Information Processing Letters 111 (2011) 754–760

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

On the complexity of tree pattern containment with arithmetic comparisons

Foto N. Afrati [a], Sara Cohen [b,*,1], Gabriel Kuper [c]

[a] Department of Electrical and Computing Engineering, National Technical University of Athens (NTUA), 15773 Athens, Greece
[b] Department of Computer Science and Engineering, Hebrew University of Jerusalem, Jerusalem, Israel
[c] Dipartimento di Ingegneria e Scienza dell'Informazione Via Sommarive, 14 I-38123 Povo, Trento, Italy

ARTICLE INFO

Article history:
Received 15 November 2010
Received in revised form 10 February 2011
Accepted 28 April 2011
Available online 12 May 2011
Communicated by J. Chomicki

Keywords:
Databases
Query containment
Tree patterns
XPath queries
Arithmetic comparisons

ABSTRACT

In this paper we investigate the complexity of query containment problem for tree patterns (which express a fragment of XPath) with arithmetic comparisons. We assume that attributes take values from a totally ordered domain and allow constraints that involve arithmetic comparisons. We show that the containment problem is Π_2^p-complete in the general case, but remains co-NP complete for tree patterns with left semi-interval ($<, \leqslant, =$) or right semi-interval ($>, \geqslant, =$) attribute constraints.

1. Introduction

Containment of XPath queries is a well-developed area (e.g., [1–3]). There has been some work on containment with equality ($=$) or inequality constraints ($\neq$) on variables. In particular, [3] studies the containment problem for many fragments of XPath, with and without DTDs, and with constraints on variables. Work on containment in the presence of dependency constraints is studied in [1].

In this paper, we focus on tree patterns with the child and descendant axes, but allow arbitrary arithmetic constraints on the variables; our constraints are comparisons from the set $\{\leqslant, <, \geqslant, >, =\}$. Our tree patterns can express the fragment of XPath including the child and descendent axes, and data comparison filters. In addition, we can express arbitrary comparisons between data values of nodes, which is not directly expressible in XPath. Containment of XPath, or of tree patterns, with arbitrary arithmetic constraints has not been studied in the past. We use the term attributes, as in the XPath standard, to allow more than one variable per node, and to allow comparisons of different attributes with each other. We give complexity results for the general problem, and then give tighter bounds for the case in which the comparisons are restricted to LSI (left semi-interval) or RSI (right semi-interval) constraints [4,5].

1.1. Related work

The are few works that study extending XPath with constraints. [6] discusses satisfiability for a large class of XPath fragments, including *data join*. These are a special case of arithmetic comparisons, and can be easily expressed in our framework. Examination of the proofs in [6] shows that their proofs carry over easily to our model, giving the same PTIME complexity bounds for containment.

In [3], complexity results for the query containment problem are derived for XPath queries with DTDs, disjunction and equality and inequality constraints. An almost

* Corresponding author.
E-mail addresses: afrati@softlab.ntua.gr (F.N. Afrati), sara@cs.huji.ac.il (S. Cohen), kuper@acm.org (G. Kuper).

[1] Sara Cohen was partially supported by the Israel Science Foundation (Grant 143/09).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2011.04.014

complete classification of the complexity of the containment problem is established there for various fragments of such languages. In [1], dependency constraints are added to the basic fragment of XPath and both the problem of query containment and the related problem of query reformulation are studied. [7] shows decidability of containment for various XPath fragments with DTDs and a special class of integrity constraints. In [8] query containment is investigated for a larger fragment of XPath than the core (which does not include constraints) XPath considered in previous papers and of the core XPath fragment in the present paper. Containment of conjunctive queries with data comparisons, over documents, was studied in [9]. Since arbitrary conjunctive queries are allowed, their queries do not necessarily correspond to trees.

A related problem is that of determining satisfiability of XPath queries, since in logics closed under Boolean operators, containment reduces to satisfiability. Data tree patterns, i.e., trees with child and descendant edges, labeled nodes and (in)equality constraints on data values, were studied in [10]. Specifically, the query evaluation problem, as well as satisfiability in the presence of a DTD were studied for Boolean combinations of data tree patterns. Note that [10] is unique in that it uses injective semantics, i.e., nodes in a data pattern must be mapped to different document nodes. Satisfiability of several fragments of XPath, allowing data comparisons, was also studied in [11,12].

2. Preliminary definitions

2.1. XML documents

Let $\mathcal{N}_e$ and $\mathcal{N}_a$ be fixed infinite sets of *element and attribute names*, and let $\mathcal{D}$ be a fixed infinite and totally ordered *domain of values*. We model XML documents as unranked trees, where each node has a label taken from $\mathcal{N}_e$. In addition a node can have any number of attributes, with names from $\mathcal{N}_a$ and values in $\mathcal{D}$. We formally define documents below.

Definition 2.1 *(Document).* A *document* is a tuple $d = (\mathcal{V}, \mathcal{E}, r, lab, @)$ where

1. $(\mathcal{V}, \mathcal{E}, r)$ is an unranked tree with nodes $\mathcal{V}$, edges $\mathcal{E}$ and root $r \in \mathcal{V}$;
2. *lab* is a function from $\mathcal{V}$ to $\mathcal{N}_e$, which associates each node with an element name;
3. for each $a \in \mathcal{N}_a$, $@_a$ is a partial function from $\mathcal{V}$ to $\mathcal{D}$, which associates nodes with attributes and values.

2.2. Query syntax

Tree pattern queries (or simply *queries* for short) are defined in a similar fashion to documents, with two main differences. First, queries have two types of edges: child-edges and descendant-edges. Second, queries can have a boolean formula of constraints over attribute values. The formal definition of a query follows next.

Definition 2.2 *(Query).* A *query* is a tuple $q = (\mathcal{V}, \mathcal{E}_/, \mathcal{E}_{//}, r, lab, \alpha, \bar{X})$ where

- $(\mathcal{V}, \mathcal{E}_/ \cup \mathcal{E}_{//}, r)$ is an unranked tree with nodes $\mathcal{V}$, edges $\mathcal{E}_/ \cup \mathcal{E}_{//}$ and root $r \in \mathcal{V}$;
- $\mathcal{E}_/$ and $\mathcal{E}_{//}$ are disjoint sets, called *child* and *descendant* edges, respectively;
- *lab* is a function from $\mathcal{V}$ to $\mathcal{N}_e \cup \{*\}$, which associates each node with an element name or with a special symbol $*$, called the *wildcard symbol*;
- α is a conjunction of atomic comparisons of the form

  $@_a X \theta @_b Y \qquad @_a X \theta c$

  where $X, Y \in \mathcal{V}$, $a, b \in \mathcal{N}_a$, $c \in \mathcal{D}$, and θ is one of the relations $\geqslant$, $\leqslant$, $<$, $>$, $=$, and $\neq$.
- $\bar{X}$ is a tuple of distinct nodes of q, called the *distinguished nodes*.

To make the presentation clear, we use lowercase letters from the end of the alphabet $(x, y, \ldots)$ to denote nodes in a document, while uppercase letters from the end of the alphabet $(X, Y, \ldots)$ denote nodes in a query. We will use lowercase letters from the beginning of the alphabet (a, b) to denote names of attributes and uppercase letters from the beginning of the alphabet to denote element names (A, B).

We call α the *attribute constraint* of q. We will be particularly interested in some special types of attribute constraints. We say that α is a *semi-interval constraint* (*SI constraint* for short) if all the atomic comparisons in α are of the form $@_a X \theta c$ where $\theta \in \{\geqslant, \leqslant, <, >, =\}$. Similarly, we say that α is a *left (resp. right) semi-interval constraint* (or *LSI, resp. RSI*, for short) if all its atomic comparisons have the form $@_a X \theta c$ where $\theta \in \{<, \leqslant, =\}$ (resp. $\theta \in \{>, \geqslant, =\}$). Note that our LSI and RSI constraints allow comparisons that equate an attribute value to a constant, which is a somewhat more general definition of LSI and RSI than is typical.

If $\bar{X} = (X_1, \ldots, X_k)$, we say that q is of *arity* k. In particular, if $\bar{X}$ is the empty tuple, we say that q is *Boolean*. If α is the empty conjunction, then we say that q is *attribute oblivious*.

2.3. Query semantics

We formally define the semantics of our queries, by means of homomorphisms.

Let $q = (\mathcal{V}_q, \mathcal{E}_{/q}, \mathcal{E}_{//q}, r_q, lab_q, \alpha_q, \bar{X})$ be a query and $d = (\mathcal{V}_d, \mathcal{E}_d, r_d, lab_d, @_d)$ be a document. A *homomorphism* $h : q \to d$ is a mapping from $\mathcal{V}_q$ to $\mathcal{V}_d$ satisfying the following conditions

1. $h(\text{root}_q) = \text{root}_d$;
2. for all $(X, Y) \in \mathcal{E}_{/q}$, $(h(X), h(Y)) \in \mathcal{E}_d$;
3. for all $(X, Y) \in \mathcal{E}_{//q}$, $h(X)$ is a strict ancestor of $h(Y)$ in d;
4. for all $X \in \mathcal{V}_q$, $lab_q(X) = *$ or $lab_q(X) = lab_d(h(X))$;
5. α is satisfied by h, i.e., replacing every X in α with $h(X)$ yields a conjunction that is satisfied over d.

In particular, Property 5 implies that for all $@_a X$ appearing in α, the function $@_a$ is defined over $h(X)$ in d.