Information Processing Letters 119 (2017) 19–24

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

A revised result on chasing tree patterns under schema graphs

Junhu Wang [a,*], Jeffrey Xu Yu [b], Jixue Liu [c], Chaoyi Pang [d]

[a] Griffith University, Australia
[b] The Chinese University of Hong Kong, Hong Kong, China
[c] The University of South Australia, Australia
[d] Zhejiang University (NIT), China

ARTICLE INFO

Article history:
Received 23 May 2016
Received in revised form 18 August 2016
Accepted 16 November 2016
Available online 30 November 2016
Communicated by Jef Wijsen

Keywords:
Databases
XML
Tree pattern
DTD
Containment

ABSTRACT

We first provide an example to show that two previously published results in [2] and [4] on tree pattern containment under schema graphs are incorrect. We then show that the original result in [2] holds under some special conditions on the schema graph.

1. Introduction

XPath plays a central role in all XML query languages. A major fragment of XPath can be represented as tree patterns [3]. Testing tree pattern containment, i.e., whether the answers to a tree pattern are all answers to another tree pattern, is fundamental for XML query optimization. It is shown in [3] that, when P and Q involve only /, // and [], P is contained in Q if and only if there is a homomorphism from Q to P. Unfortunately, when a DTD is present, the existence of a homomorphism from Q to P is no longer a necessary condition for P to be contained in Q. However, [5] shows that if the DTD is *duplicate-free* and the tree patterns involve / and [] only, then testing whether a tree pattern P is contained in another pattern Q under the DTD can be reduced to testing whether P is contained in Q under two types of constraints implied by the DTD. Subsequently [2] claimed this result can be extended to tree patterns involving /, // and [] under an acyclic schema graph (which is a type of simplified DTD), and [4] attempted to extend the result of [2] to cyclic schema graphs. Unfortunately, the claims in [2] and [4] are both incorrect, as we will show in this paper. We then present a special case and show that in this special case, the containment problem of two tree patterns can be reduced to the containment problem under the constraints identified in [2].

2. Preliminaries

For any rooted graph $\mathcal{G}$, we use $\texttt{root}(\mathcal{G})$ and $\texttt{node}(\mathcal{G})$ to denote the root and the node set, respectively, of $\mathcal{G}$.

Schema graph and XML tree. Similarly to [2], we model an XML schema as a connected directed graph G (called a **schema graph**) satisfying the following conditions: (1) Each node is labeled with a distinct label; (2) Each edge is labeled with one of 1, ?, +, and $*$; (3) There is a unique node, called the root, such that every other node is reachable from this node. The set of labels occurring in G

* Corresponding author.
E-mail address: J.Wang@griffith.edu.au (J. Wang).

http://dx.doi.org/10.1016/j.ipl.2016.11.005
0020-0190/

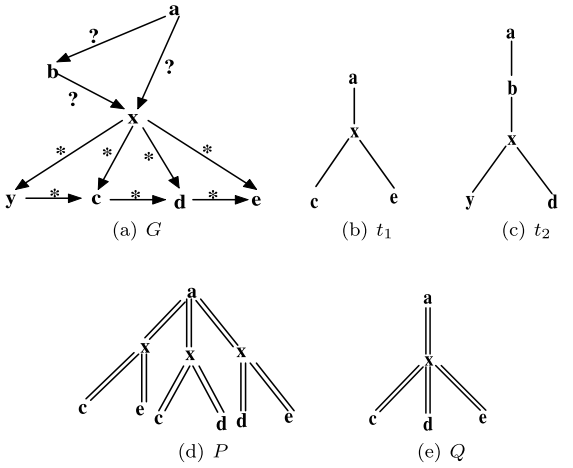

Fig. 1. Schema graph G, conforming XML trees t_1 and t_2, and tree patterns P and Q satisfiable under G.

is denoted Σ_G. Because a node in a schema graph G has a unique label, we also refer to a node by its label. A schema graph may be cyclic or acyclic. An example acyclic schema graph is shown in Fig. 1 (a).

As in previous work [1,3,2] we model an XML document as a finite unordered tree (referred to as an **XML tree**) where every node has a label. Let v be a node in an XML tree t. The label of v is denoted $label(v)$. An XML tree t is said to **conform** to schema graph G if (1) for every node $v \in \mathtt{node}(t)$, $label(v) \in \Sigma_G$; (2) $label(\mathtt{root}(t)) = label(\mathtt{root}(G))$; (3) for every edge (u, v) in t, there is a corresponding edge $(label(u), label(v))$ in G; and (4) for every node $v \in \mathtt{node}(t)$, the number of children of v labeled with x is constrained by the label of the edge $(label(v), x)$ in G: there must be exactly one, zero or one, one or many, or zero or many children of v labeled with x if the label of the edge $(label(v), x)$ is 1, ?, $+$, or $*$ respectively. Fig. 1 (b) and (c) show two XML trees conforming to the schema graph G in Fig. 1 (a). The set of all XML trees conforming to G is denoted T_G.

Tree patterns. We consider the class of Boolean tree patterns (or simply *tree patterns* or *patterns*) $\mathcal{P}^{\{/,//,[]\}}$ as defined in [3]. Each pattern in $\mathcal{P}^{\{/,//,[]\}}$ is a node-labeled tree with two types of edges: /-edges and //-edges. It corresponds to an XPath expression involving the child-axis (/), descendant-axis (//), and branch condition ([]). Fig. 1 (d) and (e) show two tree patterns. Here, single and double lines represent /-edges and //-edges respectively. A branch in the tree represents a condition ([]) in the XPath expression.

Given a pattern P and an XML tree t, $P(t)$ returns `true` if and only if there is an embedding of P in t. An **embedding** of a tree pattern P in an XML tree t is a mapping δ from $\mathtt{node}(P)$ to $\mathtt{node}(t)$ with the following conditions: (1) label-preserving, i.e., $\forall v \in \mathtt{node}(P)$, $label(v) = label(\delta(v))$; (2) root-preserving, i.e., $\delta(\mathtt{root}(P)) = \mathtt{root}(t)$; and (3) structure-preserving, i.e., for every edge (x, y) in P, if it is a /-edge, then $\delta(y)$ is a child of $\delta(x)$; if it is a //-edge, then $\delta(y)$ is a descendant of $\delta(x)$, i.e., there is a path from $\delta(x)$ to $\delta(y)$ of length greater than 0.

A tree pattern P is said to be **satisfiable** under schema graph G if there exists $t \in T_G$ such that $P(t)$ is true. The tree patterns shown in Fig. 1 (d) and (e) are satisfiable under the schema graph G in Fig. 1 (a). In this paper we implicitly assume all tree patterns are satisfiable under the schema graphs being discussed.

Tree pattern containment. Given two patterns P and Q, P is said to be contained in Q (under G), denoted $P \subseteq Q$ ($P \subseteq_G Q$) if for every XML tree t ($\in T_G$), $P(t) \Rightarrow Q(t)$. P and Q are said to be *equivalent* (under G) if $P \subseteq Q$ and $Q \subseteq P$ ($P \subseteq_G Q$ and $Q \subseteq_G P$). It is well known that for any $P, Q \in \mathcal{P}^{\{/,//,[]\}}$, $P \subseteq Q$ if and only if there is a homomorphism from Q to P [1]. A **homomorphism** [3] from Q to P is a mapping δ from $\mathtt{node}(Q)$ to $\mathtt{node}(P)$ that is label-preserving, root-preserving as defined in the definition of embedding, and structure-preserving which now means that, for every edge (x, y) in Q, if it is a /-edge, then $(\delta(x), \delta(y))$ is a /-edge in P; and if it is a //-edge, then there is a path from $\delta(x)$ to $\delta(y)$ of length > 0.

In the following, we will use *G-path* to refer to a path in G; */-child* (resp. //-child) to refer to a child connected to the parent via a /-edge (resp. //-edge); *(/, x)-child* to refer to a /-child labeled x, and $x//y$-edge to refer to a //-edge from an x-node to a y-node.

3. The incorrect results in previous work

It is claimed in Theorem 4 of [2] that, if P and Q are both in $\mathcal{P}^{\{/,//,[]\}}$ and the schema graph G is acyclic, then whether P is contained in Q under G can be reduced to tree pattern containment under the following five types of constraints[1] (referred to as the LWZ constraints in this paper) implied by G (a schema graph G is said to **imply** a constraint C, denoted $G \models C$, if every XML tree conforming to G satisfies C).

(1) **Parent-Child Constraints (PC)**, denoted $a \Downarrow^1 x$, which means that whenever an x-node is the descendant of an a-node, it must be the child of the a-node.
(2) **Sibling Constraints (SC)**, denoted $a : \downarrow y$. The constraint means that every a-node must have a y-child.[2]
(3) **Cousin Constraints (CC)**, denoted $a : x \Downarrow y$. The constraint means that for every a-node, if it has an x-descendant, then it also has a y-descendant.
(4) **Intermediate Node Constraints (IC)**, denoted $a \xrightarrow{x} y$, which means that every path from an a-node to a y-node must pass through an x-node.
(5) **Functional Constraints (FC)**, denoted $a \twoheadrightarrow x$, which means that every a-node has at most one x-child.

[1] The SC and FC were originally defined in [5].

[2] The original SC constraint in [5] is of the form $a : S \downarrow y$ where S is a set of labels, which means that if an a-node has a z-child for every $z \in S$, then it must also have a y-child. The SC constraint in [2] is a special case where S may contain at most 1 label. We note the schema graph in [2] allows a node to represent a sequence of labels. If there are no sequence types in the schema graph, then any SC must be of the form $a : \emptyset \downarrow y$ (called *child constraints* in [5]), which is abbreviated as $a : \downarrow y$ in this paper.