Information Systems 35 (2010) 315–334

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Query rewritings using views for XPath queries, framework, and methodologies

Jian Tang[a,*], Ada Waichee Fu[b,1]

[a] Department of Computer Science, Memorial University of Newfoundland, Elizabeth Ave, St. John's NL, Canada A1B 3X5
[b] Department of Computer Science and Engineering, Chinese University of Hong Kong, Shatin, Hong Kong

ARTICLE INFO

Article history:
Received 9 September 2006
Received in revised form
26 January 2009
Accepted 26 October 2009
Recommended by: Y. Ioannidis

Keywords:
XML
Query
View
Rewriting
Pattern
Containment
Embedding
Maximality

ABSTRACT

Query rewriting using views is a technique that allows a query to be answered efficiently by using pre-computed materialized views. It has many applications, such as data caching, query optimization, schema integration, etc. This issue has been studied extensively for relational databases and, as a result, the technology is maturing. For XML data, however, the work is inadequate. Recently, several frameworks have been proposed for query rewriting using views for XPath queries, with the requirement that a rewriting must be complete. In this paper, we study the problem of query rewriting using views for XPath queries without requiring that the rewriting be complete. This will increase its applicability since in many cases, complete rewritings using views do not exist. We give formal definitions for various concepts to formulate the problem, and then propose solutions. Our solutions are built under the framework for query containment. We look into the problem from both theoretic perspectives, and algorithmic approaches. Two methods to generate rewritings using views are proposed, with different characteristics in terms of generalities and efficiencies. The maximality properties of the rewritings generated by these methods are discussed.

1. Introduction

Since it emerged as a language for information transfer and storage late last century, XML has caught increasing attentions from the research communities across different disciplines for its flexible encoding schemes and expressive power. Recently, due to the near completion of the standardization of XQuery language [25], the usage of XML has reached far beyond the simple information-encoding domain. Such a broad adoption has led not only to the developments of new paradigms, but also the reformulations of some existing theories and methodologies in relational databases. One of the areas in which such a reformulation is in a pressing need is in query processing. Due to the relatively complex structure of XML documents compared with relation tables, efficiency in querying XML documents has become one of the most widely investigated topics in recent years.

XQuery employs XPath as its core sub-language for navigating XML documents. In the query processing literature for XML documents, therefore, a lot of attentions have been in XPath processing. One approach is query rewriting. It is a technique that allows a query to be answered efficiently by using pre-computed materialized views. It has many applications, such as data caching, query optimization, schema integration, etc. This issue has been studied extensively, in both theoretical and algorithmic aspects, for relational databases. As a result, sound theories and methodologies have been proposed in that context. For XML data, however, the work is inadequate. Recently, several frameworks have been proposed for query rewriting using views for XPath queries, with the requirement that a

* Corresponding author. Tel.: +1 7097374580; fax: +1 7097372009.
E-mail addresses: jian@mun.ca (J. Tang), adafu@cse.cuhk.edu.hk (A.W. Fu).
[1] Tel.: +852 26098432; fax: +852 26035024.

0306-4379/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2009.10.006

rewriting must be complete, being that *all* the answers to the query running on the original database must be generated by the rewriting running on the materialized views [2,3,22]. Requiring a complete rewriting using views, however, is not always realistic. The most prominent scenario is in the area of schema integration. Local-as-view (LAV) is an important strategy for schema integration in database systems [8]. In the LAV model, queries are written over a global schema (also called a mediated schema). Views are queries that describe the contents of the local data sources, and are also written over the global schema. The model does not assume the existence of a separate 'original' data source. The direct goal of a query is to retrieve information from the local data sources. To accomplish this, it must be rewritten into some query that can run on the local data sources and produce the answer it needs. Since there is no original data source, 'completeness' of the query results generated from the local data sources is not required. Incomplete rewriting may also be useful in the case where the original data are not conveniently accessible but a materialized view is available. In this case, a user has an option to use the materialized view to obtain an answer which is only a proper subset of the answer he/she has expected from the original data.

It is well known that if a rewriting using view is complete, then the query and its rewritten version are equivalent. Such an equivalence provides valuable information to guide an algorithm to generate the rewriting. If the completeness requirement is removed, however, the aforementioned information will not be available. This makes generating a rewriting using views with desirable properties a challenging task. In this paper, we study the problem of query rewriting using views for XPath queries without requiring the rewriting be equivalent to the original query. We give formal definitions for various concepts to formulate the problem. We look into the problem from both theoretic perspectives, and algorithmic approaches, and then propose solutions. Our solution is built on top of the theories for query containment. We introduce the concept of 'trap', based on which two methods to generate rewritings using views are proposed. These methods have different characteristics in terms of generalities and efficiencies. We describe conditions under which our generated rewritings are optimal. The class of the XPath queries that we consider in this paper uses four kinds of symbols, /, //, [·], and *, which, respectively, denote child axis, descendant axis, branches, and wildcard. We denote this class by $XP^{[/,\ //,\ [\ ],\ *]}$. The query in this class can be described in a condensed grammar as follows:

$$X = X/X|X{/\!/}X|X[X]|L|^{*}$$

where L denotes labels from an infinite symbol set. We will omit tagging templates that normally accompany XPath queries, and concentrate only on the navigation scripts. This is because, technically, tagging templates and navigation scripts are orthogonal, and the navigation scripts are where the most technical sophistications arise in the query rewriting.

The rest of the paper is organized as follows. In Section 2, we propose a model, and introduce related concepts, and then precisely define the problem. In Section 3, we introduce two alternative solutions to the problem, and discuss their strengths and limitations. In Section 4, we describe the conditions under which the rewritings using views generated by our methods are optimal. Section 5 concludes the paper by summarizing the main results, and suggesting some issues for further study.

1.1. A motivating example

If the information required by a query has already been included in the result of a materialized view, and there is a way to retrieve it, then this usually will make the query processing more efficient. Consider the XPath query */publication/book[@review_id]//author/name*. This query requires the names of the authors of the books which belong to the publication category and have a review_id attribute. Let us consider three cases: (1) The view is */publication/book*. In this case the answer of the query is retrievable from the result of the view. It is because, according to the execution semantics, this query will return the entire subtree rooted at each book node in the input document tree. (2) The view is */publication[@permit_no]/book*. In this case, we cannot retrieve a complete answer to the query from the view result. This incompleteness is due to the lack of data, and is intrinsic to the way the queries are formulated. (3) The view is the same as that in the first case, but the query is */publication[@permit_no]/book[@review_id]//author/name*. In this case, the required information is not retrievable. This is not due to the lack of data, but due to the lack of knowledge: we do not know which book in the view result is a child of a publication with permit_no attribute. Obviously, the first case is most preferable. However, given the way the queries are formulated in the second case, if we can get everything contained in the result that fits the user's requirement, it may still be useful if sufficient data is difficult to obtain. Note that the problem arising in the third situation is worse than that in the second case: we cannot retrieve any answer without risking an error.

2. Concepts and definitions

In this paper, we will use the following notations. For any tree or path t, $|t|$ denotes the number of nodes in t. We use $\langle a,\ldots,b\rangle$ to denote a generic path which can be of any length (in nodes), where a and b are the start and the end nodes of the path, respectively, while $\langle a\rangle$ and $\langle a, b\rangle$ denote a path with a length of one and two, respectively. For two graphs, in particular, trees, we use the terms 'isomorphic', 'equal', 'same' interchangeably. Sometimes, for easy presentation and notations, we will allow same nodes (and associated edges) to belong to multiple trees, and therefore avoid using isomorphism symbols on them.

2.1. Pattern tree and input tree

An XPath query can be denoted as a tree, called a *pattern tree* (or simply pattern). Each node is attached with a label, except for the root. The tree may contain

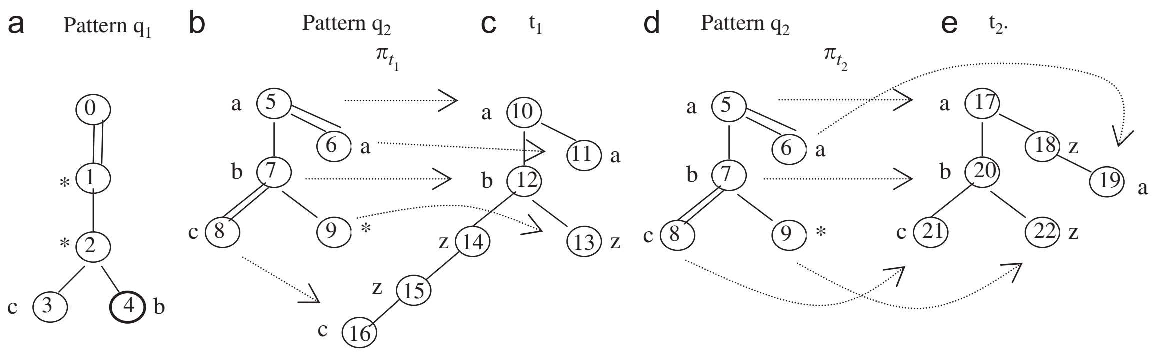

Fig. 1. Canonical models and canonical model mappings.

branches, and can contain two kinds of edges, parent/child and ancestor/descendent, called C_edges and D_edges, respectively. (A C_edge and a D_edge are denoted by a single line and a double line, respectively, in a pattern tree.) If there is a C_edge or a D_edge from n_1 to n_2, we say n_2 is, respectively, a *C_child* or *D_child* of n_1. (Or n_1 is a C_parent or D_parent of n_2.)

Each node in the pattern tree is labeled, and there is a special label '*', which is a wildcard meaning that the node label can be any label. Among all the nodes in a query, there is a distinguished node, called the *return node*. A return node will be matched to a data node as the output of the query. XPath queries execute on XML documents, which can be modeled as labeled trees. In this paper, we call them *input trees* (or simply trees). The execution proceeds by matching the nodes in the pattern tree to the nodes in the input tree, as explained in the next subsection.

2.2. Embedding and query containment

The concepts and notations in this subsection are from [12]. Let q be a pattern and t be an input tree.

An *embedding* is a mapping e: nodes(q)→ nodes(t) such that

(1) $e(\text{root}(q))=\text{root}(t)$,
(2) For any non-root node $n \in$ nodes(q), either label(n)='*', or label(n)=label($e(n)$), and
(3) For any $n_1, n_2 \in$ nodes(q), if n_1 is a C_parent of n_2, then there is an edge from $e(n_1)$ to $e(n_2)$; and if n_1 is a D_parent of n_2, then there is a path from $e(n_1)$ to $e(n_2)$.

For easy reference, we call condition 1 the *root condition*, condition 2 the *node condition* and condition 3 the *edge condition*. For each $n \in$ nodes(q), we say e matches n to $e(n)$.

Let r be the return node in a pattern tree p. The set anws(p, t)={$e(r)$|e: nodes(p)→ nodes(t) is an embedding} is called the answer to p on t. We say that pattern q *is contained in* p, denoted as $q \subseteq_{\clubsuit} p$, if anws($q$, t)⊆ anws(p, t) for all t. (To avoid confusing pattern containment from set containment, unless otherwise mentioned, the expression 'p contains q' always refers to pattern containment for any patterns p and q in this paper.)

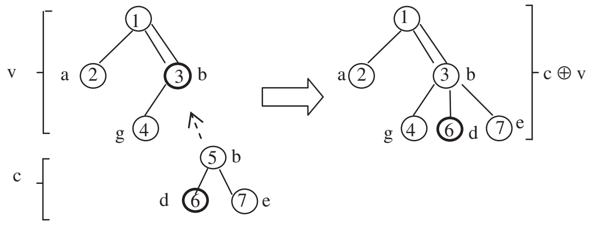

Fig. 2. Concatenating two patterns.

Let q_1 and q_2 be patterns, and q_2 contains m D_edges. Let $\vec{u} = \langle u_1, \ldots, u_m \rangle$, where for all $1 \le i \le m$, u_i is a non-negative integer. The *$\vec{u}$-extension of q_2 for q_1* is a tree formed as follows. First, make an identical copy of q_2. Then, select a symbol, say z, which is not labeled by any node in q_1. Finally, in the copy of q_2, replace the label * with symbol z, and replace the ith D_edge, say ab, by a path $a\lambda b$ where λ contains u_i nodes, all labeled z. We call $a\lambda b$ a *rubber path*,[2] signifying that it can have different lengths for different $\vec{u}$-extensions, the nodes in λ *rubber nodes*, and the number of rubber nodes in $a\lambda b$ the *degree* of $a\lambda b$.

Given any $\vec{u}$-extension t of q_2 for q_1, we use π_t to denote the *canonical model mapping* that maps each node in q_2 to its copy in t. Thus, π_t is a one-to-one onto mapping from the nodal set of q_2 to the set of non-rubber nodes in t. (Obviously, π_t is an embedding.) If in t, for all $1 \le i \le m$, u_i=c, then t is referred as a *c-extension*. Call a path a *star-path* in a pattern if all its nodes are labeled *, and incident only with C_edges. A $\vec{u}$-extension of q_2 for q_1 is referred as a *canonical model* of q_2 for q_1 if $\vec{u} = \langle u_1, \ldots, u_m \rangle$ where for all $1 \le i \le m$, $0 \le u_i \le L+1$ and L is the length, in nodes, of the longest star-path in q_1. Thus, there are $(L+2)^m$ canonical models of q_2 for q_1.

Consider the example in Fig. 1, where t_1 and t_2 are two of the canonical models of q_2 for q_1. The dotted lines, indicated by $\pi_{t_1}$and π_{t_2}, are the canonical model mappings. There are two rubber paths in t_1, $\langle 10, 11 \rangle$ and $\langle 12, \ldots, 16 \rangle$, with degrees 0 and 2, respectively. Nodes 14 and 15 are the rubber nodes in t_1. Note that path $\langle 2 \rangle$, the longest star-path in q$_1$, has a length of 1.

[2] The original work in [12] has not given a name to this concept.

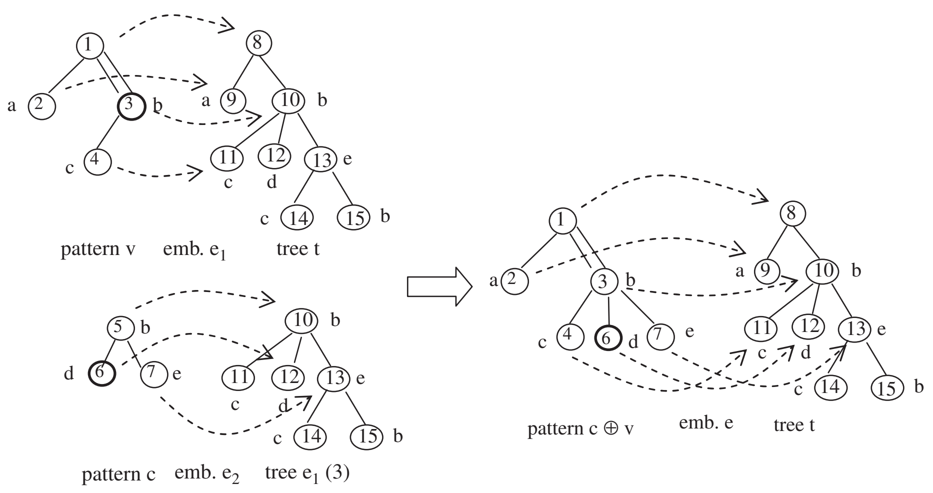

Fig. 3. Merge two embeddings into one.

Thus there are totally 9 canonical models of q_2 for q_1. From the figure, it is clear that a canonical model mapping maps any node only to its copy, and therefore will never map nodes to rubber nodes. This point is crucial for some derivations in later sections.

2.3. Specification of the problem

Throughout this paper, we assume that when an XPath expression returns a node from an input tree, it actually returns the entire sub-tree rooted at that node, which includes both the structure and the labeling. Thus, the following statements are equivalent: ‘a return node matches a sub-tree’ and ‘a return node matches the root of a sub-tree’. Also, we assume that no other information in addition to the result itself, such as the information about the path, is returned. We believe this makes practical sense since at the time when a query is run, a user probably has no interest in obtaining extra information that is unnecessary for his/her jobs.

Definition 2.1. Let v be a pattern and t be an input tree. Let e: nodes(v)→ nodes(t) be an embedding. Let $\mathbf{r_v}$ be the return node in v. The *result of v on t under e*, is *result*[v, t, e]=e(r_v).

The result of v on t under e denotes the output of r_v under e on t. Here, we treat the result of a pattern as part of the input tree, not a separate document. In general there can be multiple embeddings for v and t, and hence there can be multiple results. We call the set of all such results the result set. Given a query q, and the result set S of a view v from an input tree t, we would like to find a set of queries Q to run on result set S, such that the execution will generate the maximal subset of the output set that the query would also be able to generate should it run on the input tree t.

Forming the set of queries Q above is a rewriting problem of q based on v. In our solution, we use the concept of *concatenation* introduced in [22]. Let r_v be the return node in v, and c be a separate pattern from v. Assume either label(root(c))=* or label(root(c))=label(r_v). Then we

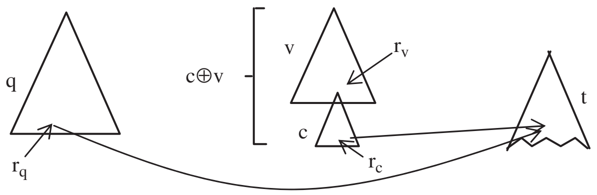

Fig. 4. An abstract view of rewriting.

can concatenate c with v by merging root(c) with r_v. The merged node will have the same label as that for r_v. If r_v has a child prior to the merge, it will keep the child after the merge. The return node in c will be the return node in the new pattern. We call the new pattern a *concatenated pattern for c and v*, and denote it by $c⊕v$. (Note that r_v will no longer be the return node in $c⊕v$, unless root(c) is the return node in c.) Shown in Fig. 2 is an example of concatenation, where nodes 3 and 5 are merged.

Given an embedding e_1: nodes(v)→ nodes(t) and e_2: nodes(c)→ nodes(e_1(r_v)), (Recall e_1(r_v)=result(v, t, e_1)) we can merge e_1 and e_2 into a single embedding e: nodes($c⊕v$)→nodes(t) in the following way: e(n)=e_1(n) if n∈ nodes(v), and e(n)=e_2(n) if n ∈ nodes(c)–root(c). Thus e has the same effects as e_1 and e_2 combined. This is the key point for using concatenation. An example is shown in Fig. 3, where we duplicate the patterns and concatenations from Fig. 2. The two embeddings (denoted by broken lines) e_1 and e_2 are merged into embedding e. Note that e_1(3) denotes the sub-tree rooted at 10, which is the result of pattern v running on tree t. From the figure, we can see that since e_2 matches node 6 in pattern c to node 12 in tree e_1(3), e matches node 6 in pattern $c⊕v$ to node 12 in t also, and vice versa.

What does a concatenation have to do with rewriting? Let us consider Fig. 4, where r_c is the return node in both c and $c⊕v$. If for all t, and all embedding from $c⊕v$ to t, there is an embedding from q to t, such that r_c and r_q are matched to the same node in t, then $c⊕v$ is contained in q. Under this condition, we view $c⊕v$ as a rewriting of q using v. We will formalize this idea shortly.

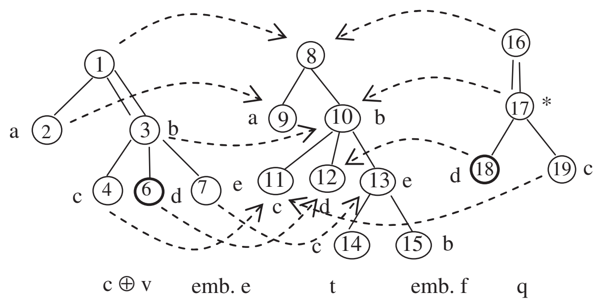

Fig. 5. $\{q\}\supseteq_{\clubsuit}\{c\oplus v\}$ on t.

Definition 2.2. Let Q_1 and Q_2 be two sets of patterns, and t be an input tree. Then

1. Q_1 *contains* Q_2 on t if: $\forall q_2\in Q_2$, $\forall e_2$: nodes(q_2)→ nodes(t), $\exists q_1\in Q_1$, $\exists e_1$: node(q_1)→node(t), [e_1(return_-node(q_1))=e_2(return_nodes(q_2))].
2. If Q_1 contains Q_2 on all t, then Q_1 *contains* Q_2.
3. If Q_1 contains Q_2 and Q_2 contains Q_1, then Q_1 *is equivalent to* Q_2.

The above definitions simply extend the traditional pattern containment in [12] to sets of patterns.

Definition 2.3. Let q and v be patterns, and C be a set of patterns. Let $R=\{c\oplus v|c\in C\}$. Then

1. R is a *rewriting for q using v* if $\{q\}$ contains R. In this case, for each $c\oplus v\in R$, c is called a *compensation for q using v*.
2. R is a *complete rewriting for q using v* if $\{q\}$ is equivalent to R.
3. R is a *maximal rewriting for q using v* if for all rewriting R' for q using v, R contains R'.

Informally, R is a rewriting for q using v if q generates all the results that R generates. This captures the soundness property. For a rewriting R, it is a complete rewriting if it can generate all the results that q generates. It is maximal if it can generate all the results that any other rewriting generates. Clearly, if a complete rewriting exists, then the maximal rewriting is necessarily complete.

Consider the example shown in Fig. 5. We duplicate the concatenation from Fig. 3. We can see that f and e both map the corresponding return nodes to the same node in t. Since e is the only embedding from $c\oplus v$ to t, by Definition 2.2, $\{q\}$ contains $\{c\oplus v\}$ on t. We now prove that $\{q\}$ contains $\{c\oplus v\}$ on any tree t'. This result will establish that $\{q\}$ contains $\{c\oplus v\}$, and therefore, $\{c\oplus v\}$ is a rewriting for q using v. Let e': nodes($c\oplus v$)→ nodes(t') be any embedding. We must have the following: (1) $e'(1)$ is the root of t', and $e'(3)$ is labeled b and is a descendant of $e'(1)$ in t′, (2) $e'(4)$ is labeled c, and is a child of $e'(3)$ in t', (3) $e'(6)$ is labeled d, and is a child of $e'(3)$ in t'. Consider the mapping f: nodes(q)→ nodes(t') defined as: $f(16)=e'(1)$, $f(17)=e'(3)$, $f(18)=e'(6)$ and $f(19)=e'(4)$. Clearly, f is an embedding. Since 18 is the return node in q, and 6 is the return node in $c\oplus v$, by Definition 2.2.1, our claim follows.

Note that in this example, the rewriting contains only a single pattern. In more complex case, however, it may contain multiple patterns. (Refer to the examples in later sections.)

Now, we can formally describe the problem: given patterns q and v, how do we find a rewriting R for q using v, and under what conditions R is maximal? In the subsequent sections, we introduce our solutions.

3. Trap-based search methods

Let q and v be patterns. Searching for a rewriting for q using v essentially requires searching for compensation patterns. As shown in Fig. 4, a compensation pattern c is the lower part, while v is the upper part, in concatenation $c\oplus v$. Since q must contain $c\oplus v$, we can partition q into a lower and an upper portion in such a way that they contain c and v, respectively. The question is how we do the partition. This motivates the trap-based search method.

3.1. Trap embeddings

3.1.1. Concepts

We now define a new kind of embedding, where a distinguished symbol, #, is attached to some input tree. A node labeled # in an input tree can be matched by any node in the pattern.

Definition 3.1. Let q and t be a pattern and an input tree, respectively. An embedding e: nodes(q)→ nodes(t) is a *trap embedding* if the following conditions hold true:

1 e(root(q))=root(t)
2 For any $n\in$nodes(q), either label(n)=*, or label(n)= label(e(n)), or label(e(n))=#, and
3 For any n_1, $n_2\in$nodes(q)
 a. If there is a C_edge from n_1 to n_2, then
  if label(e(n_1))=#, then e(n_2)=e(n_1)
  else there is an edge from e(n_1) to e(n_2)
 b. If there is a D_edge from n_1 to n_2, then
  if label(e(n_1)=#, then e(n_2)=e(n_1)
  else there is a path from e(n_1) to e(n_2)

The definition differs from that of a normal embedding only when a node is matched to a # node, in which case all its descendants are matched also to that # node. For the nodes in q not matched to the node labeled # in t, a trap embedding is just a normal embedding defined at the beginning of Section 2.2. Similar to the definition for a normal embedding, we call conditions 1, 2 and 3 *root*, *node* and *edge conditions*, respectively. Throughout the remaining of the paper, we will use 'embedding' without a preceding 'trap' to mean a normal embedding.

Example 3.1. Shown in Fig. 6a and b are a pattern and an input tree, respectively. Note node 9 is labeled #. Define e_1

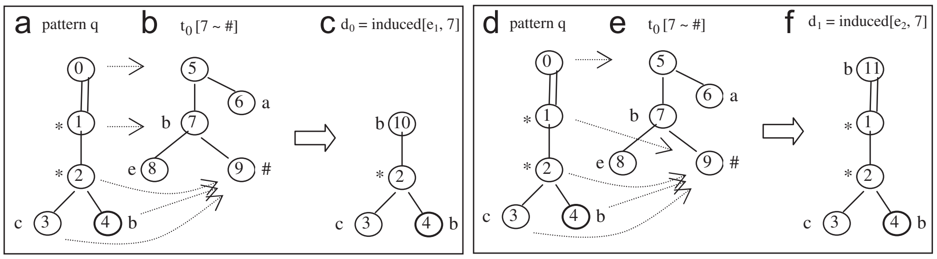

Fig. 6. Trap embeddings and induced patterns.

as $e_1(0)=5$, $e_1(1)=7$, and $e_1(2)=e_1(3)=e_1(4)=9$, and define e_2 as $e_2(0)=5$, $e_2(1)=e_2(2)=e_2(3)=e_2(4)=9$. Both e_1 and e_2 are trap embeddings. They are shown as the dotted arrows on the left and right boxes in Fig. 6, respectively. (Ignore the bold arrows for now.)

A node labeled # in *t* behaves like a trap, which absorbs all the descendents of the node that is first trapped to it. Thus, we call the node labeled # a *trapping node,* and the parent of a trapping node a *trap attach point* (or simply an attach point). We call the nodes that are mapped to a trapping node *trapped nodes.* Trapping nodes and the attach points are independent of embeddings, while the trapped nodes may vary under different trap embeddings. It is always the case that an input tree initially does not contain any trapping node. If it is necessary to do a trap embedding to it, we attach a trapping node to a predetermined attach point. We will use the notation $t[n\sim\#]$ to denote input tree t in which a trapping node has been attached to node *n*. For a convenient abuse of symbols, when confusion is not possible, we will use # to denote both the label for a trapping node and the trapping node itself.

3.1.2. Induced patterns

If a node is trapped and its parent (either C_parent or D_parent) is not, then the entire sub-tree rooted at that node is a maximal sub-tree that has been trapped. For example, for the embedding in Example 3.1, {2, 3, 4} is a maximal sub-tree in *q* that is trapped to node 9 under e_1. In the general case, there may be multiple maximal sub-trees trapped to a single trapping node under any particular trap embedding.

Let *q* be a pattern and *t* be an input tree. Let *r* be the return node in *q*, and *n* be an attach point in *t*. Let *e* be a trap embedding from *q* to $t[n\sim\#]$. We now extract from *q* a pattern, *d*, as follows. First, if $e(r)\neq\#$ and $e(r)\neq n$, let $d=\Phi$, otherwise, create a root for *d*, and let label(root(*d*))=label(*n*). Then for each maximal sub-tree α trapped, let root(α)=*w*. If *w* is a D_child of its parent in *q*, then let w be a D_child of root(*d*), otherwise, let it be a C_child of root(*d*), in *d*.[3] If $e(r)=\#$, then let *r* be the return node of *d*, and if $e(r)=n$, let root(*d*) be the return node of *d*. We call pattern d so constructed an *induced pattern for e w.r.t. n*, and denote it by *induced*[*e*, *n*].

[3] Recall that we allow a node to belong to multiple trees to avoid using isomorphism symbols.

Example 3.2. Shown in Fig. 6c and f are two induced patterns, where e_1 and e_2 are defined in Example 3.1. In *c*, since sub-tree {2, 3, 4} is the maximal trapped sub-tree in *q* under e_1, it has been made also a sub-tree of node 10, the root of induced[e_1, 7]=d_0. Note that since 2 is a C_child in *q*, we let it be a C_child also in d_0. Similarly, in *f*, sub-tree {1, 2, 3, 4} is the maximal trapped sub-tree in *q* under e_2, it becomes also a sub-tree of induced[e_2, 7]=d_1. Since node 1 is a D_child in *q*, it is also a D_child in d_1.

3.1.3. Trap embedding for query rewriting

Query rewriting is defined by the containment of two sets of patterns, and containment in turns is defined based on checking embeddings for all possible input trees. Obviously it is not possible to follow the definition directly and check all possible trees. Instead, we can check a set of representative trees, which should together replace the set of all possible trees. This is the set of canonical models, which is defined based on the view *v* and query *q*. In particular we make use of the following Lemma 3.1, which is a theorem from [12]. We rewrite it here for ease of reference.

Lemma 3.1. *Let q_1 and q_2 be patterns, in which r_1 and r_2 are return nodes, respectively. Let M be the set of canonical models of q_2 for q_1. Then q_1 contains q_2 iff for all $t\in M$, there is an embedding e: nodes(q_1)→ nodes(t), such that $e(r_1)=\pi_t(r_2)$.*

The following lemma will make use of the concept of concatenation of trees. Let *s* and *t* be labeled trees, and $n\in$nodes(*t*). Assume label(root(*s*))=label(*n*), we use $s\oplus_n t$ to denote the tree formed by merging root(*s*) with *n*, where all the children of *n* before the merging remain. We call the new tree the *concatenation of s and t at node n.* This concept is very similar to the concept of concatenation of patterns, except that here we need to mention explicitly the node in t that will be merged with root(*s*), while in the case of concatenating patterns, $c\oplus v$ always means the root of *c* is merged with the return node of *v*.

Lemma 3.2. *Let q be a pattern, where r_q is the return node. Let t and s be trees, and $b\in$nodes(t). Then it is true that:*

1. *If there is a trap embedding f: nodes(q)→ nodes($t[b\sim\#]$) and an embedding g: nodes(d)→ nodes(s) where d=induced[f, b], then e: nodes(q)→nodes($s\oplus_b t$), defined as: e(n)=f(n) if $f(n)\neq\#$, and e(n)=g(n) otherwise, is an embedding such that $e(r_q)=g(r_d)$.*
2. *If there is an embedding e: nodes(q)→nodes($s\oplus_b t$) such that $e(r_q)\in$nodes(s), then f: nodes(q)→nodes($t[b\sim\#]$)*

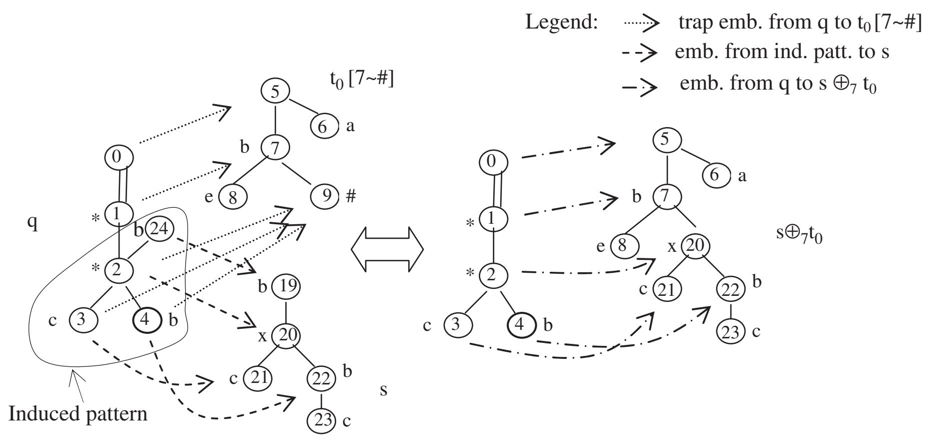

Fig. 7. Diagrammatic explanation of Lemma 3.2.

defined as: f(n)=e(n) if e(n) ∈ nodes(t), and f(n)=# otherwise, is a trap embedding. Let d=induced[f, b] be its induced pattern. Define g: nodes(d)→ nodes(s) as: g(root(d))=root(s) (=b), and g(n)=e(n) for n≠root(d), then g is an embedding, and $g(r_d)=e(r_q)$.

Idea of Proof. From the definition, a trap embedding separates q into two portions, an embedded portion and a trapped portion. The embedded portion is targeted to t, and the trapped portion is targeted to #. From the definition, the induced pattern contains the trapped portion as its first level sub-pattern. For condition 1, the trap embedding is given. Also given is an embedding from the induced pattern. We can create a normal embedding from q to $s\oplus_b t$ by retaining the embeddings from the embedded portion and the first level sub-pattern of the induced pattern. For condition 2, a normal embedding from q to $s\oplus_b t$ is given. We can separate it into an upper and a lower part in such a way that the upper part is embedded to t and the lower part is embedded to s. We then create a trap embedding by retaining the embedding from the upper part, and letting the lower part be trapped. The given normal embedding from the lower part to s can be copied to the induced pattern. The detail of the proof is mainly based on the definitions. (Refer to Appendix A for a formal proof.) □

Lemma 3.2 underlines a crucial theorem later in this section. Its idea is further explained in the following example.

Example 3.3. Shown in Fig. 7 is a pictorial explanation of Lemma 3.2. On the left-hand side of the two-head arrow are two embeddings. The dotted lines denote a trap embedding from pattern q to tree $t_0[7\sim\#]$ (i.e., f in the lemma). The broken lines denote an embedding from the induced pattern to tree s (i.e., g in the lemma). The trap embedding generates an embedded portion $\langle 0, 1\rangle$, and a trapped portion, which is the sub-pattern rooted at node 2. On the right-hand side is a normal embedding from q to $s\oplus_7 t_0$ (i.e., e in the lemma). Suppose the embeddings on the left are given. We can obtain the embedding on the right by retaining the embedding from $\langle 0, 1\rangle$, i.e., the top two dotted lines, and the embedding from the first level sub-pattern of the induced pattern, i.e., the bottom three broken lines. Now suppose the embedding on the right is given. We can obtain the embeddings on the left as follows. We first separate q into an upper part, $\langle 0, 1\rangle$, which is embedded to t_0, and a lower part, the sub-pattern rooted at 2, which is embedded to s. Then we create the trap embedding on the left by retaining the embedding from $\langle 0, 1\rangle$, and letting the sub-pattern rooted at 2 be trapped. The embedding from the induced pattern on the left is formed by copying the bottom four lines on the right.

Until this point, we have required that the root of an induced pattern be labeled the same as that for the attach point, i.e., label(root(induced[e, n]))=label(n). This is appropriate for the purpose of illustrating the concept, i.e., the root of the induced pattern must match the attach point. For a canonical model, however, we need to relax this requirement. This is because, for a view pattern v, the attach point in its canonical model for query pattern q is labeled z if the return node of v is labeled *, where z is a symbol that no node in q has been labeled. Our goal is not just match the symbol. What we actually need is that if a node in our pattern can match z, it can match any symbol. This is also the original motivation for introducing symbol z in [12]. Thus, from now on, if the attach point in the canonical model is labeled symbol z, we will label the root of the induced pattern asterisk.

Theorem 3.3. *Let q and v be patterns, and* r_q *and* r_v *be the return nodes in q and v, respectively. Let M be the canonical model set of v for q. Let c be any pattern. Then* $c\oplus v$ *is a rewriting for q using v, if and only if, for all* $t \in M$, $\{c\}\sqsubseteq_{\clubsuit} P_t$, *where* $P_t=\{\text{induced}[e, \pi_t(r_v)] \mid e: \text{nodes}(q)\rightarrow \text{nodes}(t[\pi_t(r_v)\sim\#]) \text{ is a trap embedding}\}$.

Proof. *only if*: We must prove $P_t\sqsupseteq_{\clubsuit}\{c\}$ for all $t\in M$, given that $q\sqsupseteq_{\clubsuit} c\oplus v$. Let r_c be the return node in c, s be any tree and e: nodes(c)→ nodes(s) be any embedding. Recall π_t: nodes(v)→ nodes(t) is an embedding that maps each node in v to its copy in t. Thus, $e\cup\pi_t$: nodes($c\oplus v$)→ nodes($s\oplus_{\pi_t(r_v)}t$) is an embedding. So there is an embedding

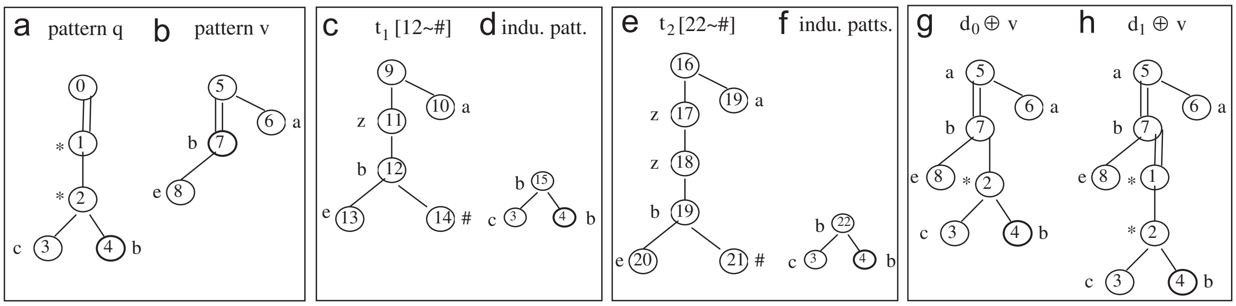

Fig. 8. Induced patterns and concatenated patterns.

y: nodes$(q)\rightarrow$ nodes$(s\oplus_{\pi_t(r_v)}t)$ such that $y(r_q)=e(r_c)\in$ nodes(s). By part 2 of Lemma 3.2, there is a trap embedding f: nodes$(q)\rightarrow$nodes$(t[\pi_t(r_v){\sim}\#])$ and an embedding g: nodes$(d)\rightarrow$ nodes(s), where d=induced[f, $\pi_t(r_v)$]. Furthermore, $y(r_q)=g(r_d)$, where r_d is the return node in d. This implies $e(r_c)=g(r_d)$. Note $d\in P_t$. By Definition 2.2, $P_t\supseteq_{♣}\{c\}$.

if: We need to prove that $q\supseteq_{♣}c\oplus v$, given $\{P_t\}\supseteq_{♣}\{c\}$ for all $t\in M$. Let O be the canonical model set of c for q and N be the canonical model set of $c\oplus v$ for q. Let $h\in N$ be an arbitrary canonical model. Then it must be the case that $h=s\oplus_{\pi_t(r_v)}t$, for some $s\in O$ and $t\in M$. Since $\{P_t\}\supseteq_{♣}\{c\}$, there is a trap embedding f: nodes$(q)\rightarrow$ nodes$(t[\pi_t(r_v){\sim}\#])$ and a g: nodes$(d)\rightarrow$nodes(s) such that $g(r_d)=\pi_s(r_c)$, where d=induced[f, $\pi_t(r_v)$]. By part 1 of Lemma 3.2, there is an embedding e: nodes$(q)\rightarrow$ nodes(h), such that $e(r_q)=g(r_d)=\pi_s(r_c)$. Recall that r_c is also the return node in $c\oplus v$ and, clearly, $\pi_h(r_c)=\pi_s(r_c)$. Thus $e(r_q)=\pi_h(r_c)$. By Lemma 3.1, $q\supseteq_{♣}c\oplus v$. □

Theorem 3.3 is crucial for the following discussions. It tells us that, for the given patterns q and v, if we can find a set of patterns that is contained by every set of the induced patterns for the canonical models of v, then that set of patterns is necessarily a set of compensation patterns. On the other hand, any compensation pattern must be contained by every set of the induced patterns. (Note that 'any' here necessarily includes maximal compensation pattern.) Thus, Theorem 3.3 has the full power for the existence of (maximal) rewriting. However, a direct implementation of the theorem for the purpose of searching for rewriting may not be realistic, since there are $(L+2)^m$ $\vec{u}$-extensions of the view, and for each of them we need to test the containment. Nonetheless, the theorem provides a theoretic basis on which other schemes can be built. In the subsequent sections, we will introduce two such schemes, with varying degrees of powers and efficiencies. We first introduce a corollary that imposes a stronger condition than Theorem 3.3 to avoid checking for containment.

Corollary. *Let c, q and v be patterns, and r_v be the return node in v. Let M be the canonical model set of v for q. If for all $t\in M$, there is a trap embedding e_t nodes$(q)\rightarrow$ nodes$(t[\pi_t(r_v){\sim}\#])$, such that c=induced$[e_t,\pi_t(r_v)]$, then $c\oplus_{r_v}v$ is a rewriting for q using v.*

Proof. The condition implies, for all t, $c\in P_t$. This in turn implies $\{c\}\subseteq_{♣}P_t$. □

The corollary states that if a pattern is a member of every set of the induced patterns, then it is a compensation pattern. Since checking the membership for patterns is easier than checking pattern containment, the corollary provides a simple way to search for compensation patterns in a restricted context.

Example 3.4. Shown in Fig. 8a and b are patterns q and v, respectively. There are three $\vec{u}$-extensions in the canonical model set of v, t_0, t_1, and t_2, where t_0 is depicted in Fig. 6b, and t_1 and t_2 are in Fig. 8c and e. (Note that the diagrams are the $\vec{u}$-extensions plus an extra # node.) All these three $\vec{u}$-extensions have the induced patterns, d_0 and d_1 shown in Fig. 6c and f, respectively. Furthermore, t_1 and t_2 both have an additional induced pattern, as shown in Fig. 8d and f. By the above corollary, $\{d_0\oplus v, d_1\oplus v\}$ is a rewriting for q using v, which are, respectively, shown in Fig. 8g and h.

3.1.4. Algorithms

In this subsection, we introduce a method that searches for rewritings. The method is based directly on Theorem 3.3. We first introduce two functions, GenEmbC(·) and GenEmbD(·). The former generates trap embeddings from a pattern to a tree, and the latter generates those from a pattern to the tree as well as all its descendant sub-trees. This is necessary due to different requirements on C_edges and D_edges in the definition. The main data structure is a two-dimensional array E, with each entry corresponding to a pair of pattern and tree nodes. For a pattern node x and tree node y, $E[x, y]$ is initially *undef*, and will contain a set of trap embeddings from sub-pattern x to sub-tree y or its descendant. We also use two one-dimensional arrays $B[x]$ and $D[x]$ as working variables. For pattern node x, $B[x]$ contains a set of trap embeddings from sub-pattern x. $D[x]$ is a set of sequences of trap embeddings. Each such sequence contains exactly one trap embedding from each child of x. We assume the following initializations:

$E[x, y]$=*undef* for all pattern node x and tree node y.
$B[x]=\Phi$ for all pattern node x.
$D[x]=\{\varepsilon\}$ for all pattern node x.

GenEmbC(PatternNode x, TreeNode y, Embeddings E[x, y])

```
1. if (label(x)≠* and label(x)≠label(y) and label(y)≠#) then
2.    E[x, y]←Φ; return
3. if label(y)=# then
4.    define a new embedding e as: e(x)=y, and e(z)=y for all the descendents z of x, insert e into E[x, y], and return
5. for each child x′ of x
6.    for each child y′ of y
7.       if E[x′, y′]=undef
8.          if x′ is a C_child then GenEmbC(x′,y′, E[x′, y′])
9.       else GenEmbD(x′, y′, E[x′, y′])
10.      insert all members of E[x′, y′] into B[x′]
11.   D[x]←D[x] × B[x′] // the result will be Φ if an operand is Φ
12. if D[x]=Φ then E[x, y]←Φ, and return
13. for each ⟨e_1,…,e_n⟩ ∈D[x] //1,…,n are the ids of the children of x
14.    define a new embedding e_x as: e_x(x)=y, and e_x(z)=e_i(z) if z is in sub-pattern rooted at i
15.    insert e_x into E[x, y]
16. return
```

GenEmbD(PatternNode x, TreeNode y, Embeddings E[x, y])

```
1. if (label(x)≠* and label(x)≠label(y) and label(y)≠#)
2. then go to 16
3. if label(y)=# then
4.    define a new embedding e as: e(x)=y, and e(z)=y for all the descendents z of x, insert e into E[x, y], and return
5. for each child x′ of x
6.       for each child y′ of y
7.          if E[x′, y′]=undef
8.             if x′ is a C_child then GenEmbC(x′,y′, E[x′, y′])
9.          else GenEmbD(x′, y′, E[x′, y′])
10.         insert all members of E[x′, y′] into B[x]
11.      D←D × B[x′]//the result will be Φ if an operand is Φ
12. if D=Φ then go to 16
13. for each ⟨e_1,…,e_n⟩ ∈D
14.    define a new embedding e_x as: e_x(x)=y, and e_x(z)=e_i(z) if z is in sub-pattern rooted at i
15.      insert e into E[x, y]
16. for each child y′ of y
17.         if E[x, y′]=undef then GenEmbD(x, y′, E[x, y′])
18.      insert a copy of E[x, y′] into E[x, y]
19. return
```

Upon return, GenEmbC(x, y, $E[x, y]$) stores into $E[x, y]$ all the trap embeddings from sub-pattern x to sub-tree y, and GenEmbD(x, y, $E[x, y]$) stores in the entry all the trap embeddings from sub-pattern x to sub-tree y or its descendant sub-trees. Line 11 assembles the sequences of the trap embeddings from the children of x. When the control reaches line 12, $D[x]$ contains all possible sequences of trap embeddings from one child of x each. If the test in line 12 evaluates to true, at least one child of x cannot be embedded, and therefore no trap embedding exists from x to y. When the control reaches line 18 in GenEmbD(·), $E[x, y']$ is defined (possibly Φ). Note that an insertion into an undefined array entry always overwrites *undef*. Also note that a recursive call is made only when the corresponding array entry is undefined, indicating no earlier calls have been made to the corresponding pattern and tree nodes. Thus, all the calls are made on different pairs. Let x=root(q) and y=root(t). The total number is $|q| \cdot |t|$. Assuming $k<|q|$, where k is the total number of trap embeddings from q to t, the time it takes to execute GenEmbC(·) is $O(|q| \cdot |t|)$.

After the trap embeddings are generated, we must create the induced pattern for each of them. This is done by the following algorithm.

Induced(Pattern q, Input t, TrapEmbedding e)
{Let r_q be the return node in q, and s be the attach point in t, where a trapping node has been attached.}

```
1. if (e(r_q)≠s and e(r_q)≠#) then return Φ
2. get a node r and let label(r)=label(s)
3. for each n in pre-order traversal of q
4.    if e(n)=# then
5.       if n is a C_child of its parent then
6.          let n be a C_child of r
7.       else let n be a D_child of r
8.       skip all the descendants of n from traversal
9. return pattern rooted at r
```

The algorithm is just a rephrase of the definition for an induced pattern. (Refer to Section 3.1.2.) Since we traverse q only once, the complexity is $O(|q|)$.

Based on the previous functions, the following top level function retrieves all the rewritings for q using v. The following notations are used. M is the set of canonical models of v for q, root$_q$ and root$_t$ denote the roots of q and t, respectively, and r is the return node in v. For each canonical model t of v, we assume # has been attached to $\pi_t(r)$. Variable P_t has been initialized to Φ for all t.

GenRewriting

```
1. for each t∈M
2.    GenEmbC(root_q, root_t, E_t[root_q, root_t])
3.    for each e∈E_t[root_q, root_t]
4.       P_t←P_t∪Induced(q, t, e)
5. C $\bigcap_{t \in M} P_t$
6. for each c∈C
7.    insert $c \oplus_r v$ into R
8. return R
```

The correctness of the above algorithm is easy to see. P_t contains all the induced patterns for each canonical model $t \in M$, and C contains all the induced patterns common to all P_t, $t \in M$. By the corollary of Theorem 3.3, R is a rewriting for q using v.

For the time complexity, we consider the loop only, since it dominates asymptotically. First note that $|M|=(L+2)^m$, where m is the number of D_edges in v and

L is the length of the longest star-path in q. This is the number of calls for GenEmbC($\cdot$) in line 2. Each call to GenEmbC($\cdot$) takes a time of $O(|q|\cdot|t|)$. For each such call, line 4 is executed k times, where k is the number of trap embeddings generated by the call. Each execution of line 4 takes a time of $O(|q|)$. Thus, the total time complexity is $O((L+2)^m\cdot(|q|\cdot|t|+k|q|))$. Assuming $k\ll|q|$, the total time is $O((L+2)^m\cdot|q|\cdot|t|)$.

3.2. Trap relay

3.2.1. F_segment and prefix–suffix matching

A trap embedding is a mapping from a pattern to a tree that allows us to extract a compensation pattern. The exponential overhead results from the need to consider all the canonical models of the view. Does there exist a mapping directly between two patterns that allows extracting compensation patterns? We consider this issue in this subsection. We first discuss the issue for patterns that are paths. We then extend the results to the general case. The following notations and conventions will be used. Greek letters denote paths, and English letters denote nodes. (An exception is q and v, which we will still use for patterns, in order to be consistent with the previous sections.) Let λ and μ be path patterns. We use the notation $\varepsilon(\lambda)=\mu$ to denote mapping ε: nodes(λ)→ nodes(μ) such that ε(start(λ))=start(μ) and ε(end(λ))=end(μ). We define an operator *precede join*, denoted by ◆, as follows. If end(λ)=start(μ), then λ◆$\mu=\gamma$ where γ is a concatenation of λ and μ, but contains only one occurrence of end(λ) or start(μ), otherwise it is undefined. We use the expression of the form (*condition A*) *except (condition B*) to mean that condition A is true with some exception specified by condition B.

Any path pattern is a sequence of C_edges and D_edges. If the two end points of a D_edge are both labeled *, then it can be matched to any tree path. It turns out that a similar property also holds for certain kinds of paths. This implies that there is a need to distinguish different kinds of paths.

Definition 3.2. Let λ be a path pattern. It is called a *C_segment* if it does not contain D_edges, and it is called an *F_segment* if $\forall n\in$nodes(λ): [($n\neq$(start(λ) & $n\neq$end(λ)⇒label(n)=*], and $\exists n_1$, n_2 ∈ nodes(λ): [$\langle n_1, n_2\rangle$ is a D_edge].

An F_segement has an interesting property. Before we describe it, we first introduce a concept.

Definition 3.3. Let $\lambda=\langle a_1,\ldots,a_m\rangle$ be a path pattern, and $t=\langle b_1,\ldots,b_n\rangle$ be a tree path. Let e be an embedding from λ to t. We say e is a *prefix–suffix matching with separating edge* $\langle a_i, a_{i+1}\rangle$, where $1\le i<m$, if $\forall k$, $[(1\le k\le i\Rightarrow e(a_k)=b_k)$ and $(i+1\le k\le m\Rightarrow e(a_k)=b_{n-m+k})]$.

A prefix–suffix matching must first of all be an embedding. It maps the nodes preceding the separating edge to the prefix and those following the separating edge to the suffix, from the path pattern to the tree path. Note a prefix–suffix matching may not always exist. For example, it cannot possibly exist if $|\lambda|>|t|$. If $|\lambda|<|t|$ and $\langle a_i, a_{i+1}\rangle$ is a C_edge, then no prefix–suffix embedding exists with $\langle a_i, a_{i+1}\rangle$ as the separating edge.

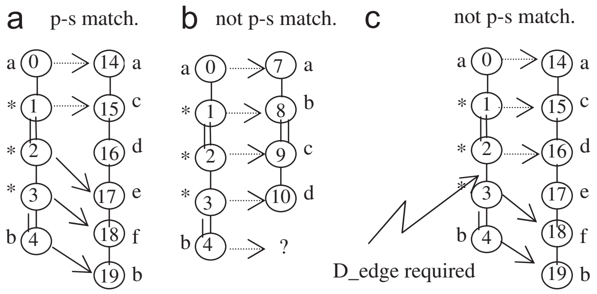

Fig. 9. The idea of prefix–suffix match.

Theorem 3.4. *Let $\lambda=\langle a_1,\ldots,a_m\rangle$ be an F_segment, and $\langle a_i, a_{i+1}\rangle$ be a D_edge, where $1\le i<m$. Let $t=\langle b_1,\ldots,b_n\rangle$ be a tree path. If label(a_1)$\neq$*⇒ label(a_1)=label(b_1), label(a_m)$\neq$*⇒label(a_m)=label(b_n), and $m\le n$, then there is a prefix–suffix matching with separating edge $\langle a_i, a_{i+1}\rangle$ from λ to t.*

Proof. Define e: nodes(λ)→ nodes(t) as: for all $1\le k\le i$, $e(a_k)=b_k$, and for all $i+1\le k\le m$, $e(a_k)=b_{n-m+k}$. We now show e is an embedding. Since $e(a_1)=b_1$, $e(a_m=b_n)$ and $\forall k$, $[1<k<m\Rightarrow$label(ak)=*], the root and node conditions are satisfied. Let $\langle n_1, n_2\rangle\in$ edges(λ). If $n_1=a_k$ for some $1\le k<i$, or $i+1\le k<m$, then $e(\langle n_1, n_2\rangle)=\langle b_k, b_{k+1}\rangle\in$ edges(t), or $e(\langle n_1, n_2\rangle)=\langle b_{n-m+k}, b_{n-m+k+1}\rangle\in$ edges(t). If $n_1=a_i$, then $e(\langle n_1, n_2\rangle=\langle b_i, b_{n-m+i+1}\rangle$. Since $n-m+i+1\ge i+1$, $\langle b_i, b_{n-m+i+1}\rangle$ is a path in t. Thus, e is an embedding. By definition, it is s prefix–suffix embedding. □

Theorem 3.4 states that, for *any* F-segment, as long as it is not longer than a tree path, and its start and end nodes can be embedded, then any D-edge in it is a separating edge for a prefix–suffix matching.

Example 3.5. Consider Fig. 9. The left most diagram shows a prefix–suffix matching with separating edge $\langle 1,2\rangle$. It is clear that the nodes preceding and following that edge are mapped to the prefix and suffix of the tree path, respectively. In addition, the mapping is an embedding. In Fig. 9.b, no prefix–suffix matching exists, since the path pattern is longer than the tree path. (In this case, there does not even exist an embedding.) In Fig. 9c, the mapping is not prefix–suffix matching since the separating edge is not a D_edge. Thus, when we map the nodes preceding and following $\langle 2, 3\rangle$ to the prefix and suffix of the tree path, respectively, we do not have an embedding.

Corollary. *Let λ be an F_segment, μ be a path pattern, and t be a tree path. Assume label(start(λ))$\neq$*⇒ label(start(λ))=label(start(μ)), and label(end(λ))$\neq$*⇒ label(end(λ))=label(end(μ)). If there is a mapping $e_1(\lambda)=\mu$, such that $\forall n_1$, n_2 ∈nodes(λ), $[n_1\prec n_2\Rightarrow e_1(n_1)\prec e_1(n_2)]$,*[4]

[4] The symbol '$\prec$' indicates precedence relation between nodes in a path pattern.

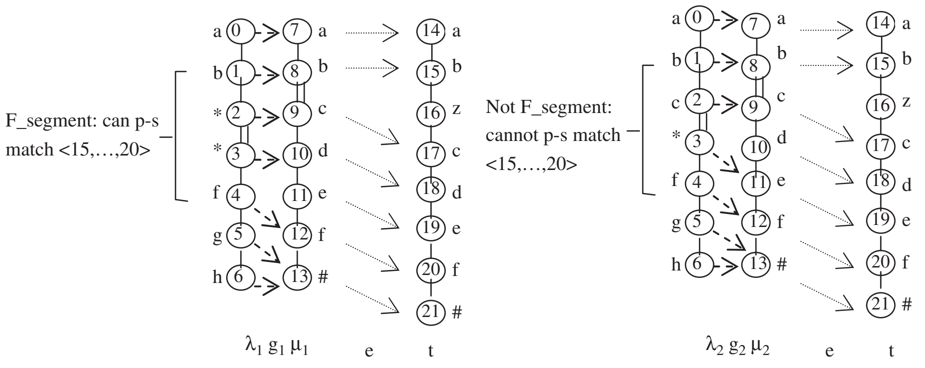
F_segment: can p-s match <15,…,20>
Not F_segment: cannot p-s match <15,…,20>
λ1 g1 μ1
e
t
λ2 g2 μ2
e
t

Fig. 10. The idea of a trap relay.

and an embedding $e_2(\mu)$=t. Then there is a prefix–suffix matching from λ to t.

Proof. For our convention mentioned earlier in this subsection, $e_1(\text{start}(\lambda))=\text{start}(\mu)$ and $e_1(\text{end}(\lambda))=\text{end}(\mu)$. Since e_1 preserves the order of the nodes in λ, $|\lambda|\le|\mu|$. Since e_2 is an embedding, surely we have $|\mu|\le|t|$. Thus $|\lambda|\le|t|$. In addition, label(start(μ))$\neq^*\Rightarrow$ label(start(μ))=label(start(t)), and label(end(μ))$\neq^*\Rightarrow$ label(end(μ))=label(end(t)). Combining this with the assumption in the corollary, we have label(start(λ))$\neq^*\Rightarrow$ label(start(λ))=label(start(t)), and label(end(λ))$\neq^*\Rightarrow$ label(end(λ))=label(end(t)). By Theorem 3.4, the claim follows. □

Note that the mapping e_1 in the above corollary is required to preserve the node order, but is *not* required to preserve the structural information of λ. On the other hand, e_2 is required to be an embedding.

Example 3.6. Consider Fig. 10. There is an F_segment $\langle 1,\dots,4\rangle$ in path pattern λ_1. (Ignore the other segments for now.) In the figure, g_1 is a mapping, $g_1(\langle 1,\dots,4\rangle)=\langle 8,\dots,12\rangle$, which preserves the node order. Also, e is an embedding $e(\langle 8,\dots,12\rangle)=\langle 15,\dots,20\rangle$. This meets the conditions in the above corollary. It is easy to see that there is a prefix–suffix matching from $\langle 1,\dots,4\rangle$ to $\langle 15,\dots,20\rangle$, i.e., nodes 1 and 2 are matched to 15 and 16, respectively, and nodes 3 and 4 are matched to 19 and 20, respectively. Note that g_1 does not preserve the edge structures. For example, both $\langle 1, 2\rangle$ and $\langle 3, 4\rangle$ are C_edges, but $\langle g_1(1), g_1(2)\rangle$ is a D_edge, and $\langle g_1(3),\dots,g_1(4)\rangle$ is a path, and this does not affect the existence of the prefix–suffix matching mentioned above. On the other hand, $\langle 1,\dots,4\rangle$ in λ_2 is not an F_segment. Thus, although g_2 also preserves the node order, there is no prefix–suffix matching from $\langle 1,\dots,4\rangle$ to $\langle 15,\dots,20\rangle$ in this case. However, $\langle 2,\dots,4\rangle$ is an F_segment in λ_2, and there is a prefix–suffix matching from $\langle 2,\dots,4\rangle$ to $\langle 17,\dots,20\rangle$, i.e., 2 matches 17, and 3 and 4 match 19 and 20, respectively.

Since a prefix–suffix mapping is also an embedding, and t is an arbitrary tree path, the above corollary implies λ contains μ. (View end(λ) and end(μ) as return nodes.) A crucial point here is, *this containment is ensured without requiring the structural information of λ to be preserved in μ.* This property of an F_segment plays a pivotal role in the method for rewriting to be introduced in the next section.

3.2.2. Definition of trap relay

Our intention is to have a kind of mapping defined directly between a query and view that allows us to extract compensation patterns, and at the same time imposes as weak a requirement as possible. As such, a mechanism like homomorphism is not appropriate, since it has a strong requirement that the edge structures of the query be preserved. On the other hand, with the concept of F_segment, we can attain our goal by incorporating it into such a mapping.

Definition 3.4. Let λ be a path pattern where $|\lambda|\ge 2$, and μ be a path pattern, such that $\forall n \in$ nodes(λ) $\cup$ nodes(μ): [$n\neq$end(μ))$\Rightarrow$label(n)$\neq$#]. Then $\varepsilon(\lambda)$ *is a trap relay*, if $\varepsilon(\lambda)=\mu$, and $\forall n \in$ nodes(λ): [label(n)=* or label(n)=label($\varepsilon(n)$) or label($\varepsilon(n)$)=#], and

1. start(μ)=end(μ)=#, or
2. start(μ)$\neq$#, λ and μ are C_segments, $|\lambda|=|\mu|$, and $\forall n_1, n_2 \in$ nodes(λ): [$\langle n_1, n_2\rangle$ is an edge $\langle\varepsilon(n_1), \varepsilon(n_2)\rangle$ is an edge], or
3. start(μ))$\neq$#, λ is an F_segment, and $\forall n_1, n_2 \in$ nodes(λ): [$n_1\prec n_2\Rightarrow\varepsilon(n_1)\prec\varepsilon(n_2)$], or
4. $\lambda=a_1\lambda_1a_2\lambda_2a_3$ and $\mu=b_1\mu_1b_2\mu_2b_3$ where λ_1, λ_2, μ_1, and μ_2 are path patterns, and a_i and b_i are nodes, such that $\varepsilon(a_1\lambda_1a_2)=b_1\mu_1b_2$, $\varepsilon(a_2\lambda_2a_3)=b_2\mu_2b_3$, and both $\varepsilon(a_1\lambda_1a_2)$ and $\varepsilon(a_2\lambda_2a_3)$ are trap relays

Call base cases 1, 2, and 3 trap matching, C_segment matching and F_segment matching, respectively.

For trap matching, μ is degenerated to a single node, which is labeled #. For C_segment matching, each C_edge in λ is mapped to a C_edge in μ. For F_segment matching, we only require the mapping to preserve the order of the source nodes. Case 4 is a recursive statement, which essentially states that λ is composed of sub-paths that fall into the three base cases. We exclude the case where $|\lambda|=1$ from the definition only for purpose of simplifying presentation. (A query pattern containing only a singleton root is uninteresting, while including this into our model requires a separate, but non-sophisticated, treatment.)

Note that since any path pattern in $XP^{[/,\ //,\ [\cdot],\ *]}$ is a sequence of C_edges and D_edges, which are just special cases of C_segments and D_segments, respectively, for any pair of path patterns λ, $\mu \in XP^{[/,\ //,\ [\cdot],\ *]}$ (where only the last node in μ can possibly be labeled #), and mapping $\varepsilon(\lambda)=\mu$, ε is either a relay or is not a relay. Thus, Definition 3.4 is applicable for path patterns in the entire class $XP^{[/,\ //,\ [\cdot],\ *]}$. This definition will be extended to the tree patterns in Definition 3.5.

Example 3.7. In Fig. 10, g_1 is a trap relay from λ_1 to μ_1. To see why, we decompose λ_1 into $\langle 0, 1\rangle$, $\langle 1,\ldots,4\rangle$, $\langle 4, 5\rangle$ and $\langle 5, 6\rangle$. We have $g_1(\langle 0, 1\rangle)=\langle 7, 8\rangle$, $g_1(\langle 1,\ldots,4\rangle)=\langle 8,\ldots,12\rangle$, $g_1(\langle 4, 5\rangle)=\langle 12, 13\rangle$, and $g_1(\langle 5, 6\rangle)=\langle 13\rangle$. These four sub-mappings meet respectively conditions 2, 3, 2 and 1 in Definition 3.4. On the other hand, g_2 is not a trap relay, since we cannot decompose λ_2 in the way specified by Condition 4. Note that decomposing λ_2 in the same way as decomposing λ_1 does not work, since $\langle 1,\ldots,4\rangle$ is not an F_segment any more here. On the other hand, decomposing it into $\langle 0,\ldots,2\rangle$, $\langle 2,\ldots,4\rangle$, $\langle 4, 5\rangle$, and $\langle 5, 6\rangle$ does not work either. Although $\langle 2,\ldots,4\rangle$ is an F_segment in λ_2, $\langle 1, 2\rangle$ now is part of a C_segment, and therefore should meet condition 2, which it does not.

3.2.3. *A characterization of trap relay*

Abstractly, trap relay is similar to the matching proposed in [2], but in an opposite direction. The matching there is designed to test the existence of a complete rewriting using view. Thus, it requires each component in the view to contain its target in the query. This is necessary since otherwise we would not be able to retrieve a complete answer from the view. In our context, we require each edge pattern, i.e., C_segment or F_segment in the query to contain its target in the view. This will make it possible to extract compensation pattern from using trap relay. The main differences are as follows: (1) The matching introduced in [2] does not contain a mechanism that can generate a pattern directly for rewriting purpose. An entirely separate algorithm must be developed to deduce such a pattern. On the other hand, trap relay can automatically generate a compensation pattern directly for rewriting purpose, thanks to its trapping capability. (Refer to the discussion below.) (2) The matching in [2] is based on homomorphism, which requires the edge structures in the view to be matched by the query. In trap relay, an F_segement requires only the relative orders of its nodes to be preserved in the target, which is a strictly weaker requirement than homomorphism.

We will show how we extract compensation patterns from a trap relay. We use the same notations as those for the trap embeddings, such as trapping node, trapped node, attach point, etc. Also, the notion of induced pattern is defined similarly as that for trap embedding. The main result we will establish is: *any induced pattern from a trap relay is a compensation pattern.*

In the following, we assume any path initially does not contain a trapping node. For path μ (i.e., either path pattern or tree path), we use $\mu[b\sim\#]$ to denote that a trapping node has been attached to the last node b in μ. For example, in Fig. 10, 13 is a trapping node, and 12 is the attach point for μ_1. Nodes 5 and 6 are trapped nodes, $\mu_1=\langle 7,\ldots,12\rangle$ and $\mu_1[12\sim\#]=\langle 7,\ldots,13\rangle$. The induced pattern for g_1 is the path pattern $\langle 4, 5, 6\rangle$.

Lemma 3.5. *Let λ and μ be two path patterns. Assume $\varepsilon(\lambda)=\mu[b\sim\#]$, and $\varepsilon(\lambda)$ is a trap relay. Let t be a tree path, and $e(\mu[b\sim\#])=t[c\sim\#]$ be an embedding.*[5] *Then (1) there is a trap embedding $f(\lambda)=t[c\sim\#]$, and (2) let d_1=induced[ε, b] and d_2=induced[f, c]. Then ($d_1=d_2$) except (label (root(d_1))$\neq *\Rightarrow$ label(root(d_1))=label(root(d_1))).*

The above lemma states that for any tree path $t[c\sim\#]$ to which $\mu[b\sim\#]$ has an embedding, λ also has an embedding to it whose induced pattern is identical to that induced by the trap relay. This lemma establishes a basis for the correctness of trap relay, i.e., the induced pattern from a trap relay is indeed a compensation pattern.

Idea of Proof. If $\varepsilon(\lambda)$ is either a C_segment matching, or a trap matching, then let $f=\varepsilon\circ e$. If $\varepsilon(\lambda)$ is an F_segment matching, then let f be the prefix–suffix matching. If $\varepsilon(\lambda)$ is a sequence of the above three kinds of matching, then let f be the union of the trap embedding formed for each individual kind of matching as outlined above. (Note a trap matching can occur only at the end of this sequence.) (A formal proof is found in Appendix B.) □

Example 3.8. We have seen from Example 3.7 that g_1 is a trap relay from λ_1 to μ_1. It consists of two C_segment matching: $g_1(\langle 0,1\rangle)=\langle 7,8\rangle$ and $g_1(\langle 4,5\rangle)=\langle 12,13\rangle$, one F_segment matching: $g_1(\langle 1,\ldots,4\rangle)=\langle 8,\ldots,12\rangle$, and one trap matching: $g_1(\langle 5,6\rangle)=\langle 13\rangle$. Observe e is an embedding from μ_1 to t. We can then construct a trap embedding f:(λ_1)=t as follows. For nodes 0, 1, 4, 5, 6, let $f=g_1\circ e$. For nodes 1, 2, 3, 4, let f be the prefix–suffix matching from $\langle 1,\ldots,4\rangle$ to $\langle 15,\ldots,20\rangle$ with separating edge $\langle 2,3\rangle$, i.e., $f(1)=15$, $f(2)=16$, $f(3)=19$, and $f(4)=20$. Clearly, f so defined is a trap embedding from λ_1 to t. Furthermore, the induced pattern for g_1 and that for f are identical, i.e., both are the path pattern $\langle 4, 5, 6\rangle$. Now, consider g_2. We have shown that it is not a trap relay. Note μ_2 is identical to μ_1. Thus e is also an embedding from μ_2 to t. But we cannot claim that there is a trap embedding from λ_2 to t. (In fact, none exists.)

We will establish our central result, i.e., the induced pattern for a trap relay is a compensation pattern, in a more general setting to be introduced below.

A trap relay is defined over paths. To apply the concept to a general tree pattern, we can view a pattern as a collection of *blocks*. A block is a path delimited by two fork nodes, or a fork node and a leaf node. Thus, any node in a block does not branch, except for possibly its start and the

[5] By our convention, this is a normal embedding. Thus, it should treat # as a normal label, rather than a trapping label.

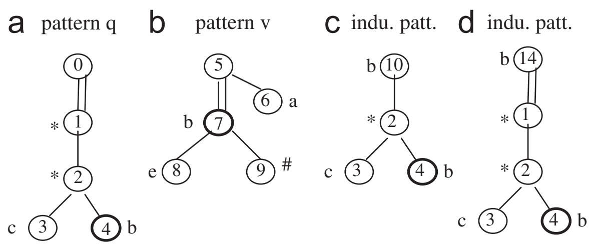

Fig. 11. Induced patterns for trap relays.

end nodes. For any given tree pattern, the collection of the blocks is uniquely determined.

Definition 3.5. Let q and v be patterns, r be the return node in v, and S be the set of the blocks in q. Let g: nodes(q)→ nodes($v[r\sim\#]$) be a mapping such that g(root(q))=root($v[r\sim\#]$). Then g is a *trap relay from q to* $v[r\sim\#]$ if for all $s \in S$, $g(s)$ is a trap relay.

Let g: nodes(q)→ nodes($v[r\sim\#]$) be a trap relay. Its induced pattern, denoted as *induced*[g, r] is formed in exactly the same way in which an induced pattern for a trap embedding is formed. (Refer to Section 3.1.2.) For the induced pattern for the trap relay on full tree patterns, we have similar results to Lemma 3.5.

Lemma 3.6. *Let q and v be patterns, where $v \neq \Phi$. Let r_1 be the return node in v. Assume g: nodes(q)→ nodes($v[r_1\sim\#]$) is a trap relay. Let t be a tree and e: nodes($v[r_1\sim\#]$)→ nodes($t[r_2\sim\#]$) be an embedding. Then there is a trap embedding f: nodes(q)→ nodes($t[r_2\sim\#]$) such that (1) for each block λ in q, $f(\lambda)=e(g(\lambda))$, and (2) let induced[g, r_1]=d_1, and induced[f, r_2]=d_2, then (d_1=d_2) except (label(root(d_1))≠*⇒ label(root(d_1))=label(root(d_2))).*

Idea of Proof. By Lemma 3.5, for each block λ in q, there exists a trap embedding $f_\lambda(\lambda)=e(g(\lambda))$. Let f be the union of the embeddings for all the individual blocks. We can show f is a trap embedding from q to $t[r_2\sim\#]$. Condition 1 is trivial. For condition 2, we show that a path α belongs to induced[g, r_1] if and only if a path β belongs to induced[f, r_2] such that (α=β) except (label(root(α))≠*⇒ label(root(α))=label(root(β))). (For a formal proof, see Appendix B.) □

Based on Lemma 3.6, we have our main result in the following theorem.

Theorem 3.7. *Let q and v be patterns, and r be the return node in v. Assume g: nodes(q)→ nodes($v[r\sim\#]$) is a trap relay, and c=induced[g, r]. Then $c \oplus v$ is a rewriting for q using v.*

Proof. Let M be the canonical model set of v for q, and $t \in M$ be an arbitrary $\vec{u}$-extension. Since π_t: nodes($v[r\sim\#]$)→ nodes($t[\pi_t(r)\sim\#]$) is an embedding, by Lemma 3.6, there is a trap embedding f: nodes(q)→ nodes($t[\pi_t(r)\sim\#]$). Let induced[f, $\pi_t(r)$]=d. Then (c=d) except (label(root(c))≠*⇒ label(root(c))=label(root(d))). Note that label(root(c))=* implies label(r)=*, implying label($\pi_t(r)$)=z. This means label(root(d))=*. (Refer to the illustration preceding Theorem 3.3.) Thus c=d. Since $d \in P_t$, and t is any canonical model, $c \in \bigcap_{t \in M} P_t$. By the corollary of Theorem 3.3, the claim follows. □

Example 3.9. Consider pattern q in Fig. 11a. There are three blocks in q: ⟨0, 1, 2⟩, ⟨2, 3⟩, and ⟨2, 4⟩. Fig. 11b is a view pattern, with a # node attached to 7. We define e(0)=5, e(1)=7, and e(2)=e(3)=e(4)=#. We have e(⟨0, 1, 2⟩) is an F_segment matching, and e(⟨2, 3⟩) and e(⟨2, 4⟩) are trap matching. (Refer to Definition 3.4.) Thus, e is a trap relay from q to $v[7\sim\#]$. Its induced pattern is d_0, depicted in Fig. 11.c. Similarly, there is another trap relay e' from q to $v[7\sim\#]$: e'(0)=5, and e'(1)=e'(2)=e'(3)=e'(4)=#, with induced pattern d_1 in Fig. 11d. We have already seen that both $d_0 \oplus_7 v$ and $d_1 \oplus_7 v$ are rewritings for q using v. (Refer to Example 3.4.)

3.2.4. Searching for trap relays

Theorem 3.7 in the previous section implies that, if we can find a trap relay, then its induced pattern is necessarily a compensation pattern, and therefore we can concatenate it with the view to form a rewriting. The question now is, how we can find a trap relay in the first place. The following theorem provides a simple approach.

Theorem 3.8. *Let q and v be query patterns, and t be any $\vec{u}$-extension of v for q. Let r be the return node in v. If g: nodes(q)→ nodes($v[r\sim\#]$) is a trap relay, then there is a trap embedding f: nodes(q)→ nodes($t[\pi_t(r)\sim\#]$) such that for all block λ in q, $f(\lambda)=\pi_t(g(\lambda))$.*[6]

Proof. Recall π_t is a trivial embedding that maps each node in v to its copy in t. The claim follows directly from condition 1 in Lemma 3.6 by setting e there to π_t. □

The above theorem suggests that, to search for all the trap relays from q to $v[r\sim\#]$, we can first find the set of all the trap embeddings from q to any given $\vec{u}$-extension of $v[r\sim\#]$. We therefore use the simplest one, the 0-extension. For each such trap embedding f, we keep track of $f(\lambda)$ for each block λ in q. Then we identify $\pi_0^{-1}(f(\lambda))$, which is simply a copy of $f(\lambda)$ in $v[r\sim\#]$. We can then check if this mapping meets the conditions in Definition 3.3.

Since any pattern and its 0-extension are identical in shape, in the physical implementation, we can conveniently use a single tree for both the view and its

[6] π_t here should be understood as also mapping the # node in $v[r\sim\#]$ to the # node in $t[\pi(r)\sim\#]$.

0-extension. Initially, the tree plays the role of the 0-extension, to which we search for trap embeddings from q. This task is fuled by using function GenEmbC($\cdot$) in Section 3.1.4. Then for each embedding found, we test if it is a trap relay from q to the view. It is a trap relay if and only if for all the blocks of q, it is a trap relay. For each trap relay, we generate its induced pattern, which is then concatenated with the view to form a rewriting. The following is the pseudo-code for the function described above.

UseRelay(Query q, View v)

```
1. GenEmbC(root$_q$, root$_v$, E)
2. for each e $\in$ E
3.    if for all block $\alpha$ of q, IsRelay($\alpha$, e)=true
4.    then insert e to R
5. for each e$\in$R
6.    insert Induced(q, t$_0$, e) to I
7. for each c$\in$I
8.    insert $c\oplus v$ into W
9. return (W)
```

In the above function, IsRelay(α, e) is true if and only if e is a trap relay when it is restricted to α. Its pseudo-code is shown below.

IsRelay(Block α, TrapEmb e)

```
1. $\langle$n$_1$, n$_2\rangle\leftarrow$next edge from $\alpha$
2. if $\langle$n$_1$, n$_2\rangle$=NULL then return yes
3. if e(n$_1$)=# then return yes
4. find the longest segment $\lambda$ starting from n$_1$ in which
   all the inner nodes are labeled *
5. if $\exists$n$\in$nodes($\lambda$): [e(n)=#] then n$\leftarrow$ first node in $\lambda$ such
   that e(n)=#
6. else n$\leftarrow$end($\lambda$)
7. if $\langle n_1,\ldots, n\rangle$ is a C_segment & $\exists\langle a_1, a_2\rangle\in$edges($\langle$
   $n_1,\ldots,n\rangle$): [ < e($\langle a_1, a_2\rangle$) is a D_edge] then return no
8. else $\langle n_1, n_2\rangle\leftarrow$ the edge following n; go to 2
```

We scan block α and check if it can be decomposed in a way specified in condition 4 of Definition 3.3. In line 3, if the test evaluates to true, then all the following nodes will be mapped to # by e. Thus condition 1 is true, and $e(\alpha)$ is a trap relay. If the test in line 7 evaluates to true, $e(\langle n_1,\ldots,n\rangle)$ contradicts all three conditions 1, 2, and 3 in Definition 3.3. When control reaches line 8, $\langle n_1,\ldots,n\rangle$ is either an F_segment, or a C_segment in which no edge is matched to a D_edge. The former corresponds to condition 3, and the latter corresponds to condition 2. Thus we continue the process for the suffix of α following $\langle n_1,\ldots,n\rangle$.

For the time complexity of *UseRelay*(*), with $O(|q|\cdot|E|\cdot|v|)$, we can obtain the set E of all the trap embeddings from q to the 0-extension of v. With $O(|q|)$, we can retrieve the set of all the blocks in q. By storing necessary information while scanning, lines 4–7 can be implemented with a single scanning. Thus the amount of time for each call to IsRelay(.) is $O(|\alpha|\cdot|e(\alpha)|)$. This means determining whether or not e is a relay for pattern q takes $O(|q|\cdot|v|)$, and hence checking all the trap embeddings for relays takes a total of $O(|q|\cdot|v|\cdot|E|)$. Thus, the total time complexity for finding all trap relays from q to $v[r\sim\#]$ is $O(|q|)+O(|q|\cdot|v|)+O(|q|\cdot|v|\cdot|E|)=O(|q|\cdot|v|\cdot|E|)$. Clearly, this performance is superior to that for *GenRewriting*(*), since its complexity depends mainly on $|E|$, the number of trap embeddings from q to a single canonical model (i.e., the 0-extension) of v, while the complexity of *GenRewriting*(*) depends not only on the number of canonical models of v, but also on the number of trap embeddings from q to all these canonical models. As mentioned in Section 2.2, the number of canonical models of v is an exponential in the number of D-edges in v, which in the worse case can be $|v|-1$ (i.e., all the edges in v are D-edges). The price paid for that performance for *UseRelay*(*), however, is the stronger condition used by trap relay than that in the corollary of Theorem 3.3. This point follows directly from Theorem 3.8. For any $\vec{u}$-extension t of v, for each trap relay g from q to v, there is a trap embedding f from q to t whose target nodes are copies of the target nodes of the trap relay. This implies the induced pattern of g is also the induced pattern of f. Since t is an arbitrary $\vec{u}$-extension of v, the condition in the corollary of Theorem 3.3 is true. However, the reverse is not always true, i.e., an induced pattern shared by all the $\vec{u}$-extensions of v is not necessarily an induced pattern of any trap relay from q to v. (It is worth noting here that, although it is more efficient than the GenRewriting(*), the algorithm *UseRelay*(*) is an exponential algorithm in nature. This is because, in the general case, $|E|$ is exponential.)

3.3. Maximality

The methods we have discussed so far consider how to generate rewritings. In the general case these are not necessarily maximal. However, under some conditions, the maximality holds. In this section, we introduce these conditions. We use the phrase 'selection path' to refer to the path that starts from the root and ends at a return node in any pattern. Note that, for any pattern, there is a unique selection path.

Theorem 3.9. *Let M be the set of canonical models of v for q, and r be the return node in v. If there is a $t_0\in M$, such that there is a unique trap embedding e: nodes(q)$\rightarrow$ nodes($t_0[\pi_t(r)\sim\#]$), then any non-empty rewriting generated by the corollary of Theorem 3.3 is maximal.*

Proof. The 'if' clause implies $|P_{t_0}|=1$ where P_t is defined in Theorem 3.3. Let $P_{t_0}=\{c_0\}$. Let $c\oplus v$ be a rewriting generated by the corollary, i.e., for all $t\in M$, $c\in P_t$. Since $c\in P_{t_0}$, $c=c_0$. Now, let $c'\oplus v$ be any rewriting. By Theorem 3.3, for all $t\in M$, $\{c\}\subseteq_{\clubsuit} P_t$. Thus, $\{c'\}\subseteq_{\clubsuit}\{c_0\}$. This means $c_0\oplus v$ is maximal. □

The condition in Theorem 3.9 does not constrain the structures of the patterns, and hence has some degree of flexibility. However, if the corollary returns an empty set, this does not necessarily mean there does not exist a rewriting. The following theorem is motivated by the observation that the asterisk symbols and descendant edges are the two most flexible structures for a pattern. If

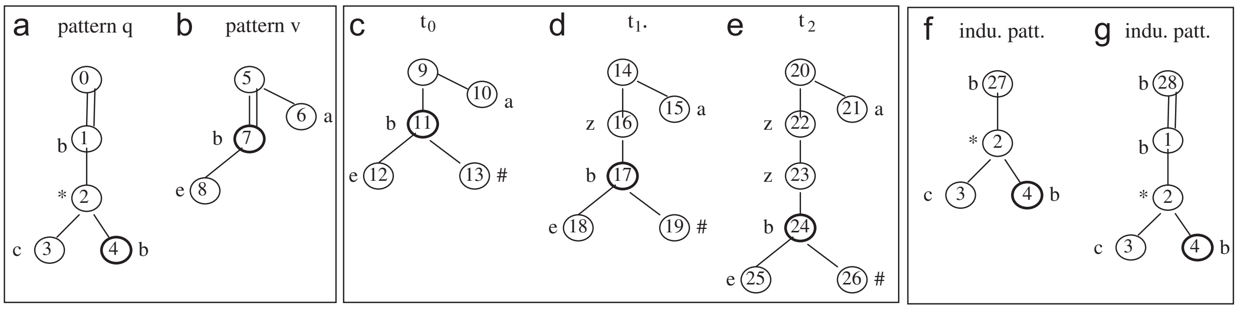

Fig. 12. Generating maximal rewriting.

we can somehow separate them from forming structurally a joint-force, then they can become more manageable.

Theorem 3.10. *The set of rewritings generated by the corollary of Theorem 3.3 is maximal if the following conditions hold true for pattern q: (1) any node labeled * is not incident with a D_edge, and (2) if a leaf is labeled *, then its parent is not.*

Idea of Proof. We consider a special canonical model of v for q, t_{L+1}, where L is the length (in nodes) of the longest star-path in q. We can show that for any trap embedding e from q to t_{L+1}, every node in q must be matched to a non-rubber node in t_{L+1}, except for some leaf node, which may be matched to a rubber node. Note that non-rubber nodes in t_{L+1} are copies of the nodes in v. Based on this fact, for any other canonical model t of v for q, we can form a trap embedding f from q to t by simulating e, i.e., we map each node of q to the node in t that is the copy of the same node in v for e. Then we can show that the induced pattern for t_{L+1} under e is also an induced pattern for t under f. By the corollary of Theorem 3.3, $P_{t_{L+1}}$, the set of all the induced patterns for t_{L+1}, is a set of compensations. Then by the theorem itself, any other compensation is contained (i.e., pattern containment) by $P_{t_{L+1}}$. Thus, the compensations in $P_{t_{L+1}}$correspond to a maximal rewriting. (Refer to Appendix C for a formal proof.) □

The above theorem is true whether the corollary returns an empty set or not. That is, if no pattern meets the condition of the corollary, then there does not exist a rewriting. Also, from the proof of the theorem, to obtain a maximal rewriting, we need to consider a single canonical model of v for q only, i.e., t_{L+1}. This can be done in polynomial time. Note that if no nodes in q are labeled *, then the two conditions in the theorem are trivially true. Thus we immediately have the following.

Corollary. *If q belongs to the sub-class $XP^{[/,\ //,\ []]}$, then the rewriting generated by the corollary of Theorem 3.3 is maximal.*

The result stated by the corollary is weaker than restricting the entire problem to the sub-class $XP^{[/,\ //,\ []]}$. On the other hand, if we do restrict both p and q to that subclass, we will have a stronger result, which will be introduced shortly.

Example 3.10. Consider Fig. 12. The only node labeled * in q is 2, which is not incident with D_edges. Thus the condition in Theorem 3.10 is met. Let us explain some points in our proof to see why the claim in the theorem is true. Since the longest star-path contains only one node, we have $L=1$. Thus there are three canonical models, depicted in 12c, 12d and 12e. We consider the 2-Extension, t_2. The rubber path is $\langle 20,\ 24\rangle$. We can see that no node in q can be matched to the two inner nodes, 22 and 23. Consider a trap embedding e_1 from q to $t_2[24{\sim}\#]$ defined as: $e_1(0)=20$, $e_1(1)=24$, $e_1(2)=e_1(3)=e_1(4)=\#$. This results in the induced pattern depicted in Fig. 12.f. Now, consider t_1. From trap embedding e_1, we have trap embedding e'_1 from q to t_1: $e'_1(0)=14$, $e'_1(1)=17$, $e'_1(2)=e'_1(3)=e'_1(4)=\#$, where 14 and 20 are copies of 5, 17 and 24 are copies of 7, etc. Thus, e'_1 results in same induced pattern as that in 12f. This can be repeated for t_0. This means the induced pattern in 12f is shared by all the canonical models. The same can be said for the induced pattern in 12g. Thus, concatenating 12f and 12g with v, we get the maximal rewriting for q using v.

The above results are related to trap embeddings. They are not universally applicable to the concept of trap relays. This is because the latter has weaker modeling power than the former. To make the maximality also hold for relay, we need to make the conditions stronger.

Theorem 3.11. *The set of all the rewritings generated by Theorem 3.7 is maximal if both q and v belong to subclasses $XP^{[/,\ *,\ [\cdot]]}$ or $XP^{[/,\ //,\ [\cdot]]}$.*

Proof for $XP^{[/,\ *,\ [\cdot]]}$. Let r_v be the return node in v. Since v does not contain D_edge, any canonical model of v for q does not contain rubber path. This means there is exactly one canonical model t of v, which is identical to v in shape. Let P_v={induced[f, r_v]|f is a trap relay from q to $v[r_v{\sim}\#]$}, and P_t={induced[e, $\pi_t(r_v)$]|e is a trap embedding from q to $t[\pi_t(r_v){\sim}\#]$}. Let e: nodes(q)→nodes($t[\pi_t(r_v){\sim}\#]$) be a trap embedding. We define a mapping f: nodes(q)→ nodes($v[r_v{\sim}\#]$) as follows. For all $n \in$ nodes(q), let $f(n)=\pi_t^{-1}(e(n))$ if $e(n)\neq\#$, and $f(n)=\#$ otherwise. We prove that f is a trap relay. First, e(root(q))=root(t)$\neq\#$. Thus f(root(q))$=\pi_t^{-1}(e(root(q))=\pi_t^{-1}(root(t))=root(v)$. Second, assume $n \in$ nodes(q)–{root(q)}. If $e(n)\neq\#$, then label(n)$\neq$* implies label(n)=label($e(n)$)$\neq z$. Thus label($e(n)$)= label($\pi_t^{-1}(e(n))$)=label($f(n)$). If $e(n)=\#$, we have $f(n)=\#$.

Let $\langle n_1, n_2\rangle$ be a C_edge in q. Assume $e(n_1)\neq\#$. Then $f(n_1)=\pi_t^{-1}(e(n_1))\neq\#$. If $e(n_2)\neq\#$, then $\langle e(n_1), e(n_2)\rangle$ is an edge in t, imply $\langle \pi_t^{-1}(e(n_1)),\pi_t^{-1}(e(n_2))\rangle$ is a C_edge in v. If $e(n_2)=\#$, then $e(n_1)=\pi_t(r_v)$ and $f(n_2)=\#$. Thus $f(n_1)=\pi_t^{-1}(e(n_1))=\pi_t^{-1}(\pi_t(r_v))=r_v$. meaning $\langle f(n_1), f(n_2)\rangle$ is a C_edge in v. Now assume $e(n_1)=\#$. Then $e(n_2)=\#$, and $f(n_1)=f(n_2)=\#$. Thus f is a trap relay. Note that for any node n, $f(n)=\#$ if and only if $e(n)=\#$. Thus induced $[f, r_v]$=induced$[e, \pi_t(r_v)]$. This means induced$[e, \pi_t(r_v)]\in P_v$. Since e is any trap embedding, we have $P_t\subseteq P_v$. Let c be any compensation for q using v. By Theorem 3.3, $\{c\}\subseteq_{\clubsuit} P_t$. Thus $\{c\}\subseteq_{\clubsuit} P_v$. This means P_v is maximal □.

Proof for XP$^{[/,//,[\cdot]]}$. In this case, a trap embedding cannot map the nodes in q to the rubber nodes in any canonical model t of v. We define P_v and P_t the same way as in the case for XP$^{[/,*,[\cdot]]}$. The wording of the argument is almost identical to that case. The only exception is that following each occurrence of word 'C_edge', insert a '(D_edge)', and following each occurrence of word 'edge', insert a '(path)'. Also, the sentence 'If $e(n_2)=\#$, then $e(n_1)=\pi_t(r_v)$ and $f(n_2)=\#$. Thus $f(n_1)=\pi_t^{-1}(e(n_1))=\pi_t^{-1}(\pi_t(r_v))=r_v$.' should be replaced by 'If $e(n_2)=\#$, then $e(n_1)=(\prec=)\pi_t(r_v)$ and $f(n_2)=\#$. Thus $f(n_1)=\pi_t^{-1}(e(n_1))=\pi_t^{-1}(\pi_t(r_v))=(\prec=)r_v$.' □

4. Related work

Query containment for XML queries has been studied extensively. As a result, significant results have been generated. In [1,12,23], it has been shown that the containment problem for XPath queries in class XP$^{[/,//,[\cdot],*]}$ is coNP-hard and is in P when the problem is restrict to the subclasses XP$^{[/,//,[\cdot]]}$, or XP$^{[/,//,*]}$, or XP$^{[/,[\cdot],*]}$. The strength and weakness of homomorphism-based and canonical model-based methods are discussed in detail in [12]. In [15,23], the authors further extend the results to the case where disjunctions, DTDs and some limited predicates are allowed.

Query rewriting using views has been studied extensively for relational databases [9,11,14,17,18]. As a result, the technologies, both in theoretical foundations and in methodologies, are maturing. A comprehensive survey for the problems and solutions for relational databases are given in [9]. In [3,7,13,16], the authors study the query rewriting problem in a general context of semi-structured data. Several works have studied this issue for XML data in specific contexts recently. Most of them study the complete rewritings [2,4,6,20,21,22]. Among these works, the ones closest to ours in paradigm, i.e., based directly on query containment, are in [2,21,22]. In [2], the authors introduce an algorithm that systematically matches all the nodes in a view to those in the query. A successful matching signifies the existence of a complete rewriting using the view. The information collected from the matching phase is then used to generate rewriting. Their method of matching is based on the concept of homomorphism introduced in [12]. In [22], the authors present some theoretic as well as algorithmic approaches for the problem of query rewriting using views. They introduce the idea of concatenation of a compensation pattern and a view. This simplifies the analysis of the problem without compromising the generality. Their method is complete for the three subclasses of XP$^{[/,//,[\cdot],*]}$. Our previous work in [21] discussed some theoretic aspects relating to how to determine the existence of a query rewriting using views for multiple return nodes. It nonetheless is based on a simplified model where a view itself serves as a rewriting. Thus, how to generate a rewriting is not an issue. The only work we know of that study incomplete rewriting is in [10]. Their key concepts, 'useful embedding' and 'clip-away tree', are the counter parts of our 'trap embedding' and 'induced pattern'. (The following comparison is for purpose of clarifying the differences between the two methods, and not meant to be critical of the method in [10]. The authors in [10] dealt with a context without wildcards, as such it was not their intention to circumvent the problems caused by wildcards.) Useful embedding is based on homomorphism [12], and is defined from a query to a view. It requires the latter to preserve the structural information of the former. This requirement is both sufficient and necessary for the existence of maximal rewriting in class XP$^{[/,//,[\cdot]]}$ [10]. In class XP$^{[/,//,[\cdot],*]}$, however, it is reduced to a sufficient condition. This is because within this class, in many cases where a maximal rewriting exists, the structural information of the query cannot be preserved in the view. Our concept of trap embedding is defined from a query to the canonical models of a view. It is both necessary and sufficient for the existence of maximal rewriting in class XP$^{[/,//,[\cdot],*]}$. (Refer to Theorem 3.3. and the interpretation following the proof.) The trap embedding method, which is based on a weaker version of the theorem (specified in the corollary that follows the theorem), can generate maximal rewritings in some common cases where the wildcard is present. For the trap relay method, although it is weaker than trap embedding method, it nonetheless is still strictly stronger than useful embedding. This is because a trap relay requires preserving the structural information only for C_segments, not for F_segments. The correctness is still guaranteed due to the ability of an F_segment to perform prefix–suffix matching. Thus, for example, in Fig. 10, no useful embedding exists from λ_1 to μ_1, since node 2 cannot be mapped to node 9 or any of its descendents, but trap relay g_1 is allowed to map nodes 2–9. (As a matter of fact, it can be proven theoretically that in class XP$^{[/,//,[\cdot],*]}$, a useful embedding is necessarily a trap relay, but not vice versa.)

The work in [19] discusses the issue of efficiently maintaining a materialized view in the face of updates. The work in [5,24] study how the queries over the target schema can be answered using the views over the source data, and hence provide a way for schema integration. These works assume the target schema and the source schema are heterogeneous, and therefore a mapping between the two that resolves semantic heterogeneity is crucial for the solution. It is well known, however, that resolving semantic heterogeneity is a difficult problem, cannot be fully automatic, and its accuracy is not without question in many cases. In our context, both the query and the view use the same set of XPath labels, and use the standard syntax and semantics specified by the W3 consortium [25]. Thus, semantic heterogeneity does not exist. (However, if the problem does contain semantic heterogeneity as a component, our method is not directly applicable.)

5. Summary

Although the benefits of query rewriting using views have been well appreciated for the relational database, it starts to emerge as a research topic for XML data only in the last few years. Although the problem statements are similar in both contexts, due to the tree-based structures of XPath queries and XML documents compared with table-based structure for relational data, the problem posts very different challenge for XML data. While most of the work approaching this problem is restricted to equivalence rewriting, we study this problem under the assumption that the original query and its rewritten version are not necessarily equivalent. Our study is restricted to a subclass of XPath queries that consists of four kinds of symbols, /, //, [·], *. We first propose a model for the problem, and introduce some related concepts. Then, we look into the issue that, given a query q, and a view v, how a rewriting for q using v is found. The general framework for our study is query containment. The key notion underlying our approach is trap embedding. We provide two alternatives, which have different characteristics in terms of generalities and performances. We feel this is of practically significant since a user can adopt one based on his/her specific requirement. We also provide conditions under which the rewritings generated by our algorithms are maximal.

In a context where equivalence requirement is removed, maximality becomes the most desirable property that any rewriting technique looks for. Unfortunately, as alluded by Theorem 3.3, there may not exist an efficient algorithm that can generate a maximal rewriting in $XP^{[\,/,\,//,\,[\cdot],\,*]}$ in the general case. Therefore, an interesting question that deserves further study is: what is the weakest restriction on $XP^{[\,/,\,//,\,[\cdot],\,*]}$ under which such an efficient algorithm exists? We have given some conditions under which our methods can generate maximal rewriting using views. However, it is not clear to us at this time whether or not these conditions can be further weakened without compromising the results. Our second method is more efficient compared with the first one, and can generate maximal rewriting for subclasses $XP^{[/,\,//,\,[\cdot]]}$ and $XP^{[/,\,*,\,[\cdot]]}$. However, we do not have the similar result for subclass $XP^{[/,\,//,\,*]}$. At this time, we are not aware of any efficient algorithm that can generate a maximal rewriting for this subclass without further restrictions. Whether or not such an algorithm exists deserves further study.

Appendix A. Proof of Lemma 3.2

We first prove another lemma.

Lemma. *Let q be a pattern and $t[b\sim\#]$ be a tree with trapping node. Then e: nodes(q)→nodes($t[b\sim\#]$) is a trap embedding if there is a pattern $q_0 \subseteq q$ with root(q_0)=root(q) such that (1) e is an embedding from q_0 to t when it is restricted on nodes(q_0), (2) for all n ∈ nodes(q)–nodes(q_0), e(n)=#, (3) for all n_1∈nodes(q_0), n_2∈nodes(q)–nodes(q_0), if ⟨n_1, n_2⟩ is a C_edge, then e(n_1)=b, and if ⟨n_1, n_2⟩ is a D_edge, then e(n_1)$\prec$=b.*

Proof. Note that since q_0 is a tree structure, $\forall n_1,n_2 \in$ nodes(q): [n_2∈nodes(q_0) & $n_1 \prec n_2 \Rightarrow n_1$∈nodes($q_0$)]. We show e meets the conditions in Definition 3.1. First, e(root(q))=e(root(q_0))=root(t). Condition 1 in the definition, is true. Second, let n ∈ nodes(q). If n ∈ nodes(q_0), then either label(n)=*, or label(n)=label(e(n)), else n ∈ nodes(q)–nodes(q_0), in which case by condition 2 in the lemma, e(n)=#. Condition 2 in the definition is true. Let n_1, n_2 ∈ nodes(q). Assume ⟨n_1, n_2⟩ is a C_edge (D_edge). If e(n_1)=#, then n_1 ∈ nodes(q)–nodes(q_0). By the above note, n_2 ∈ nodes(q)–nodes(q_0). Thus e(n_2)=#. Assume e(n_1)≠#, i.e., n_1 ∈ nodes(q_0). If n_2 ∈ nodes(q_0) also, e(n_1) is a parent (ancestor) of e(n_2) in t. If n_2 ∈ nodes(q)–nodes(q_0), then e(n_2)=#. By condition 3 in the lemma, e(n_1)=b (e(n_1)$\prec$=b). This means e(n_1) is a parent (ancestor) of e(n_2) in $t[b\sim\#]$. This proves Condition 3 in the definition. □

Lemma 3.2. *Let q be a pattern, where r_q is the return node. Let t and s be trees, and b∈nodes(t). Then it is true that*:

1. *If there is a trap embedding f: nodes(q)→ nodes($t[b\sim\#]$) and an embedding g: nodes(d)→nodes(s) where d=induced[f, b], then e: nodes(q)→ nodes($s\oplus_b t$), defined as: e(n)=f(n) if f(n)≠#, and e(n)=g(n) otherwise, is an embedding such that e(r_q)=g(r_d).*
2. *If there is an embedding e: nodes(q)→ nodes($s\oplus_b t$) such that e(r_q) ∈ nodes(s), then f: nodes(q)→ nodes($t[b\sim\#]$) defined as: f(n)=e(n) if e(n) ∈ nodes(t), and f(n)=# otherwise, is a trap embedding. Let d=induced[f, b] be its induced pattern. Define g: nodes(d)→ nodes(s) as: g(root(d))=root(s)(=b), and g(n)=e(n) for n ≠ root(d), then g is an embedding, and g(r_d)=e(r_q).*

Proof. *Part* 1: We need to prove e is an embedding. Let n ∈ nodes(q). Since either e(n)=f(n) or e(n)=g(n), and both f and g meet the node condition, e also meets the node condition. Let n_1, n_2 ∈ nodes(q). If both f(n_1)≠# and f(n_2)≠#, then e(n_1)=f(n_1) and e(n_2)=f(n_2). Since the edge condition is met for n_1 and n_2 under f, it is also met under e. Similar argument applies to the case where both f(n_1)=# and f(n_2)=#. Now assume f(n_1)≠# and f(n_2)=#. Thus, e(n_1)=f(n_1), and e(n_2)=g(n_2). First assume ⟨n_1, n_2⟩ is a C_edge. By the definition of a trap embedding, f(n_1)=b. By the definition of an induced pattern, n_2 is a C_child of root(d). Thus, g(n_2) is a child of g(root(d))=root(s) in s, and hence a child of b in $s\oplus_b t$. Now assume ⟨n_1, n_2⟩ is a D_edge. Thus either f(n_1)=b, or f(n_1)$\prec b$, and n_2 is a D_child of root(d). This means g(n_2) is a descendant of g(root(d))=root(s) in s, and hence a descendant of b in $s\oplus_b t$. This in turn implies g(n_2) is a descendant of f(n_1). Since d is non-empty, either f(r_q)=b or f(r_q)=#. In the former case, root(d)=r_d. Thus g(r_d)=root(s)=b=f(r_q)=e(r_q). In the latter case, e(r_q)=g(r_q).

Part 2: First note that $\forall n_1$ ∈ nodes(t), $\forall n_2$ ∈ nodes(s)–b, [n_2 not($\prec$)n_1]. Let T={n|n∈nodes(q) & e(n) ∈ nodes(t)}

and $S=\{n|n \in \text{nodes}(q)\ \&\ e(n) \in \text{nodes}(s)-b\}$. Thus $T \cap S=\Phi$ and $T \cup S=\text{nodes}(q)$. Since e is an embedding, $\forall m_1 \in T$, $\forall m_2 \in S$, $[m_2 \text{ not}(\prec) m_1]$. Let $S_1=\{n|n \in S\ \&\ \text{parent}(n) \in T\}$. We claim $\forall n_1 \in S, \exists n_2 \in S_1, [n_2 \prec n_1]$. To prove, let $x \in S$ and y be the oldest ancestor of x in S. Since $\text{root}(q) \in T$ and $\text{root}(q) \prec = y$, $\text{parent}(y) \in T$. The claim follows. Denote by p_n the sub-pattern rooted at n in q, we have $\cup_{n \in S_1} nodes(p_n)=S$. Also, if we remove all the nodes in S from q (along with the associated edges), we obtain a pattern, q_T, which is a sub-graph of q, and $\text{nodes}(q_T)=T$. We first show f is a trap embedding. Note that when f is restricted on $\text{nodes}(q_T)$, it is an embedding to t, and for all $n \in S, f(n)=\#$. Let $n_1 \in \text{nodes}(q_T)$ and $n_2 \in S$. If $\langle n_1, n_2 \rangle$ is a C_edge, then $e(n_1)$ is a parent of $e(n_2)$. Since $e(n_1) \in \text{nodes}(t)$ and $e(n_2) \in \text{nodes}(s)-b$, $e(n_1)=b$, i.e., $f(n_1)=b$. If $\langle n_1, n_2 \rangle$ is a D_edge, then $e(n_1)$ is an ancestor of $e(n_2)$. Thus $e(n_1) \prec b$. By the above lemma, f is a trap embedding. Now, we show g is an embedding from d to s. First, note that $\forall i, j \in S_1$, $[p_i \cap p_j=\Phi]$. This follows directly from the fact that both p_i and p_j are tree structures, and i not$(\prec)$ j and j not$(\prec)i$. Thus, $\forall j \in S_1$, [j is a child of root(d)], implying nodes(d)–root(d)=S. Since $e(S) \subseteq \text{nodes}(s)-b$, $g(S) \subseteq \text{nodes}(s)-b$. Thus g is a mapping from nodes(d) to nodes(s). By definition of an induced pattern, label(root(-d))=label(b). Let $n \in S$. Either label(n)=* or label(n)=label($e(n)$). The latter implies label(n)=label($f(n)$). Let $n_1, n_2 \in \text{nodes}(d)$. If $n_1, n_2 \in S$, then $\langle n_1, n_2 \rangle$ is a C_edge (D_edge) in d, and $e(n_1)$ is a parent (ancestor) of $e(n_2)$ in s, implying $g(n_1)$ is a parent (ancestor) of $g(n_2)$ in s. Assume $n_1=\text{root}(d)$ and $n_2 \in \text{nodes}(d)-\text{root}(d)$. If $\langle n_1, n_2 \rangle$ is a C_edge in d, then $\langle n_0, n_2 \rangle$ is a C_edge in q for some $n_0 \in \text{nodes}(q_T)$, implying $e(n_0)$ is a parent of $e(n_2)$ in $s \oplus_b t$. This can happen only when $e(n_0)=b=\text{root}(s)$. Recall $e(n_2) \in \text{nodes}(s)-b$, $g(n_1)=b$ and $g(n_2)=e(n_2)$. Thus $g(n_1)$ is a parent of $g(n_2)$ in s. If $\langle n_1, n_2 \rangle$ is a D_edge, from the expressions recalled above, it is trivial to see $g(n_2)$ is a descendant of $g(n_1)$. Thus, g is an embedding. Finally, since $e(r_q) \in \text{nodes}(s)$, either $e(r_q)=b$ (i.e., root(s)) or $e(r_q) \in \text{nodes}(s)-b$. In the former case, $f(r_q)=b$, and therefore $\text{root}(d)=r_d$, implying $g(r_d)=b$. In the latter case, $f(r_q)=\#$, implying $r_q=r_d$. We have $e(r_q)=g(r_q)=g(r_d)$. □

Appendix B. Proof of Lemmas 3.5 and 3.6

We first show an additional lemma.

Lemma A. *Let $\varepsilon(\lambda)=\mu$ be a trap relay, and $e(\mu)=t$ be an embedding. If $\varepsilon(\lambda)$ is in one of the three base cases in Definition 3.3, then there is a trap embedding $f(\lambda)=t$.*

Proof. If $\varepsilon(\lambda)$ is a trap mapping, then $\mu=\#$. Since $e(\mu)=t$, we also have $t=\#$. Define $f(n)=\#$ for all $n \in \text{nodes}(\lambda)$. So $f(.)$ is a trap embedding. If $\varepsilon(\lambda)$ is a C_segment mapping, define $f(n)=e(\varepsilon(n))$. Since both $e(.)$ and $\varepsilon(.)$ meet node and edge conditions, clearly $f(.)$ also does. Thus it is a trap embedding. Now assume $\varepsilon(\lambda)$ is an F_segment mapping. Since both $\varepsilon(.)$ and $e(.)$ are 1–1 mapping, we have $|\lambda| \le |\mu| \le |t|$. By Theorem 3.4, the claim follows. □

Lemma 3.5. *Let λ and μ be two path patterns. Assume $\varepsilon(\lambda)=\mu[b\sim\#]$, and $\varepsilon(\lambda)$ is a trap relay. Let t be a tree path, and $e(\mu[b\sim\#])=t[c\sim\#]$ be an embedding. Then (1) there is a trap embedding $f(\lambda)=t[c\sim\#]$, and (2) let d_1=induced[ε, b] and d_2=induced[f, c]. Then ($d_1=d_2$) except (label(root(d_1))$\neq$*$\Rightarrow$ label(root(d_1))=label(root(d_1))).*

Proof. From Definition 3.4, $\lambda=a_1\lambda_1a_2,\ldots,a_n\lambda_n$, and $\mu[b\sim\#]=b_1\mu_1b_2\mu_2,\ldots,b_{n-1}\mu_{n-1}b_n$ where $n \ge 1$ and $b_n=\#$, such that for all $1 \le i \le n-1$, $\varepsilon(a_i\lambda_ia_{i+1})=b_i\mu_ib_{i+1}$ is either a C_segment mapping, or an F_segment mapping, and $\varepsilon(a_n\lambda_n)=b_n$ is a trap mapping. Also, from the definition of an embedding, $t[c\sim\#]=c_1t_1c_2,\ldots,c_{n-1}t_{n-1}c_n$ where $c_n=\#$, such that for all $1 \le i \le n-1$, $e(b_i\mu_ib_{i+1})=c_it_ic_{i+1}$. By Lemma A, for all $1 \le i \le n-1$, there is a trap embedding $f_i(a_i\lambda_ia_{i+1})=c_it_ic_{i+1}$, and a trap embedding $f_n(a_n\lambda_n)=c_n$. Define f: nodes(λ)$\rightarrow$ nodes($t[c\sim\#]$) as: for all $1 \le i \le n-1$, $f(n)=f_i(n)$ if $n \in \text{nodes}(a_i\lambda_ia_{i+1})$. Clearly, f is a trap embedding. In addition, induced[f, c]=$xa_n\lambda_n$ where $\langle x, a_n \rangle$ is a C_edge (D_edge) iff a_n is a C_child (D_child) in λ. We also have induced[ε, b]=$ya_n\lambda_n$ where $\langle y, a_n \rangle$ is a C_edge (D_edge) iff a_n is a C_child (D_child) in λ. Note that the only difference between $ya_n\lambda_n$ and $xa_n\lambda_n$ is in their roots, y and x. We have label(y)=label(b) and label(x)=label(c). Since $e(b)=c$, label(b)$\neq$*$\Rightarrow$ label(b)=label(c). This implies label(y)$\neq$*$\Rightarrow$ label(y)=label(x). □

Lemma 3.6. *Let q and v be patterns, where $v \neq \Phi$. Let r_1 be the return node in v. Assume g: nodes(q)$\rightarrow$ nodes($v[r_1\sim\#]$) is a trap relay. Let t be a tree and e: nodes($v[r_1\sim\#]$)$\rightarrow$ nodes($t[r_2\sim\#]$) be an embedding. Then there is a trap embedding f: nodes(q)$\rightarrow$ nodes($t[r_2\sim\#]$) such that (1) for each block λ in q, $f(\lambda)=e(g(\lambda))$, and (2) let induced[g, r_1]=d_1, and induced[f, r_2]=d_2, then ($d_1=d_2$) except (label(root(d_1))$\neq$*$\Rightarrow$ label(root(d_1))=label(root(d_2))).*

Proof. Let λ be any block in q, μ be a path pattern in $v[r_1\sim\#]$ such that $g(\lambda)=\mu$, and σ be a path in t such that $e(\mu)=\sigma$. By Lemma 3.5, there is a trap embedding $f_\lambda(\lambda)=\sigma$ that satisfies the two conditions in Lemma 3.6. Define f: nodes(q)$\rightarrow$ nodes($t[r_1\sim\#]$) as: for all $n \in \text{nodes}(q)$, $f(n)=f_\lambda(n)$ if $n \in \text{nodes}(\lambda)$. We first prove f is a well defined mapping. If n is not a fork point, then $f(n)$ is uniquely determined. Assume n is a fork point. Without loss of generality, let n=start(α) and n=start(β) where α and β are different blocks. The arguments for the other cases are similar. Note $g(n)$=start($g(\alpha)$)=start($g(\beta)$), and $e(g(n))$=start($e(g(\alpha)$)=start($e(g(\beta)$). Since $f_\alpha(\alpha)=e(g(\alpha))$ and $f_\beta(\beta)=e(g(\beta))$, $f_\alpha(n)$=start($e(g(\alpha)$) and $f_\beta(n)$= start($e(g(\beta)$), implying $f_\alpha(n)=f_\beta(n)$. Thus f is well defined. We now prove f is a trap embedding. First, let λ be such that start(λ)=root(q). Thus g(start(λ))=root($v[r_1\sim\#]$), implying (start(μ)=root($v[r_1\sim\#]$). Thus start(σ)= root($t[r_2\sim\#]$). Combine the above equalities, we have f(root(q))=root($t[r_2\sim\#]$). Since any node and any edge in q must belong to some block λ, the node and edge conditions follow directly from the fact that $f=f_\lambda$ and f_λ is a trap embedding. Thus, f is a trap embedding. We now show f meets the two conditions in the Lemma. From the way f is defined, condition 1 is clearly true. For condition 2, let d_1=induced[g, r_1]. Let root(q)=a_0, root($v[r_1\sim\#]$)=b_0, the trapping node in $v[r_1\sim\#]$ be b_1, root($t[r_2\sim\#]$)=c_0, and the trapping node in $t[r_2{\sim}\#]$ be c_1. Let $\langle \text{root}(d_1), a_1,\ldots,a_2 \rangle$ be a path in d_1 where a_1 is a child of root(d_1) and a_2 is a leaf in d_1, and $\theta = \langle a_0,\ldots,a_1,\ldots,a_2 \rangle$, which is a path in q. It must be the case that induced[g_θ, r_1]=$\langle o_1, a_1, \ldots, a_2 \rangle$ where label(o_1)=label(root(d_1)). Note that a_1 must be the first note in θ such that $g(a_1)=b_1$. Since $g(a_0)\neq b_1$, $a_0\neq a_1$. Let $\theta=\tau_1\blacklozenge\ldots\blacklozenge\tau_k\blacklozenge\ldots\ \blacklozenge\tau_m$ where each τ_i is a block, and start(τ_k)$\prec a_1 \prec$=end(τ_k), i.e., τ_k is the first block in θ that contains a_1, and hence $\forall i$: [$k<i\leq m \Rightarrow g(\tau_i)=b_1$=#]. Thus induced[$g_\theta$, r_1]=induced[g_{τ_k}, r_1] $\blacklozenge\tau_{k+1},\ldots,\blacklozenge\tau_m$. On the other hand, since # is b_1 in $v[r_1{\sim}\#]$, we have $g(\tau_k)=\mu_1 r_1 b_1$ for some path μ_1 in $v[r_1\tilde{}\#]$. Since $\langle r_1, b_1\rangle$ is a C_edge, and b_1 is the only node labeled # in $v[r_1{\sim}\#]$, and $\langle r_2, c_1\rangle$ is a C_edge, and c_1 is the only node labeled # in $t[r_2{\sim}\#]$, we have $e(\mu_1 r_1 b_1)=\sigma_1 r_2 c_1$ for some σ_1 in $t[r_2{\sim}\#]$. By Lemma 3.5, $t(\tau k)=f_{\tau_k}(\tau_k)=\sigma_1 r_2 c_1$ and, letting d_3=induced[g_{τ_k}, r_1] and d_4=induced[f_{τ_k}, r_2], we have (d_3=d_4) except (label(root(d_3))$\neq$* $\Rightarrow$ label(root(d_3))=label(root(d_4))). This implies induced[f_{τ_k}, r_2] $\blacklozenge\tau_{k+1},\ldots,\blacklozenge\tau_m=\langle o_2, a_1,\ldots,a_2\rangle$, where label($o_1$)$\neq$*$\Rightarrow$ label(o_1)=label(o_2). Since $e(b_1)=c_1$, by Lemma 3.5, we have $\forall i$: [$k<i\leq m\Rightarrow$ $f_{\tau_i}(\tau_i)=c_1$=#]. This means induced[f_θ, r_2]=induced[f_{τ_k}, r_2] $\blacklozenge\tau_{k+1},\ldots,\blacklozenge\tau_m$. Thus, induced[$f_\theta$, r_2]=$\langle o_2, a_1, \ldots, a_2\rangle$. Let d_2=induced[f, r_2]. We must have $\langle \text{root}(d_2), a_1,\ldots,a_2\rangle$ is a path in d_2, and label(o_2)=label(root(d_2)). Noting label(o_1)=label(root(d_1)), we have label(root(d_1))$\neq$* $\Rightarrow$ label(root(d_1))=label(root(d_2)). Similar argument can also show that for any path $\langle$root(d_2), a_1, …, $a_2\rangle$ in d_2, there is a path $\langle$root(d_1), a_1, …, $a_2\rangle$ in d_1 such that label(root(d_1))$\neq$*$\Rightarrow$label(root(d_1))=label(root(d_2)). Thus condition 2 follows. □

Appendix C. Proof of Theorem 3.10

Theorem 3.10. *The rewriting generated by the corollary of Theorem 3.3 is maximal if the following conditions hold true for pattern q: (1) any node labeled * is not incident with a D_edge, and (2) if a leaf is labeled *, then its parent is not.*

Proof. Consider the (L+1)-extension t_{L+1} of v for q, where L is the number of nodes in the longest star-path in q. Let r_q and r_v be the return nodes in q and v, respectively. Let e: nodes(q) → nodes($t_{L+1}[\pi_{t_{L+1}}(r_v){\sim}\#]$) be a trap embedding. Let $n \in$ nodes(q). Assume n is an internal node. If label(n)$\neq$*, then label(n)=label($e(n)$) $\neq$ *, which means $e(n)$ is not a rubber node. (Recall a rubber node is labeled z that no label other than * in q can be matched.) If label(n)=*, then let λ be the longest path containing n in which all the nodes are marked *. By condition 1, λ is a star-path. By condition 2, end(λ) is not a leaf node. This means there is a path aλb in q where label(a)$\neq$* and label(b)$\neq$*. Since $|\lambda|\leq L$ and every rubber path in t_{L+1} has a degree of L+1, we have $\forall n\in$nodes(λ): [$e(n)$ is not a rubber node in t_{L+1}]. Now assume n is a leaf node in q. If $e(n)$ is a rubber node in t_{L+1}, then label(n)=*. By condition 1, $\langle$parent(n), $n\rangle$ is a C_edge, and by condition 2 label(parent(n))$\neq$*. This means $\langle e$(parent(n)), $e(n)\rangle$ is an edge and e(parent(n)) is not a rubber node in t_{L+1}. In summary, if n is an internal node in q, then $e(n)$ is not a rubber node in t_{L+1}, and if n is a leaf node such that $e(n)$ is a rubber node, then $\langle e$(parent(n)), $e(n)\rangle$ is an edge and e(parent(n)) is not a rubber node in t_{L+1}. Let t be any canonical model of v for q, we define a mapping f: nodes(q)→ nodes($t[\pi_t(r_v){\sim}\#]$) as follows. If n is an internal node, or a leaf node in q but $e(n)$ is not a rubber node in t_{L+1}, then let $f(n)=(\pi_t\circ\pi^{-1}_{t_{L+1}}\circ e)(n)$ if $e(n)\neq\#$, and $f(n)$=# otherwise. If n is a leaf node and $e(n)$ is a rubber node, then let $f(n)$=child($\pi_t(a)$) in $\pi_t(\langle a,b\rangle)$ where $\langle a, b\rangle\in$D_edges(v) and $e(n)\in\pi_{t_{L+1}}(\langle a,b\rangle)$. We now prove that f is a trap embedding. First, root(q) is an internal node and e(root(q))=root(t_{L+1})$\neq$#. We have $f(root(q))=(\pi_t\circ\pi^{-1}_{t_{L+1}}\circ e)(root(q))=(\pi_t\circ\pi_{t_{L+}}\ 1^{-1})(root(t_L+1))$ $=\pi_t(root(v))=root(t)$. Second, let $n\in$ nodes(q)–root(q). Assume n is an internal node, or a leaf node but $e(n)$ is not a rubber node in t_{L+1}. If $e(n)\neq\#$, then label(n)$\neq$*$\Rightarrow$ label(n)=label($e(n)$), which implies label($e(n)$) =label($\pi^{-1}_{t_{L+1}}(e(n))$)$\neq$*, which in turn implies label ($\pi_t(\pi^{-1}_{t_{L+1}}(e(n)))$)=label($\pi^{-1}_{t_{L+1}}(e(n))$). Combine the above, we have label(n)=label($f(n)$). If $e(n)$=#, we have label ($f(n)$)=#. Now assume n is a leaf and $e(n)$ is a rubber node. We must have label(n)=*. We have proven the root and node conditions. For the edge condition, let $\langle n_1, n_2\rangle$ be a C_edge (D_edge) in q. Note n_1 can only be an internal node in q. We consider the following cases.

Case 1: n_2 is an internal node in q, or a leaf node but $e(n_2)$ is not a rubber node in t_{L+1}.

Subcase 1: $e(n_1)\neq\#$ and $e(n_2)\neq\#$. Thus $\langle e(n_1), e(n_2)\rangle$ is an edge (path) in t_{L+1}, implying $\langle\pi^{-1}_{t_{L+1}}(e(n_1)), \pi^{-1}_{t_{L+1}}(e(n_2))\rangle$ is a C_edge (path) in v. This means $\langle\pi_t(\pi^{-1}_{t_{L+1}}(e(n_1)))$, $\pi_t(\pi^{-1}_{t_{L+1}}(e(n_2)))\rangle$, or $\langle f(n_1), f(n_2)\rangle$, is an edge (path) in t.

Subcase 2: $e(n_1)\neq\#$ and $e(n_2)$=#. Again, $\langle e(n_1), e(n_2)\rangle$ is an edge (path) in t_{L+1}. Thus $e(n_1)$=($\prec$=)$\pi_{t_{L+1}}(r_v)$. This means $\pi^{-1}_{t_{L+1}}(e(n_1))$=($\prec$=)$r_v$, implying $\pi_t(\pi^{-1}_{t_{L+1}}(e(n_1)))$= ($\prec$=)$\pi_t(r_v)$. By definition of f, $f(n_1)$=($\prec$=)$\pi_t(r_v)$ and $f(n_2)$=#. Noting # is a child of $\pi_t(r_v)$ in t, $\langle f(n_1), f(n_2)\rangle$ is an edge (path) in t.

Subcase 3: $e(n_1)$=# and $e(n_2)$=#. By definition of f, $f(n_1)$=$f(n_2)$=#.

Subcase 4: $e(n_1)$=# and $e(n_2)\neq\#$. Since e is a trap embedding, this case is impossible.

Case 2: n_2 is a leaf node in q, and $e(n_2)$ is a rubber node in t_{L+1}. We must have label(n_2)=*. Thus $\langle n_1, n_2\rangle$ is a C_edge and label(n_1)$\neq$* in q. Let $\langle a, b\rangle\in$D_edge(v) such that $e(n_2)\in\pi_{t_{L+1}}(\langle a,b\rangle)$. We have shown that $e(n_2)$ is a child of $e(n_1)$. Thus it must be the case that $e(n_1)=\pi_{t_{L+1}}(a)$, otherwise $e(n_1)$ would be a rubber node, which is impossible. Note $e(n_1)\neq\#$, otherwise, $e(n_2)$ cannot possibly be a rubber node. By definition of f, $f(n_1)$= $\pi_t(\pi^{-1}_{t_{L+1}}(e(n_1)))=\pi_t(\pi^{-1}_{t_{L+1}}(\pi_{t_{L+1}}(a)))=\pi_t(a)$, and $f(n_2)$= child($\pi_t(a)$). Thus, $\langle f(n_1), f(n_2)\rangle$ is an edge in t. We have proven the edge condition. Note that by our definition, for any $n\in$ nodes(q), $f(n)$=# if f $e(n)$=#. Thus, *induced[e, $\pi_{t_{L+1}}(r_v)$]=induced[f, $\pi_t(r_v)$].* Since t is an arbitrary canonical model of v, and e is an arbitrary trap embedding from q to $t_{L+1}[\pi_{t_{L+1}}(r_v){\sim}\#]$, $P_{t_{L+1}}\sqsubseteq P_t$ for all t. By the corollary of Theorem 3.3, $R=\{d\oplus v : d\sqsubseteq P_{t_{L+1}}\}$ is a rewriting for q using v. Let $x\oplus v$ be any rewriting for q using v. By

Theorem 3.3, $\{x\} \subseteq_{\clubsuit} P_{t_{L+1}}$, implying $\{x \oplus v\} \subseteq_{\clubsuit} R$. Thus R is maximal .□

References

[1] S. Amer-Yahia, S. Cho, L.V.S. Lakshmanan, D. Srivastava, Minimization of tree pattern queries, in: SIGMOD, 2001, pp. 497–508.
[2] A. Balmin, F. Ozcan, K. Beyer, R. Cochrane, A framework for using materialized XPath views in XML query processing, in: Thirtieth VLDB Conference, 2004, pp. 60–71.
[3] D. Calvanese, G. Giacomo, M. Lenzerini, M. Vardi, Answering regular path queries using views, in: Sixteenth International Conference on Data Engineering, 2000, pp. 389–398.
[4] L. Chen, E. Rundensteiner, ACE-XQ: a cache-aware XQuery answering system, in: WebDB, pp. 31–36.
[5] V. Cristophides, S. Cluet, J. Simeon, On wrapping query languages and efficient xml integration, in: SIGMOD Confernce, 2000.
[6] A. Deutsch, V. Tannen, Reformultion of XML queries and constraints, in: Ninth International Conference on PODS, 2003.
[7] G. Grahne, A. Thomo, Query containment and rewriting using views for regular path queries under constraints, in: Ninth International Conference on PODS, 2003, pp. 111–121.
[8] A.Y. Halevy, Answering queries using views: a survey, VLDB Journal 10 (2001) 270–294.
[9] T. Kirk, A. Levy, Y. Sagic, D. Srivastava, The information manifold, in: AAAI Spring Symposium on Information Gathering, 1995.
[10] L.V.S. Lakshmanan, H. Wang, Z. Zhao, Answering tree pattern queries using views, in: Proceedings of the 32nd VLDB Conference, 2006.
[11] A. Levy, A. Mendelzon, Y. Sagiv, D. Srivastava, Answering queries using views, in: The 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principle of Database Systems, 1995.
[12] G. Miklau, D. Suciu, Containment and equivalence for an XPath fragment, in: ACM PODS, 2002, pp. 65–76.
[13] T. Milo, D. Suciu, Index structures for path expressions, in: ICDT, 1999, pp. 277–295.
[14] P. Mitra, An algorithm for answering queries efficiently using views, in: Australian Database Conference, 2001.
[15] F. Neven, T. Schwentick, XPath containment in the presence of disjunction, DTDs and variables, in: ICDT, 2003.
[16] Y. Papakonstantinou, V. Vassalos, Query rewriting using semistructured views, in: SIGMOD Conference, 1999.
[17] R. Pottinger, A. Levy, A scalable algorithm for answering queries using views, VLDB Journal 10 (2–3) (2001) 182–198.
[18] X. Qian, Query folding, in: Twelfth International Conference on Data Engineering, 1996.
[19] A. Sawires, J. Tatemura, O. Po, D. Agrawal, A.E. Abbadi, K.S. Candan, Maintaining XPath views in loosely coupled systems, in: Proceedings of the 32nd VLDB Conference, 2006.
[20] J. Shanmungasundaram, J. Kiernan, E. Shekita, C. Fan, J. Funderburk, Querying XML views of relational data, in: VLDB Conference, 2001.
[21] J. Tang, S. Zhou, A theoretic framework for answering XPath queries using views, in: XSym, 2005, pp. 18–33.
[22] Wanhong Xu, Z. Ozsoyoglu, Rewriting XPath queries using materialized views, in: Thirty-first VLDB Conference, 2005, pp. 121–480.
[23] P.T. Wood, Containment for XPath fragments under DTD constraints, in: ICDT, 2003, pp. 300–314.
[24] C. Yu, L. Popa, Constraint-based XML query rewriting for data integration, in: SIGMOD Conference, 2004.
[25] XQuery: a query language for XML, ⟨http://www.w3.org/TR/xquery⟩, 2003.