can prove the following.

PROPOSITION 8.3. *The problem $CQ_{\subseteq}(\downarrow, \sim)$ is Π_2^p-hard.*

9. CONCLUSION

We have analyzed the containment problem for three classes of queries – CQs, UCQs, and BCCQs – based on various classes of tree patterns (including $\mathbf{\Pi}(\downarrow)$, $\mathbf{\Pi}(\downarrow, \rightarrow)$, $\mathbf{\Pi}(\Downarrow)$, and $\mathbf{\Pi}(\Downarrow, \Rightarrow)$), also in the presence of extra features such as wildcard, schemas, and disequality comparisons.

Overall, this gives us 96 cases of possible variations of the containment problem, and our results, although not generating the full set of 96 complexity bounds, have provided answers to the majority of them. Nonetheless, there are a few questions left open, that we would like to address. These concern the cases when we have some of the extra features (wildcard, schemas, inequalities) present.

With wildcard, without any restrictions, we do not yet have the precise complexity of containment for four classes: $\mathrm{CQ}(\downarrow, \_), \mathrm{CQ}(\downarrow, \rightarrow, \_)$, $\mathrm{CQ}(\Downarrow, \_)$, and $\mathrm{UCQ}(\downarrow, \_)$. With schemas, we do not yet know whether containment for CQs and UCQs without transitive closure axes is single-exponential or double-exponential. And with disequality comparisons, we do not know if containment without transitive closure axes is decidable. Based on our investigations, all these problems appear to be rather nontrivial. We plan to address them in the future.

Acknowledgment. This work was supported by the FET-Open project FoX (Foundations of XML), grant agreement FP7-ICT-233599, by EPSRC grant G049165, and by DFG grant MA 4938/2-1.

10. REFERENCES

[1] S. Abiteboul, B. Cautis, T. Milo. Reasoning about XML update constraints. In *PODS'07*, pages 195–204.
[2] S. Abiteboul, L. Segoufin, and V. Vianu. Representing and querying XML with incomplete information. *ACM TODS*, 31(1):208–254, 2006.
[3] N. Alon, T. Milo, F. Neven, D. Suciu, V. Vianu. XML with data values: typechecking revisited. *JCSS* 66(4): 688-727 (2003).
[4] S. Amano, L. Libkin, F. Murlak. XML schema mappings. In *PODS'09*, pages 33–42.
[5] S. Amer-Yahia, S. Cho, L. Lakshmanan, D. Srivastava. Tree pattern query minimization. *VLDB J.* 11(4): 315-331 (2002).
[6] M. Arenas, P. Barceló, L. Libkin, F. Murlak. *Relational and XML Data Exchange*. Morgan & Claypool, 2010.
[7] M. Arenas, W. Fan, L. Libkin. On the complexity of verifying consistency of XML specifications. *SIAM J. Comput.* 38(3): 841-880 (2008).
[8] M. Arenas, L. Libkin. XML data exchange: consistency and query answering. *J. ACM* 55(2): (2008).
[9] P. Barceló, L. Libkin, A. Poggi, C. Sirangelo. XML with incomplete information. *J. ACM*, 58:1 (2010).
[10] H. Björklund, W. Martens, T. Schwentick. Optimizing conjunctive queries over trees using schema information. *MFCS'08*, pages 132–143.
[11] H. Björklund, W. Martens, and T. Schwentick. Conjunctive query containment over trees. *JCSS* 77(3): 450-472 (2011).
[12] H. Björklund, W. Martens, T. Schwentick. Validity of tree pattern queries with respect to schema information. Unpublished manuscript.
[13] M. Bojanczyk, C. David, A. Muscholl, T. Schwentick, L. Segoufin. Two-variable logic on words with data. In *LICS'06*, pages 7-16.
[14] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. Regular XPath: constraints, query containment and view-based answering for XML documents. In *LID'08*.
[15] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC 1977*, pages 77–90.
[16] C. David. Complexity of data tree patterns over XML documents. *MFCS'08*, pages 278–289.
[17] W. Fan, L. Libkin. On XML integrity constraints in the presence of DTDs. *J. ACM* 49(3): 368–406 (2002).
[18] D. Figueira. Satisfiability of downward XPath with data equality tests. *PODS'09*, 197-206.
[19] P. Genevés and N. Layaida. A system for the static analysis of XPath. *ACM TOIS* 24 (2006), 475–502.
[20] A. Gheerbrant, L. Libkin, and T. Tan. On the complexity of query answering over incomplete XML documents. *ICDT 2012*, 169–181.
[21] G. Gottlob, C. Koch, K. Schulz. Conjunctive queries over trees. *J. ACM* 53 (2006), 238–272.
[22] T. J. Green. Containment of conjunctive queries on annotated relations. *Theory Comput. Syst.* 49(2): 429-459 (2011).
[23] A. Halevy. Answering queries using views: A survey. *VLDB J.* 10(4):270-294 (2001).
[24] B. Kimelfeld, Y. Sagiv. Revisiting redundancy and minimization in an XPath fragment. *EDBT'08*, pages 61-72.
[25] A. Klug. On conjunctive queries containing inequalities. *J. ACM* 35(1): 146-160 (1988).
[26] P. Kolaitis, D. Martin, M. Thakur. On the complexity of the containment problem for conjunctive queries with built-in predicates. In *PODS 1998*, pages 197-204.
[27] M. Lenzerini. Data integration: a theoretical perspective. In *PODS'02*, pages 233–246.
[28] L. Libkin, C. Sirangelo. Reasoning about XML with temporal logics and automata. *J. Applied Logic*, 8:2, 210–232 (2010).
[29] W. Martens, F. Neven, T. Schwentick. Simple off the shelf abstractions for XML schema. *SIGMOD Record* 36(3): 15-22 (2007).
[30] S. Maneth, T. Perst, H. Seidl. Exact XML type checking in polynomial time. In *ICDT 2007*, pages 254–268.
[31] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1): 2–45, 2004.
[32] F. Neven. Automata, logic, and XML. In *CSL 2002*, pages 2–26.
[33] F. Neven, T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *LMCS*, 2(3): (2006).
[34] Y. Sagiv, M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM* 27(4): 633-655 (1980).
[35] Th. Schwentick. XPath query containment. *SIGMOD Record* 33(1): 101-109 (2004).
[36] J.W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *JCSS* 1 (1967), 317–322.
[37] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *JCSS* 54(1): 113-135 (1997).
[38] V. Vianu. A web Odyssey: from Codd to XML. In *PODS'01*, pages 1–15.