NAIST-IS-MT1051135

Master's Thesis

Verification of the Security against Inference Attacks on XML Databases

Chittaphone Phonharath

March 12, 2012

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

A Master's Thesis
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
MASTER of ENGINEERING

Chittaphone Phonharath

Thesis Committee:

Professor Hiroyuki Seki	(Supervisor)
Professor Minoru Ito	(Co-supervisor)
Associate Professor Yuichi Kaji	(Co-supervisor)
Assistant Professor Kenji Hashimoto	(Co-supervisor)

Verification of the Security against Inference Attacks on XML Databases*

Chittaphone Phonharath

Abstract

We examine the decidability of a static analysis problem on k-secrecy, which is a metric for the security against inference attacks on XML databases. Intuitively, k-secrecy means that the number of candidates of sensitive data of a given database instance or the result of unauthorized query cannot be narrowed down to $k-1$ by using available information such as authorized queries and their results. In this study, we investigate the decidability of the schema k-secrecy problem defined as follows: For a given XML database schema, an authorized query and unauthorized query, decide whether every database instance conforming to the given schema is k-secret.

We first show that the schema k-secrecy problem is undecidable for any finite $k > 1$ even when queries are represented by a simple subclass of linear deterministic top-down tree transducers. We next show that the schema ∞-secrecy problem is decidable for queries represented by linear deterministic top-down tree transducers. We give an algorithm for deciding the schema ∞-secrecy problem and analyze its time complexity. Moreover, we adjust/extend the decision algorithm to the case that queries are given by linear deterministic bottom-up tree transducers and linear deterministic top-down tree transducers with regular look-ahead.

Keywords:

Inference attack, security, static analysis, XML

* Master's Thesis, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT1051135, March 12, 2012.

Contents

List of Figures

1. Introduction

XML (Extensible Markup Language) has become a de facto standard of the description language for structured/semi-structured data which are exchanged by and stored in a modern computer system. An XML database is a database consisting of XML data or documents of which structure is defined by an appropriate schema language such as DTD (Document Type Definition) and W3C XML schema. Many researchers define the concept of XML database in different ways depending on their research interest and application domains. Some researchers define XML database just as a storage of XML format, while other define XML database as an XML document which various information can be stored in and retrieved from. For example, in an e-commerce system, information between user-user, provider-user, or provider-provider can be exchanged in a flexible way by using XML format. In a web hosting system, various services including website service, domain name service and server service, use XML documents to store the information to be published on the web. As the number of website hosts increases, the number and/or size of XML documents also increases. Moreover, some of the documents need to be updated frequently.

Why XML databases rather than relational databases? The answer arises from the fact that the flexibility of XML format makes it possible to satisfy the requirements such as frequently updating, analysis, and retrieval of information based on complicated conditions. XML data model is flexible and expressive compared to the relational data model of which structure is flat and fixed. XML is expressive in a sense that while the language for describing a schema or a document is simple and an XML data has a text-based format, you easily construct and process millions of XML documents as a text file. XML is flexible because it allows users to define their own tags to markup their data. The whole set of data is often called a *document or a database instance* (see Figure 1). Precisely, the set of tag names and their relative positions in a document are defined by *schema*. Schema is a description of constrains on the structure of documents by using *schema languages*, e.g., Document Type Definition (DTD). Formally, an XML database structure is modeled as an unranked labeled ordered tree (see Figure 2).

```
<Library>
<Book1>
   <BookID> 576280 </BookID>
   <Title> Personality </Title>
   <Author>
      <Gender> Male </Gender>
      <Name> Jame </Name>
   </Author>
</Book1>
<Book2>
   <BookID> 576282 </BookID>
   <Title> Job Interview </Title>
   <Author>
      <Gender> Female </Gender>
      <Name> Roony </Name>
   </Author>
</Book2>
</Library>
```

Figure 1. An XML instance

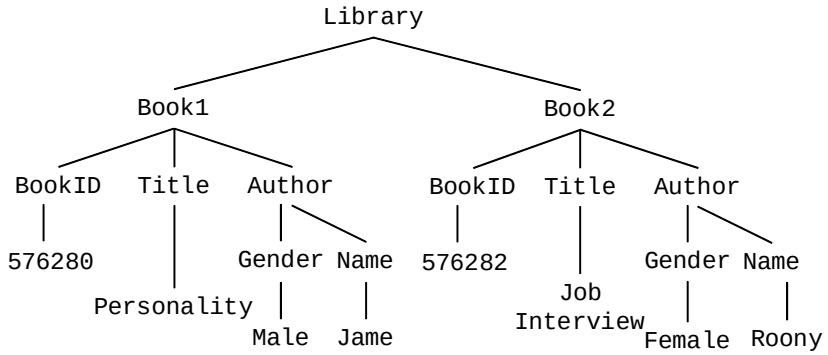


Figure 2. An XML database tree structure

Access control is one of the most important mechanisms of database management system to keep confidential information secret, which should be protected from malicious access. A typical way of realizing access control is to divide the available set of queries into authorized queries and unauthorized ones and to allow a user to obtain only the result of the authorized queries for a database instance.

At first glance, this access control policy works well. However, it may happen that a user can obtain the result (or a set of candidates of the correct result) of

an unauthorized query by cleverly combining the results of authorized queries. Such an attack is called an *inference attack*. Hence, it is desirable to be able to decide whether inference attack is possible or not for a given database. To quantify the degree of safeness against inference attacks, Hashimoto, et al. [4] defines k-secrecy. Intuitively, a database instance is *k-secret* if an attacker cannot narrow down the number of candidates of the result of an unauthorized query to less than k. In particular, an instance is ∞-secret if the number of candidates cannot be narrowed down to a finite value. The underlying idea is similar to ℓ-diversity in [7]. For XML databases, the problem of deciding k-secrecy against inference attacks is defined as follows: For a given XML database schema A_G, a database instance t conforming to the schema, authorized queries $T_{a_1}, \ldots, T_{a_n}$, and an unauthorized query T_s, decide whether t is k-secret or not. A_G is given by a tree automaton, each of the queries $T_{a_1}, \ldots, T_{a_n}$ and T_s is represented by a composition of relabeling and/or deleting tree transducers. In [4], it is shown that the problem is decidable and its time complexity is polynomial in the size of a given instance t.

The problem discussed in [4] is in instance-level in the sense that we are required to decide whether a *given instance* t is k-secret or not. In this paper, we will discuss the decidability of the problem for schema-level k-secrecy, which is the problem o describing whether *every instance* conforming to a given schema is k-secret. If we can guarantee that every instance is k-secret, then we do not need to repeatedly verify that a new instance is k-secret or not every time an instance is updated.

In this paper, we assume that the number of authorized queries is one for simplicity. Generalizing the results of this paper to the case that there are multiple authorized queries is an interesting problem. Also, we do not consider database updates in this paper. It is left as future study as discussed in the last section.

Based on these assumptions, this paper defines the problem of schema-level k secrecy, as follows. For an XML schema A_G, let $L(A_G)$ denote the set of instances conforming to A_G. For a query T and an instance t, let $T(t)$ denote the result of T applied to t. Also, for a set of instances L, let $T(L) = \{T(t) \mid t \in L\}$. We fix the value of k to be a natural number greater than 1 or $k = \infty$. For a given XML schema A_G, an authorized query T_a and an unauthorized query T_s, decide

whether $|T_s(T_a^{-1}(T_a(t)) \cap L(A_G))| \geq k$ for every instance $t \in L(A_G)$, when $k > 1$. When $k = \infty$, decide whether $T_s(T_a^{-1}(T_a(t)) \cap L(A_G))$ is an infinite set for every instance $t \in L(A_G)$. Note that $T_s(T_a^{-1}(T_a(t)) \cap L(A_G))$ is the set of candidates of the result of the unauthorized query T_s. As in [4], we assume that executable codes for both of authorized and unauthorized queries are open to a user while a database instance itself is kept secret.

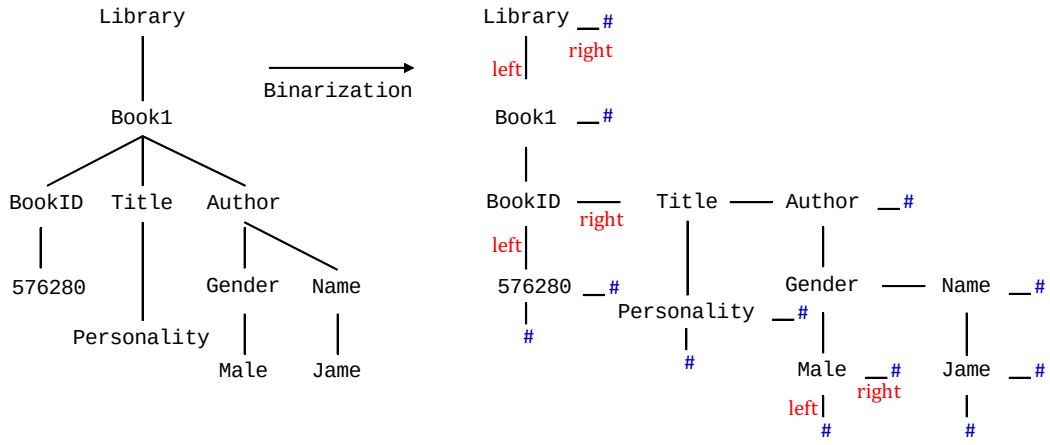
The main results of the paper are as follows. Assume that an XML schema is given by a tree automaton and a query is given by a linear deterministic top-down tree transducer, a linear deterministic bottom-up tree transducer or a linear deterministic top-down tree transducer with regular look-ahead. Then the problem is undecidable when k is a finite value and it is decidable when $k = \infty$.

The rest of this paper is organized as follows. Section 2 provides definitions and notations including tree automata and top-down tree transducers, bottom-up tree transducers, and top-down tree transducers with regular look-ahead. In section 3, we formally define the problem of deciding schema-level k-secrecy. Section 4 shows that the problem is undecidable for every finite $k > 1$. In section 5, it is shown that the problem becomes decidable when $k = \infty$. Section 6 concludes the paper, mentioning possible future research including k-secrecy preservation through updates.

2. Models

2.1 XML documents

XML documents or XML database instances are often modeled by *unranked labeled ordered trees*, where each node has an arbitrary number of child nodes. Figure 1 shows an example of an XML document and Figure 2 is its tree representation. In this paper, for simplicity, we consider only *binary* labeled ordered trees. Actually, some encodings from unranked trees to binary trees, without loss of expressive power, are proposed [2, 5]. Among them, we adopt *First-Child-Next-Sibling* encoding [5], which is a simple and widely accepted encoding. The first child of a node in an unranked tree corresponds to the left child of the node in the encoded binary tree, and the next sibling of a node in an unranked tree corresponds to the right child of the node in the encoded binary tree. Figure 3 shows an example of this encoding. We use # as the special symbol which means that there is no child or no next sibling.



(*a*) XML tree (unranked) (*b*) Binary tree

Figure 3. Binarization of a tree as an XML database instance. *(a) shows an XML tree structure model as an unranked labeled ordered tree. We do the binarization of the XML tree to obtain the binary tree shown in (b) by using First-Child-Next-Sibling encoding technique.*

The set $\mathcal{T}_\Sigma$ of (binary) labeled ordered trees over an alphabet Σ is the smallest set defined by:

- $\# \in \mathcal{T}_\Sigma$, and
- $a(t_1, t_2) \in \mathcal{T}_\Sigma$ if $t_1, t_2 \in \mathcal{T}_\Sigma$ and $a \in \Sigma$.

For $t \in \mathcal{T}_\Sigma$, let $V(t)$ denote the set of nodes of t, and let $\lambda(v)$ denote the label of v. For trees t, t' and a node $v \in V(t)$, let t/v denote the subtree of t rooted at v and let $t[v \leftarrow t']$ denote the tree obtained from t by replacing t/v by t'. Let $\mathcal{X}_2$ be the set consisting of two variables x_1 and x_2. A *context* C over Σ is a tree over $\Sigma \cup \mathcal{X}_2$ such that x_1 and x_2 can only appear on leaves of the tree at most once, respectively. Let $\mathcal{C}(\Sigma, \mathcal{X}_2)$ be the set of all contexts over Σ. For a context $C \in \mathcal{C}(\Sigma, \mathcal{X}_2)$ and $t_1, t_2 \in \mathcal{T}_\Sigma$, let $C[t_1, t_2]$ denote the tree in $\mathcal{T}_\Sigma$ obtained from C by replacing each variable x_i by t_i for $i \in \{1, 2\}$.

2.2 Sets of XML documents

We use *tree automata* [2, 5, 8] to represent sets of trees or schemas for XML databases. Tree automata are a finite state machine model for accepting a tree. They are used as a formal model of the schema description languages, RelaxNG and DTD.

Definition 1 *A tree automaton A is a 4-tuple (Q, Σ, Q_F, δ), where*

- *Q is a finite set of states,*
- *Σ is an alphabet,*
- *$Q_F \subseteq Q$ is the set of final states, and*
- *δ is a set of transition rules in the form of either*

$$q \to a(q_1, q_2) \quad or \quad q \to \#$$

where $q, q_1, q_2 \in Q$ and $a \in \Sigma$.

A tree $t \in T_\Sigma$ is *accepted* by a tree automaton A if and only if there is a mapping r from $V(t)$ to Q such that

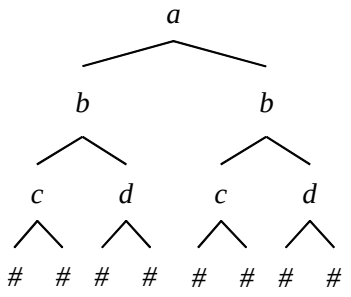
- $r(v_0) \in Q_F$ where v_0 is the root of t.
- $r(v) \to \lambda(v)(r(v_1), r(v_2)) \in \delta$
  if v_1 and v_2 are the left and right children of v, respectively and
- $r(v) \to \# \in \delta$ if v is a leaf.

The mapping r is called a *run* of A on t. We define the language $L(A)$ to be $\{t \in \mathcal{T}_\Sigma \mid A \text{ accepts } t\}$. $L(A)$ is called the language recognized by A. A *regular tree language* is a language recognized by a tree automaton. A is *unambiguous* if for every $t \in L(A)$, a run of A on t is unique. For an unambiguous tree automaton A and each $t \in L(A)$, let $r(A, t)$ denote the run of A on t. Figure 4 shows an example of a tree automaton and a run of the tree automaton on $t = a(b(c(\#), d(\#)), b(c(\#), d(\#)))$.

For a tree automaton $A = (Q, \Sigma, Q_F, \delta)$, we say that a state $q \in Q$ is *recursive* through a proper path of nodes $v_1, v_2, \ldots, v_n$ $(n \geq 2)$ of a tree $t \in L(A)$ if $r(A, v_1) = r(A, v_n) = q$.

2.3 Queries

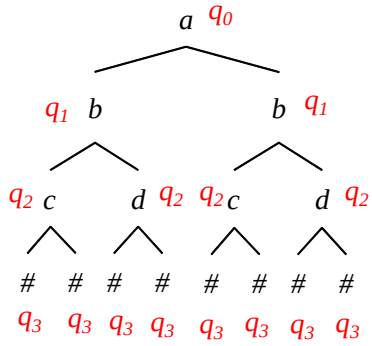
As languages for XML queries/transformations, XQuery and XSLT are developed and recommended by W3C (World Wide Web Consortium). Both queries and programs written in XQuery and XSLT can be regarded as transformations from trees to trees. Unfortunately, we naively conjecture that many static analysis problems are undecidable for the full class because XQuery and XSLT are Turing complete [6]. The core of tree transduction of XSLT can be modeled by *tree transducers* [1, 2, 5]. Tree transducers are a finite machine model for tree transformations. Here, as query models, we introduce three types of tree transducers, linear top-down tree transducers, linear bottom-up tree transducers, and linear top-down tree transducers with regular look-ahead and their deterministic subclasses.



(*a*) Tree ***t***

$A = (Q, \Sigma, Q_F, \delta)$
$Q = \{q_0, q_1, q_2, q_3\}$
$\Sigma = \{a, b, c, d\}$
$Q_F = \{q_0\}$
δ = 1. $q_0 \rightarrow a\ (q_1,\ q_1)$
2. $q_1 \rightarrow b\ (q_2,\ q_2)$
3. $q_2 \rightarrow c\ (q_3,\ q_3)$
4. $q_2 \rightarrow d\ (q_3,\ q_3)$
5. $q_3 \rightarrow \#$

(*b*) Tree automaton ***A***



(*c*) A run of ***A*** on ***t***

Figure 4. A tree automaton and a run. *This figure shows that tree t is accepted by automaton A since there is a successful run r of automaton A on t such that the state assigned to the root is a final state. We denote by $L(A)$ the set of trees accepted by a tree automaton A over Σ.*

2.3.1 Linear top-down tree transducers

A top-down tree transducer [2] processes each node of an input tree in top-down manner, according to the path from the root to the node.

Definition 2 *A linear top-down tree transducer T is a 5-tuple $(Q, \Sigma, \Delta, q_0, \delta)$, where*

- *Q is a finite set of states,*
- *Σ is an input alphabet,*
- *Δ is an output alphabet,*
- *$q_0 \in Q$ is the initial state, and*
- *δ is a set of transduction rules of the following two types:*

$$q(a(x_1, x_2)) \to C[q_1(x_1), q_2(x_2)] \quad or \quad q(\#) \to t'.$$

where $a \in \Sigma$, $q, q_1, q_2 \in Q$, $C \in \mathcal{C}(\Delta, \mathcal{X}_2)$, and $t' \in \mathcal{T}_\Delta$.

The move relation $\to$ by a linear top-down tree transducer $T = (Q, \Sigma, \Delta, Q_0, \delta)$ is defined as follows:

- if $q(a(x_1, x_2)) \to C[q_1(x_1), q_2(x_2)] \in \delta$, $t_1, t_2 \in \mathcal{T}_\Sigma$ and $t/v = q(a(t_1, t_2))$ then, $t \to t[v \leftarrow C[q_1(t_1), q_2(t_2)]]$ and
- if $q(\#) \to t' \in \delta$ and $t/v = q(\#)$ then $t \to t[v \leftarrow t']$.

The reflexive and transitive closure of $\to$ is denoted by $\to^*$. The transformation induced by T, also denoted as T, is the relation defined by: $T = \{(t, \tau) \mid q_0(t) \to^* \tau, t \in \mathcal{T}_\Sigma, \tau \in \mathcal{T}_\Delta, q_0 \in Q_0\}$. The domain $Dom(T)$ of T is the set $\{t \in \mathcal{T}_\Sigma \mid \exists \tau \in \mathcal{T}_\Delta.(t, \tau) \in T\}$. The image of a subset $L \subseteq \mathcal{T}_\Sigma$ by T is the set $T(L) = \{\tau \in \mathcal{T}_\Delta \mid \exists t \in L.(t, \tau) \in T\}$. A linear top-down tree transducer $T = (Q, \Sigma, q_0, \delta)$ is *deterministic* if (1) Q_0 is singleton and (2) for each state q and $a \in \Sigma$, there is at most one rule that contains q and a in its left-hand side. When T is deterministic, we can regard T as a function from $Dom(T)$ to $\mathcal{T}_\Delta$ since $T(t)$ has exactly one tree $\tau \in \mathcal{T}_\Delta$ for each $t \in Dom(T)$. Figure 5 shows an example of a linear top-down tree transducer T and application of its transduction rules to an input tree t.

A linear deterministic top-down tree transducer is abbreviated as LDTT.

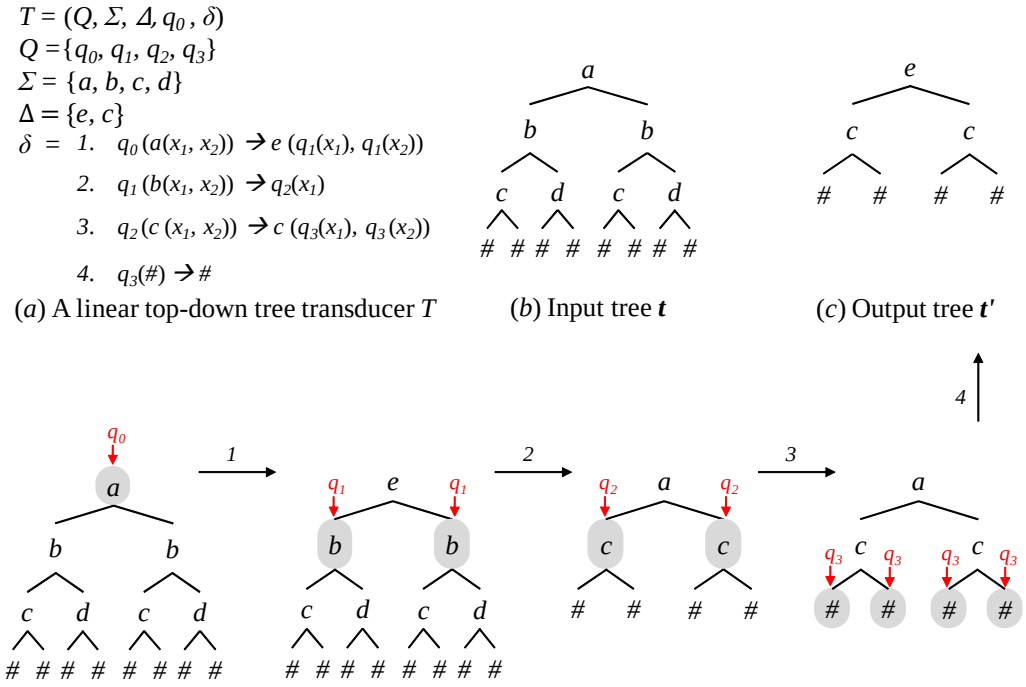


Figure 5. A linear top-down tree transducer and a transduction. *This figure shows a simple example of linear top-down tree transducer T having δ as a set of transduction rules and q_0 as the initial state. An input tree t is transformed into an output tree t'.*

2.3.2 Linear bottom-up tree transducers

A bottom-up tree transducer [2] processes each node of an iput tree in bottom-up manner, according to the subtree rooted by the node.

Definition 3 *A linear bottom-up tree transducer T is a 5-tuple $(Q, \Sigma, \Delta, Q_f, \delta)$, where*

- *Q is a finite set of states,*
- *Σ is an input alphabet,*
- *Δ is an output alphabet,*
- *$Q_f \subseteq Q$ is the set of final states, and*

- *δ is a set of transduction rules of the following two types:*

$$a(q_1(x_1), q_2(x_2)) \to q(C) \quad or \quad \# \to q(t')$$

where $a \in \Sigma$, $q_1, q_2, q \in Q$, $C \in \mathcal{C}(\Delta, \mathcal{X}_2)$, and $t' \in \mathcal{T}_\Delta$.

The move relation $\to$ by a linear bottom-up tree transducer $T = (Q, \Sigma, \Delta, Q_f, \delta)$ is defined as follows:

- if $a(q_1(x_1), q_2(x_2)) \to q(C) \in \delta$, $t_1, t_2 \in \mathcal{T}_\Delta$ and $t/v = a(q_1(t_1), q_2(t_2))$ then, $t \to t[v \leftarrow q(C[t_1, t_2])]$ and
- if $\# \to q(t') \in \delta$ and $t/v = \#$ then $t \to t[v \leftarrow q(t')]$.

The transformation induced by T, also denoted as T, is the relation defined by: $T = \{(t, \tau) \mid t \to^* q_f(\tau), t \in \mathcal{T}_\Sigma, \tau \in \mathcal{T}_\Delta, q_f \in Q_f\}$. The definition of the domain and image of T is the same as the top-down case. A linear bottom-up tree transducer T is *deterministic* if there are no two rules with the same left-hand side. When T is deterministic, we can regard T as a function from $Dom(T)$ to $\mathcal{T}_\Delta$ as with the top-down case. Figure 6 shows an example of a linear bottom-up tree transducer and application of its transduction rules to an input tree t.

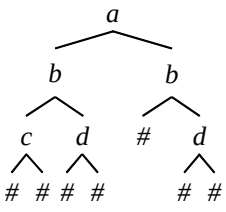
A linear deterministic bottom-up tree transducer is abbreviated as LDBT.

2.3.3 Linear top-down tree transducers with regular look-ahead

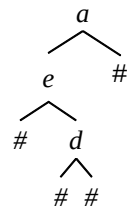
The regular look-ahead [3] allows that, when processing each node in top-down manner, it checks whether the subtrees rooted by its children are in the specified regular tree languages and then processes the node according to the result. The ability naturally corresponds to the XPath filters that used in XSLT programs.

$T = (Q, \Sigma, \Delta, Q_f, \delta)$
$Q = \{ q_1, q_2, q_3 \}$
$\Sigma = \{a, b, c, d\}$
$\Delta = \{a, e, d\}$
$Q_f = \{q_1\}$
δ =
1. $\# \rightarrow q_3(\#)$
2. $c\,(q_3(x_1), q_3\,(x_2)) \rightarrow q_2(\#)$
3. $d\,(q_3(x_1), q_3\,(x_2)) \rightarrow q_2\,(d\,(x_1, x_2))$
4. $b\,(q_3(x_1), q_2(x_2)) \rightarrow q_1\,(x_1)$
5. $a\,(q_1(x_1), q_1(x_2)) \rightarrow q_1\,(a(x_1, \#))$
6. $b\,(q_2(x_1), q_2(x_2)) \rightarrow q_1\,(e(x_1, x_2))$

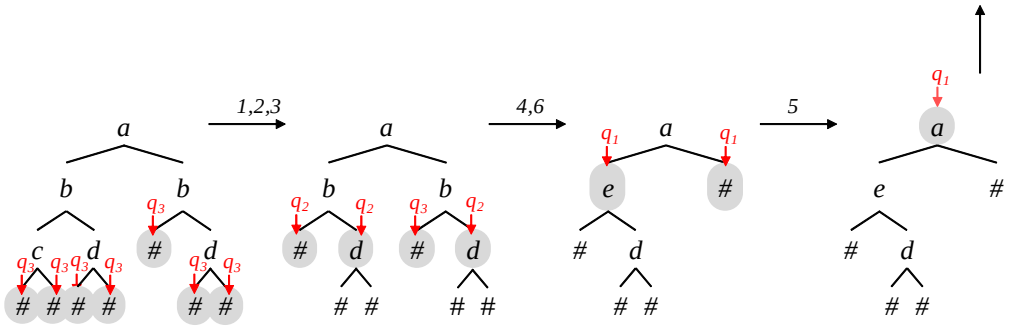
(*a*) A linear bottom-up tree transducer *T*



(*b*) Input tree ***t***



(*c*) Output tree ***t'***



(*d*) Application of bottom-up transduction rules δ of ***t*** into ***t'*** by ***T***

Figure 6. A linear bottom-up tree transducer and a transduction. *This figure shows a simple example of linear bottom-up tree transducer T having δ as a set of transduction rules and q_1 as the final state. An input tree t is transformed into an output tree t'.*

Definition 4 *A top-down tree transducer T with regular look-ahead is a 5-tuple $(Q, \Sigma, \Delta, Q_0, \delta)$, where*

- *Q is a finite set of states,*
- *Σ is an input alphabet,*
- *Δ is an output alphabet,*
- *$Q_0 \subseteq Q$ is the set of initial states, and*

- *δ is a set of transduction rules of the following two types:*

$$q(a(x_1 : A_1, x_2 : A_2)) \to C[q_1(x_1), q_2(x_2)] \quad or \quad q(\#) \to t'$$

where $a \in \Sigma$, $q_1, q_2, q \in Q$, $C \in \mathcal{C}(\Delta, \mathcal{X}_2)$, $t' \in \mathcal{T}_\Delta$, and A_1, A_2 are TAs over Σ.

The move relation $\to$ by a linear top-down tree transducer $T = (Q, \Sigma, \Delta, Q_0, \delta)$ with regular look-ahead is defined as follows:

- if $q(a(x_1 : A_1, x_2 : A_2)) \to C[q_1(x_1), q_2(x_2)] \in \delta$, $t_1 \in L(A_1)$, $t_2 \in L(A_2)$, and $t/v = q(a(t_1, t_2))$ then, $t \to t[v \leftarrow C[q_1(t_1), q_2(t_2)]]$ and
- if $q(\#) \to t' \in \delta$ and $t/v = q(\#)$ then $t \to t[v \leftarrow t']$.

The transformation induced by T, also denoted as T, is the relation defined by: $T = \{(t, \tau) \mid q_0(t) \to^* \tau, t \in \mathcal{T}_\Sigma, \tau \in \mathcal{T}_\Delta, q_0 \in Q_0\}$. A linear top-down tree transducer T with regular look-ahead is *deterministic* if (1) Q_0 is singleton and (2) there are no two different rules $q(a(x_1 : A_1, x_2 : A_2)) \to r_1$ and $q(a(x_1 : A'_1, x_2 : A'_2)) \to r_2$ such that $L(A_1) \cap L(A'_1) \neq \emptyset$ and $L(A_2) \cap L(A'_2) \neq \emptyset$. When T is deterministic, we can regard T as a function from $Dom(T)$ to $\mathcal{T}_\Delta$. Figure 7 shows an example of a linear top-down tree transducer with regular look-ahead. Note that the definition of A_1, A_2 and A_* are omitted in the figure. As in the figure, a linear top-down tree transducer with regular look-ahead can make the decision whether it can apply a rule depending on the membership test $t_i \in L(A_i)$ where t_1 and t_2 are the left and right children of a scanned node and A_1 and A_2 are tree automata. In other words, A_1 and A_2 play a role of type declaration of the left and right subtrees of the current input.

A linear deterministic top-down tree transducer with regular look-ahead is abbreviated as LDTT^R.

Note that LDTT and LDBT are incomparable classes, and LDTT^R is a superclass of both LDTT and LDBT.

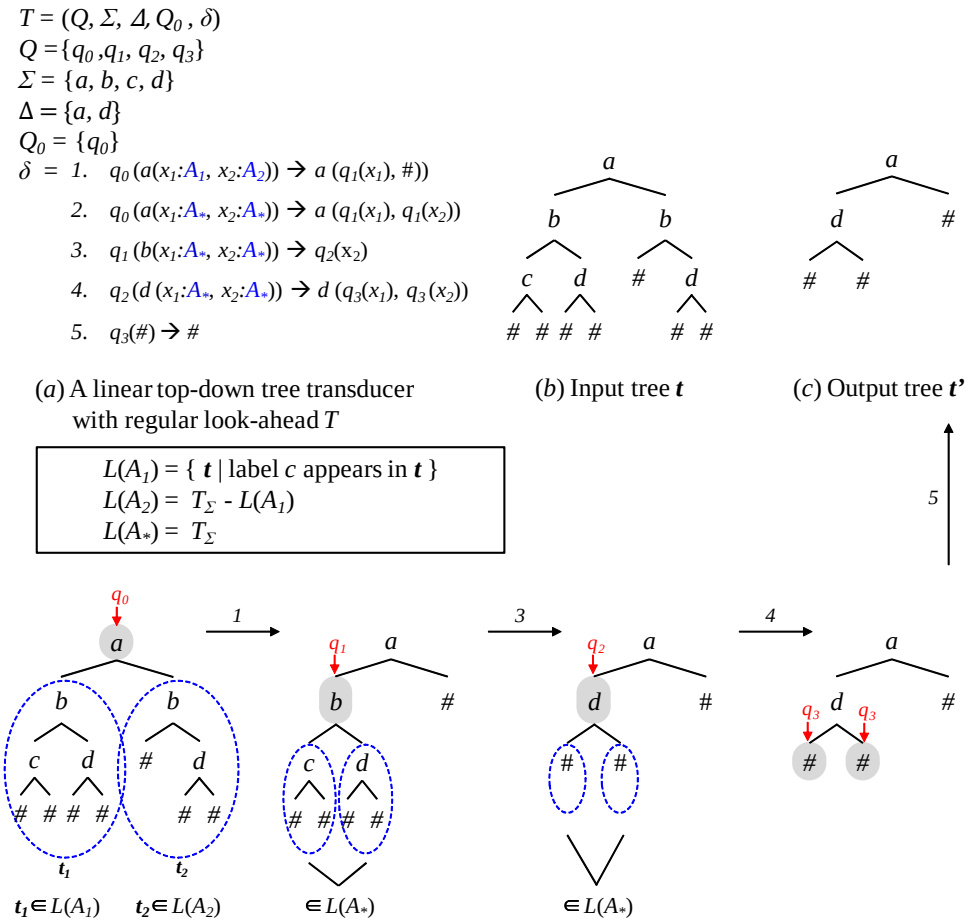


Figure 7. A linear top-down tree transducer with regular look-ahead and a transduction. *This figure shows a simple example of linear top-down tree transducer with regular look-ahead T having δ as a set of transduction rules and q_0 as the initial state. An input tree t is transformed into an output tree t'.*

3. Schema k-secrecy problem

In this section, we give the definition of a static analysis problem for k-secrecy [4], schema k-secrecy against inference attacks. We first introduce the concept of k-secrecy, and then define schema k-secrecy.

3.1 k-secrecy

In order to handle the problem of unauthorized inference, a security criterion called k-secrecy against inference attacks on XML database has been proposed [4]. For an integer $k > 1$ (or $k = \infty$), k-secrecy means that attackers cannot narrow down the candidates for the value of the sensitive information to $k-1$ (or finite) by using the results of given authorized queries and schema information. Intuitively, the larger the candidate set is, the safer the database is, because the information about sensitive data the attacker can get becomes more ambiguous.

Definition 5 (k-secrecy) *Given*

- *a database instance t,*
- *a schema of a database A_G,*
- *authorized queries $T_{a1}, \ldots, T_{an}$, and*
- *unauthorized query T_s to extract the secret information in t,*

t is k-secret if $|C(t)| \geq k$ where

$$C(t) = \{T_s(t') \mid t' \in L(A_G), T_{a1}(t') = T_{a1}(t), \ldots, T_{an}(t') = T_{an}(t)\}.$$

In particular, t is ∞-secret if $C(t)$ is infinite.

Figure 8 illustrates the definition of k-secrecy. In the above definition, $C(t)$ means the candidates of secret information. We assume that attackers know the schema, authorized queries and their results on t, but do not know the instance t and the secret information $T_s(t)$. An attacker uses the available information from the result of authorized queries $T_a(t)$ to compute the inverse image $T_a^{-1}(T_a(t))$, then extracts from trees $T_a^{-1}(T_a(t))$ tree that conform to a given schema A_G.

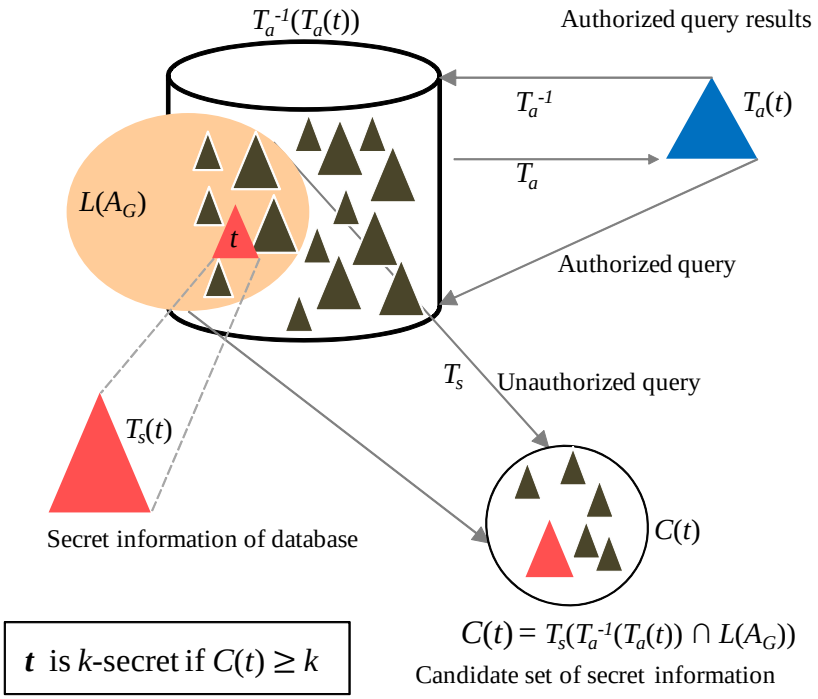


Figure 8. Definition of k-secrecy

After that, the attacker takes the unauthorized query T_s in order to obtain the candidate secret information $C(t)$ of the database. We expect that the set $C(t)$ is the maximal set of candidates such that for any candidate in $C(t)$, attackers cannot distinguish whether it is the true secret data $T_s(t)$ or not.

Instance-level k-secrecy problem, abbreviated as k-ISIA, asks whether a given database instance conforming to a given schema is k-secret or not.

3.2 Schema k-secrecy

So far, a verification algorithm for k-ISIA has been proposed [4]. However, every time an instance is updated, we need to repeatedly apply the verification algorithm in order to maintain the safeness of database (see Figure 9 (a)). In this paper, as another approach to maintaining k-secrecy, we will discuss the problem for schema-level k-secrecy, which is the problem of deciding whether *every instance* conforming to a given schema is k-secret. If we can guarantee that every instance is k-secret, then we do not need to repeatedly verify that a new instance

is k-secret or not every time an instance is updated (see Figure 9 (b)).

The definition of schema k-secrecy is as follows:

Definition 6 (Schema k-secrecy) *Given*

- *a schema of a database A_G,*
- *an authorized query T_a, and*
- *an unauthorized query T_s,*

the database is schema k-secret with respect to (A_G, T_a, T_s) if $|C(t)| \geq k$ holds for every instance t conforming to A_G, where

$$C(t) = T_s(T_a^{-1}(T_a(t)) \cap L(A_G)).$$

In particular, the database is ∞-secret with respect to (A_G, T_a, T_s) if for every instance t conforming to A_G, $C(t)$ is infinite.

We simply say that the database is schema k-secret (or schema ∞-secret) when A_G, T_a, and T_s are clear from the context. The assumption of the attacker's behavior is almost the same as in Definition 5. The difference is that schema k-secrecy requires that not only a particular instance, but also every instance t conforming to a given schema A_G is k-secret (or ∞-secret).

The static analysis problem for k-secrecy, *schema k-secrecy* (or *schema ∞-secrecy*) *against inference attacks* are abbreviated as k-SSIA (or ∞-SSIA). k-SSIA (or ∞-SSIA) asks whether, given a database schema A_G, an authorized query T_a, and an unauthorized query T_s, the database is schema k-secret (or schema ∞-secret).

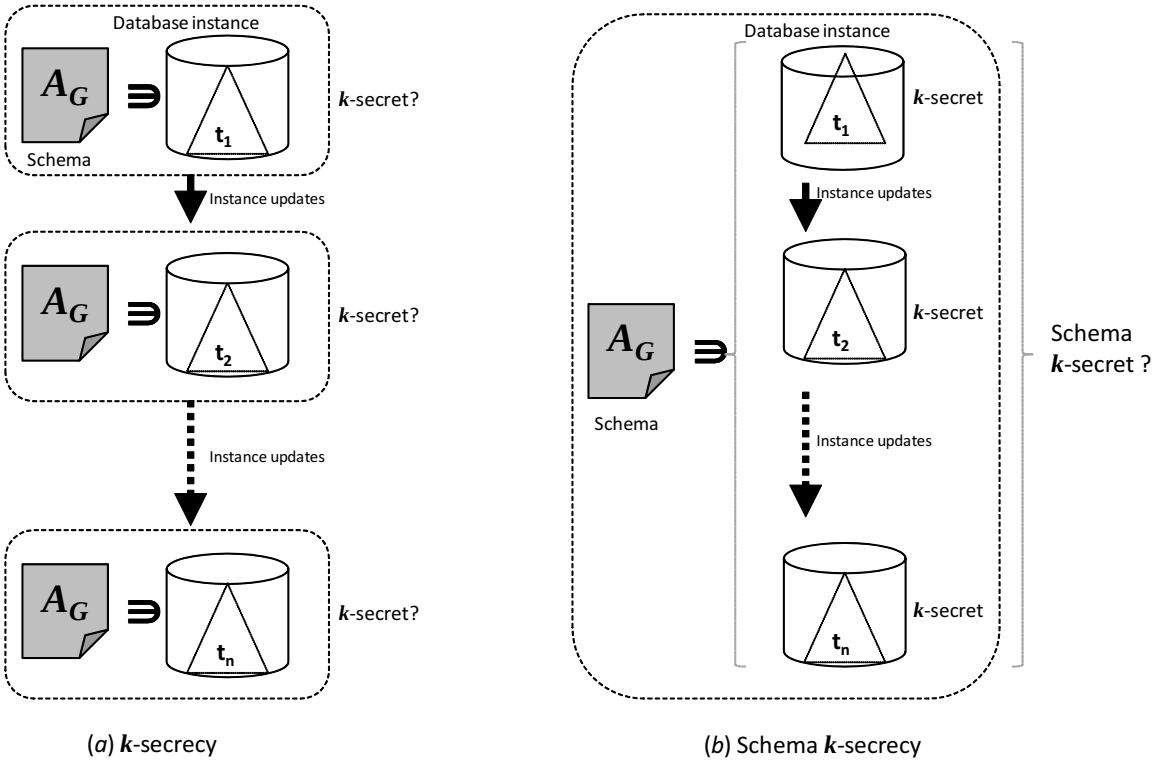


Figure 9. Verification of instance-level k-secrecy and schema-level k-secrecy. *(a) shows the instance-level k-secrecy verification. Even if an instance is guaranteed to be k-secret by k-ISIA algorithm, we need to repeatedly apply the algorithm every time an instance is updated. On the other hand, (b) shows the schema-level k-secrecy verification. Once we know every instance is k-secret by k-SSIA algorithm, we do not need to apply the algorithm after instances are updated.*

4. Undecidability result

It is desirable to decide whether the database is schema k-secret for some constant k, that is, whether attackers cannot narrow down by inference the set of candidates for the value of the sensitive information to $k-1$ for any database instance. Unfortunately, k-SSIA is undecidable for any constant $k > 1$, which will be shown in the following theorem.

Theorem 1 *k-SSIA is undecidable for any integer $k > 1$, where queries are represented by deterministic rational transducers on words (DRTW).*

Proof. We show the undecidability of 2-SIAA by reduction from PCP. The undecidability of k-SSIA for $k > 2$ can be shown in almost the same way. Here, we use a regular expression to represent a schema and deterministic rational transducers on words (DRTW) [2] to represent queries. The class of languages represented by regular expressions, when regarded as tree languages, is a proper subclass of the tree languages recognized by tree automata and a DRTW is a special case of a linear deterministic top-down tree transducer.

Consider any instance (W, X) of PCP, where $W = \langle w_1, w_2, \ldots, w_n\rangle$ and $X = \langle x_1, x_2, \ldots, x_n\rangle$ are both n-tuples of words over an alphabet Σ. Let $[n] = \{1, \ldots, n\}$ and we assume that $\Sigma \cap [n] = \emptyset$. We construct the following regular expression A_G over $\Sigma \cup [n]$ as a schema:

$$A_G = (1w_1 \mid 2w_2 \mid \ldots \mid nw_n)^+ \mid (1x_1 \mid 2x_2 \mid \ldots \mid nx_n)^+.$$

For example, let $W = \langle abc, d\rangle$ and $X = \langle a, bcd\rangle$, then we construct $A_G = (1abc \mid 2d)^+ \mid (1a \mid 2bcd)^+$. Let $L(A_G)$ be the language represented by A_G. Next, let T_a be the authorized query that extracts only symbols in $[n]$ from an input word, and let T_s be the unauthorized query that extracts only symbols in Σ from an input word. For example, given $t = 1abc2d$ as an input word, $T_a(t) = 12$ and $T_s(t) = abcd$. Both T_a and T_s can be defined by DRTW with one state.

We now show that $|T_s(T_a^{-1}(T_a(t)) \cap L(A_G))| \geq 2$ for any $t \in L(A_G)$ if and only if (W, X) has no solution. Take any $t = i_1 w_{i_1} i_2 w_{i_2} \cdots i_m w_{i_m} \in L(A_G)$ (or $t = i_1 x_{i_1} i_2 x_{i_2} \cdots i_m x_{i_m} \in L(A_G)$), and then it holds that

$$T_a^{-1}(T_a(t)) \cap L(A_G) \;=\; \{i_1 w_{i_1} i_2 w_{i_2} \cdots i_m w_{i_m},\ i_1 x_{i_1} i_2 x_{i_2} \cdots i_m x_{i_m}\}.$$

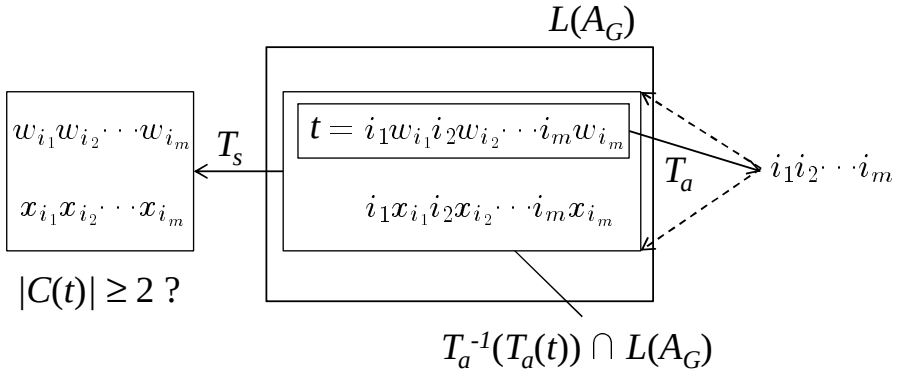


Figure 10. Reduction from PCP to 2-SSIA

Thus,

$$C(t) \;=\; T_s(T_a^{-1}(T_a(t)) \cap L(A_G)) = \{w_{i_1}w_{i_2}\cdots w_{i_m},\ x_{i_1}x_{i_2}\cdots x_{i_m}\}.$$

This situation is depicted in Figure 10.

If the instance (W, X) of PCP has a solution $i_1 i_2 \cdots i_m$, then $|C(t)| = 1$ where $t = i_1 w_{i_1} i_2 w_{i_2} \cdots i_m w_{i_m}$ or $t = i_1 x_{i_1} i_2 x_{i_2} \cdots i_m x_{i_m}$ because $w_{i_1}w_{i_2}\cdots w_{i_m} = x_{i_1}x_{i_2}\cdots x_{i_m}$. Conversely, if the instance (W, X) of PCP has no solution, then $|C(t)| = 2$ for any $t \in L(A_G)$ because $w_{i_1}w_{i_2}\cdots w_{i_m} \neq x_{i_1}x_{i_2}\cdots x_{i_m}$ for any $i_1 i_2 \cdots i_m \in [n]^+$.

Thus, we have proved that 2-SSIA is undecidable. □

The above theorem says that 2-SSIA is undecidable when queries are given by DTRW. It is not difficult to show that k-SSIA is undecidable for any finite $k \geq 2$. Note that DTRW is a subclass of both linear deterministic top-down tree transducer (LDTT) and linear deterministic bottom-up tree transducer (LDBT). Hence, for any finite k, k-SSIA is undecidable when queries are given by LDTT, LDBT or LDTT^R. On the other hand, k-SSIA is decidable when $k = \infty$, which will be shown in the next section.

5. Decidability result

In this section, we show that ∞-SSIA is decidable for linear deterministic top-down tree transducers (LDTT). We give an algorithm for deciding ∞-SSIA in section 5.1, then show the correctness of the algorithm in section 5.2. The time complexity of our algorithm is given in section 5.3. Moreover, we give decision algorithms extended to the case that queries are given by LDBT or LDTT^R in section 5.4. Lastly, we discuss an extension of the algorithm to the case that there are multiple authorized queries in section 5.5.

5.1 Decision Algorithm

5.1.1 Overview

First, we explain the key idea of our algorithm before giving the detail. For simplicity, we consider only authorized/unauthorized queries but not schemas here. More precisely, we fix a schema to be a tree automaton which accepts any tree. We assume that a schema is arbitrarily given as a tree automaton when we give the detail of our algorithm in section 5.1.2.

To begin with, we briefly look at a property of linear deterministic top-down tree transducers. Consider a pair $(t, T(t))$ of input and output trees by a linear deterministic top-down tree transducer T. We can see that for each node v of $t \in Dom(T)$, v either corresponds to some part of $T(t)$ constructed by applying some rule of T to the node, or does not correspond to any part of $T(t)$, that is, v is deleted by T. For example, let T be the linear top-down tree transducer which has the transduction rules listed in Figure 11. Now consider that the left tree is transformed into the right tree in Figure 11 by T. Then, the nodes of the left tree in the circle-dot-lines correspond to the parts of the right tree according to the applied rules of T, respectively. On the other hand, the nodes in the rectangle-dot-line are deleted by T. The subtree $c(\#, \#)$ is deleted because $q_1(b(x_1, x_2)) \to \#$ is applied to the parent node b and thus the subtrees at x_1 and x_2 are ignored. We call a rule like $q_1(b(x_1, x_2)) \to \#$ a *subtree-deleting* rule. The node b which is the right child of the root is deleted by T because $q_2(b(x_1, x_2)) \to q(x_1)$ is applied to the node and thus no symbol is output according to the node by the rule. We call a rule like $q_2(b(x_1, x_2)) \to q(x_1)$ an *erasing* rule.

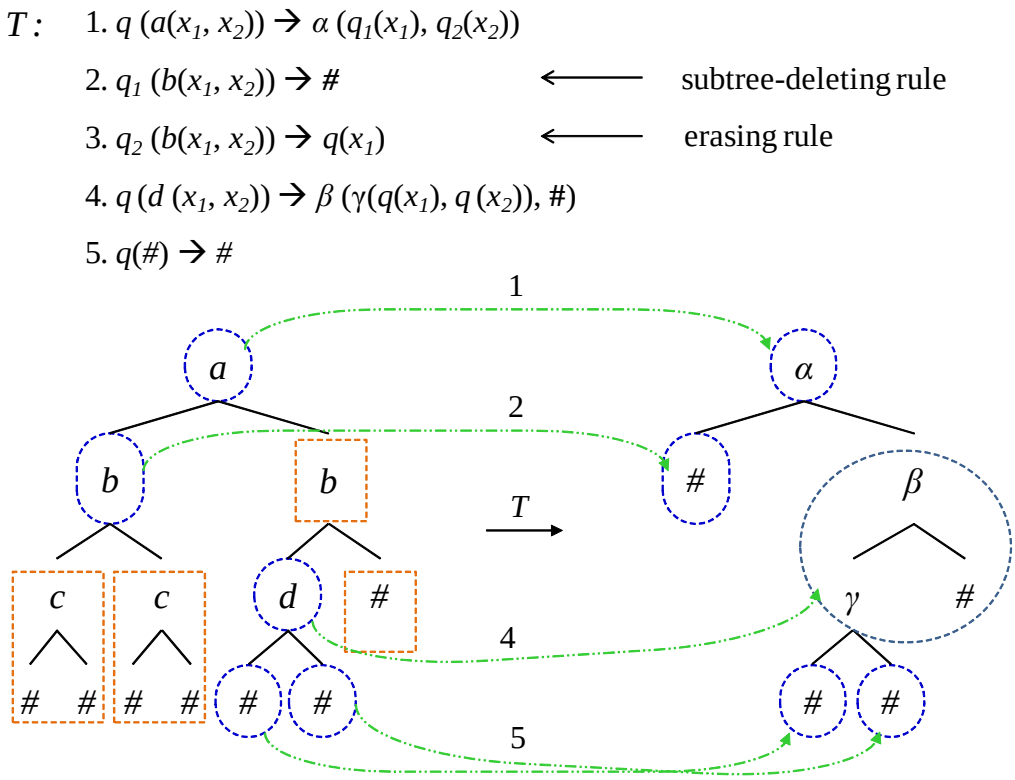


Figure 11. Deleted and undeleted nodes by T

As we shall explain later, we can construct the following unambiguous tree automaton A_a with respect to a query T_a:

- A_a recognizes the domain of T_a.
- The state set of A_a consists of the states of T_a (called undeleted states) plus a special state called the deleted state .
- For any input tree t and any node v of the tree, if the run of A_a on t assigns a deleted state to v, v is deleted by T_a; otherwise v corresponds to some part of $T_a(t)$.

By looking at the state assigned to a node by A_a, we can distinguish whether the node is deleted by subtree-deleting rules of T_a. Similarly, let A_s denote the unambiguous tree automaton with respect to T_s. Also, we can find nodes deleted

by erasing rules of T_a, by checking which rules are applied to the nodes. Thus, we can distinguish between deleted and undeleted nodes on T_a.

Now we assume that A_a has some deleted state q satisfying the following condition: q is recursive through a proper path such that all the states in the path are deleted states. Then, since every part of input trees corresponding to such a cycle path is deleted by T_a, an infinite number of trees obtained by pumping only the part are transformed into the same tree by T_a. On the other hand, if q is recursive through a path such that at least one state in the path is an undeleted state, an infinite number of trees obtained by pumping at a node assigned q by A_s are transformed into distinct trees.

From the above observation, our algorithm finds the following subset Q' of state pairs from the product automaton of A_a and A_s: $q \in Q'$ is recursive through a proper path such that all the states in the path contain deleted states of A_a and at least one state in the path contains an undeleted state of A_s. For any tree t such that the product automaton assigns $q \in Q'$ to some node of the tree, $T_a^{-1} \circ T_a(t)$ is infinite because q is recursive through the deleted path of T_a, and also $T_s \circ T_a^{-1} \circ T_a(t)$ is infinite because the deleted path of T_a is an undeleted path of T_s. Thus, any tree t such that the product automaton assigns $q \in Q'$ to some node of the tree to is ∞-secret. Let L_{inf} denote such trees.

Note that $t \notin L_{inf}$ may be ∞-secret. If there is some tree $t_{sec} \in L_{inf}$ such that $T_a(t) = T_a(t_{sec})$, t is ∞-secret because $T_s \circ T_a^{-1} \circ T_a(t) = T_s \circ T_a^{-1} \circ T_a(t_{sec})$ is infinite. Hence, let L_{fin} be the complement of L_{inf}, then we claim that $T_a(L_{fin}) \subseteq T_a(L_{inf})$ if and only if schema ∞-secrecy is satisfied.

In summary, our algorithm mainly consists of four steps:

1. Construct the tree automata A_a and A_s from T_a and T_s.
2. Compute the subset Q' of state pairs from the product automaton of A_a and A_s.
3. Compute the sets L_{inf} and L_{fin} of trees. More precisely, construct tree automata A_{inf} and A_{fin} which recognize them, respectively.
4. Decide whether $T_a(L_{fin}) \subseteq T_a(L_{inf})$.

5.1.2 Detailed construction

Let $T_a = (Q_a, \Sigma, \Delta_1, q_a^0, \delta_a)$ and $T_s = (Q_s, \Sigma, \Delta_2, q_s^0, \delta_s)$ be linear deterministic top-down tree transducers as authorized and unauthorized queries, respectively, and let $A_G = (Q, \Sigma, Q_F, \delta)$ be an unambiguous tree automaton as an XML schema. We assume that $Dom(T_a) \supseteq L(A_G)$ and $Dom(T_s) \supseteq L(A_G)$, and that for each $q \in Q$, there is no two distinct rule in δ with same left-hand sides that contain different symbols in Σ in the right-hand sides without loss of generality. Our algorithm works as follows:

Step 1. Construct the following two TAs A_a and A_s from T_a and T_s:

- $A_a = (Q_a \cup \{q_a^d\}, \Sigma, \{q_a^0\}, \delta_a')$ where $q_a^d \notin Q_a$, and δ_a' is the smallest set such that

  - for each $r = q(\sigma(x_1, x_2)) \to C[q_1(x_1), q_2(x_2)] \in \delta_a$ where $q, q_1, q_2 \in Q_a$, $\sigma \in \Sigma$, and $C \in \mathcal{C}(\Delta_1, \mathcal{X}_2)$,

    $$q \to \sigma(\tilde{q}_1, \tilde{q}_2) \in \delta_a'$$

    where for each $i \in \{1, 2\}$,

    if RHS(r) contains x_i then $\tilde{q}_i = q_i$; otherwise $\tilde{q}_i = q_a^d$.
  - for each $q(\#) \to u$ where $q \in Q$ and $u \in \mathcal{T}_{\Delta_1}$, $q \to \# \in \delta_a'$.
  - for each $\sigma \in \Sigma$, $q_a^d \to \sigma(q_a^d, q_a^d) \in \delta_a'$.
  - $q_a^d \to \# \in \delta_a'$.

- $A_s = (Q_s \cup \{q_s^d\}, \Sigma, \{q_s^0\}, \delta_s')$ where $q_s^d \notin Q_s$, and δ_s' is the smallest set such that

  - for each $r = q(\sigma(x_1, x_2)) \to C[q_1(x_1), q_2(x_2)] \in \delta_s$ where $q, q_1, q_2 \in Q_s$, $\sigma \in \Sigma$, and $C \in \mathcal{C}(\Delta_2, \mathcal{X}_2)$,

    $$q \to \sigma(\tilde{q}_1, \tilde{q}_2) \in \delta_s'$$

    where for each $i \in \{1, 2\}$,

    if RHS(r) contains x_i then $\tilde{q}_i = q_i$; otherwise $\tilde{q}_i = q_s^d$.

- for each $q(\#) \to u$ where $q \in Q$ and $u \in \mathcal{T}_{\Delta_1}$, $q \to \# \in \delta'_s$.
- for each $\sigma \in \Sigma$, $q_s^d \to \sigma(q_s^d, q_s^d) \in \delta'_s$.
- $q_s^d \to \# \in \delta'_s$.

Step 2. Construct the product tree automaton of A_a, A_s, and A_G. More specifically, construct the following tree automaton $A_I = (Q'_a \times Q'_s \times Q, \Sigma, \{q_a^0\} \times \{q_s^0\} \times Q_F, \delta_I)$ from $A_a = (Q'_a, \Sigma, \{q_a^0\}, \delta'_a)$, $A_s = (Q'_s, \Sigma, \{q_s^0\}, \delta'_s)$, and $A_G = (Q, \Sigma, Q_F, \delta)$: $(q_a, q_s, q) \to \sigma((q_a^1, q_s^1, q^1), (q_a^2, q_s^2, q^2)) \in \delta_I$ if and only if $q_a \to \sigma(q_a^1, q_a^2) \in \delta'_a$, $q_s \to \sigma(q_s^1, q_s^2) \in \delta'_s$, and $q \to \sigma(q^1, q^2) \in \delta$. Since $Dom(T_a) \supseteq L(A_G)$ and $Dom(T_s) \supseteq L(A_G)$, $L(A_G) = L(A_I)$, and for each $t \in L(A_I)$ and $v \in V(t)$, we have $r(A_I, t)(v) = (r(A_a, t)(v), r(A_s, t)(v), r(A_G, t)(v))$.

Step 3. Let $Q_I = Q'_a \times Q'_s \times Q$ be the set of states of A_I. Compute two subsets $\mathcal{Q}_a^D$ and $\mathcal{Q}_s^D$ of Q_I as follows:

$$\begin{aligned}
\mathcal{Q}_a^D &= \{(q_a, q_s, q) \in Q_I \mid q_a = q_a^d\} \\
&\quad \cup \{(q_a, q_s, q) \in Q_I \mid (q_a, \sigma) \in D_a, (q, \sigma) \in O\}, \\
\mathcal{Q}_s^D &= \{(q_a, q_s, q) \in Q_I \mid q_s = q_s^d\} \\
&\quad \cup \{(q_a, q_s, q) \in Q_I \mid (q_s, \sigma) \in D_s, (q, \sigma) \in O\}
\end{aligned}$$

where

$$\begin{aligned}
D_a &= \{(q_a, \sigma) \mid \exists i \in \{1, 2\}. q_a(\sigma(x_1, x_2)) \to q'(x_i) \in \delta_a\}, \\
D_s &= \{(q_s, \sigma) \mid \exists i \in \{1, 2\}. q_s(\sigma(x_1, x_2)) \to q'(x_i) \in \delta_s\}, \\
O &= \{(q, \sigma) \mid q \to \sigma(q_1, q_2) \in \delta\}.
\end{aligned}$$

Step 4. Let $Q' = \{q_I \in Q_I \mid q_I$ is recursive through a proper path such that all the states in the path are in $\mathcal{Q}_a^D$
and some state in the path is not in $\mathcal{Q}_s^D\}$.

Step 5. Let $A_{fin} = (Q_I - Q', \Sigma, \{q_a^0\} \times \{q_s^0\} \times Q^0 - Q', \delta'_I)$ where δ'_I is obtained from δ_I by removing rules containing a state in Q'.

Step 6. Let A_{inf} be a tree automaton such that $L(A_{inf}) = L(A_I) - L(A_{fin})$.

Note that the class of regular tree languages is closed under complementation [2] and, given any tree automaton A, a tree automaton which recognizes the complement of $L(A)$ is constructible.

Step 7. Decide whether $T_a(L(A_{fin})) \subseteq T_a(L(A_{inf}))$. If $T_a(L(A_{fin})) \subseteq T_a(L(A_{inf}))$ then output "schema ∞-secret"; Otherwise, output "Not schema ∞-secret."

Note that any linear top-down tree transducer T preserves regularity, that is, for any regular tree language L, $T(L)$ is also a regular language, and a tree automaton which recognizes $T(L)$ is constructible. Also, the inclusion problem is decidable for regular tree languages.

Note: In step 1, if the variable x_i disappear in RHS(r), we use the dummy state q_a^d instead of q_i so that A_a goes through the subtree substituted into x_i by the rule $q_a^d \to \sigma(q_a^d, q_a^d)$. The role of state q_s^d is the same as q_a^d. In step 3, $(q_a, \sigma) \in D_a$ means that T_a reads input symbol σ and produces no output symbols in state q_a. Thus, if A_I is in state $(q_a, q_s, q) \in \mathcal{Q}_a^D$, A_I is in a state such that either T_a outputs nothing or T_a already deleted the part of the input. Assume that $q_I = (q_a, q_s, q) \in Q'$ in step 4. q_I is in a nonempty path such that T_a only consumes inputs while T_s produces at least one output symbol. A_{fin} constructed in step 5 is the tree automaton obtained from the product automaton A_I by removing all states in Q'.

5.2 Correctness

We show the correctness of our algorithm. We give several lemmas for the proof of the correctness.

Lemma 1 *Let t be an arbitrary tree in $L(A_I)$. For any node v_1 and any descendant v_2 of v_1 in t such that*

- $r(A_I, t)(v_1) = r(A_I, t)(v_2)$ *and*
- *every node v that appears in the path from v_1 to v_2 (inclusive) is mapped to a state in $\mathcal{Q}_a^D$ (rsp. in $\mathcal{Q}_s^D$) by $r(A_I, t)$,*

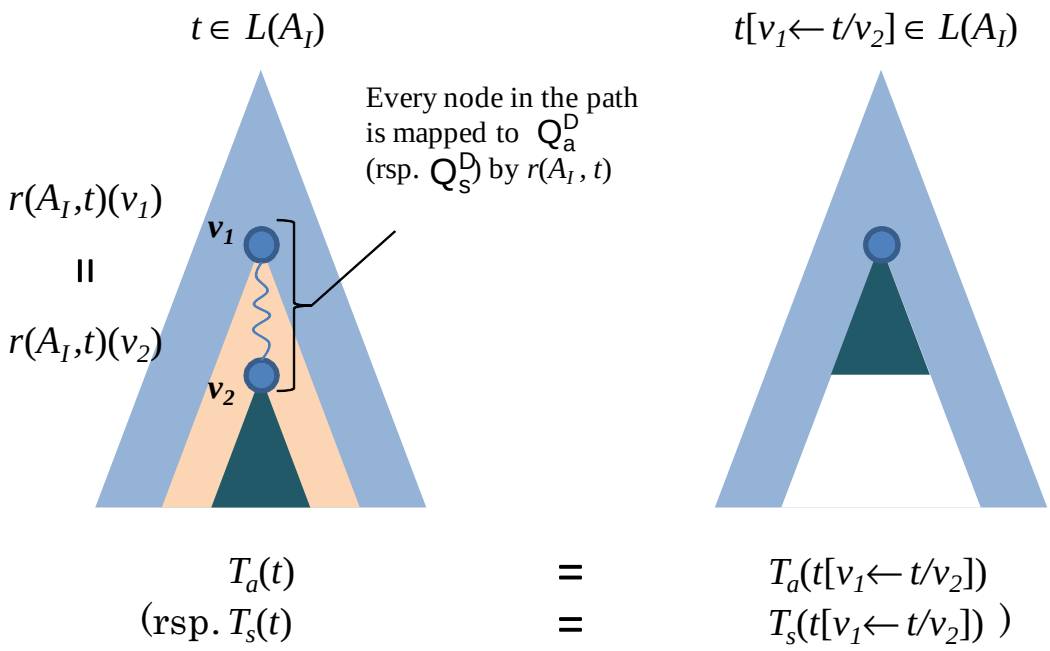


Figure 12. Lemma 1

then it holds that $T_a(t) = T_a(t[v_1 \leftarrow t/v_2])$ *(rsp.* $T_s(t) = T_s(t[v_1 \leftarrow t/v_2])$*) (Figure 12).*

Proof. Let t be an arbitrary tree in $L(A_I)$ and let v be an arbitrary node in t. By definition of A_a, A_s, and A_I, we have the following two facts:

- If $r(A_I, t)(v) = (q_a^d, q_s, q) \in \mathcal{Q}_a^D$ then for every descendant v' of v, $r(A_I, t)(v') = (q_a^d, q_s', q') \in \mathcal{Q}_a^D$ for some $q_s' \in Q_s$ and $q' \in Q$.
- If $r(A_I, t)(v) = (q_a, q_s, q) \in \mathcal{Q}_a^D$ where $q_a \neq q_a^d$ then for at least one v' of the two children of v, $r(A_I, t)(v') = (q_a^d, q_s', q') \in \mathcal{Q}_a^D$ for some $q_s' \in Q_s$ and $q' \in Q$.

Let v_1 be an arbitrary node in t, and let v_2 be an arbitrary descendant v_2 of v_1. We assume that $r(A_I, t)(v_1) = r(A_I, t)(v_2) \in \mathcal{Q}_a^D$ and that $r(A_I, t)(v) \in \mathcal{Q}_a^D$ for each v that appears in the path from v_1 to v_2. By the facts we stated above, for each node v that is a descendant of v_1 but not a descendant of v_2, $r(A_I, t)(v) \in$

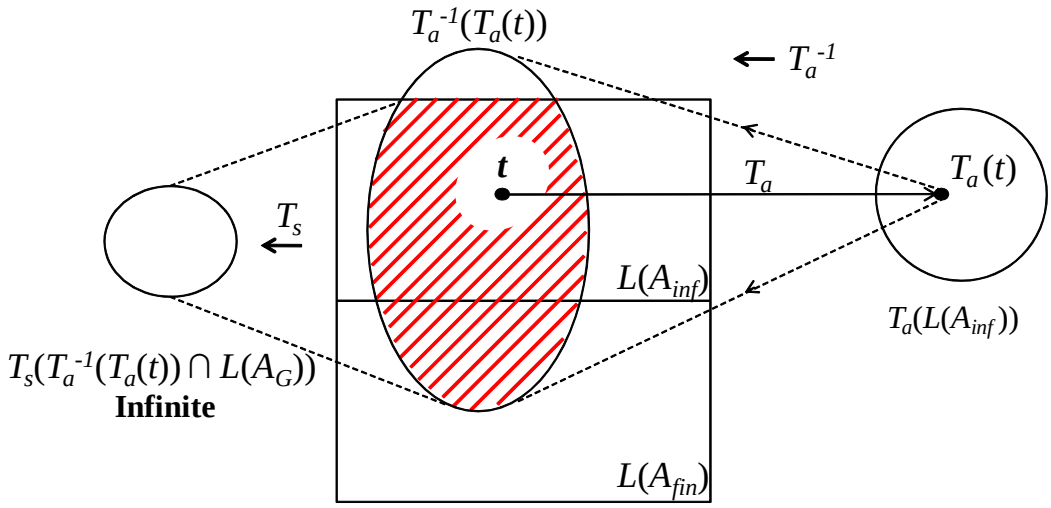


Figure 13. Lemma 2

$\mathcal{Q}_a^D$. Since nodes mapped to states in $\mathcal{Q}_a^D$ by $r(A_I, t)$ are ignored or deleted by T_a, it holds that $T_a(t) = T_a(t[v_1 \leftarrow t/v_2])$.

The case for T_s and $\mathcal{Q}_s^D$ can be shown in the same way. □

Lemma 2 *If $T_a(\tau) \in T_a(L(A_{inf}))$ then $T_s(T_a^{-1}(T_a(\tau)) \cap L(A_G))$ is infinite.*

Proof. Let t be an arbitrary tree in $L(A_{inf})$. Since $L(A_{inf}) = L(A_I) - L(A_{fin})$, there is at least one node v in t such that $r(A_I, t)(v) \in Q'$. By the definition of Q', there is another tree $t' \in L(A_{inf})$ having a node v_1 and its proper descendant v_2 such that

- $r(A_I, t')(v_1) = r(A_I, t')(v_2) = r(A_I, t)(v) \in Q'$,
- all the states in the path from v_1 and v_2 are in $\mathcal{Q}_a^D$ and some state in the path is not in $\mathcal{Q}_s^D$, and
- $t'[v_1 \leftarrow t'/v_2] = t$ and $t/v = t'/v_2$.

By Lemma 1, $T_a(t') = T_a(t)$. On the other hand, $T_s(t') \neq T_s(t)$ because there is some state $q_s \in \mathcal{Q}_s^D$ between v_1 and v_2 in t' and, by the rule which contains q_s in its left-hand side, at least one more node labeled with an output symbol in Δ_2 is emitted by T_s, that is, $|T_s(t')| \geq |T_s(t)| + 1$.

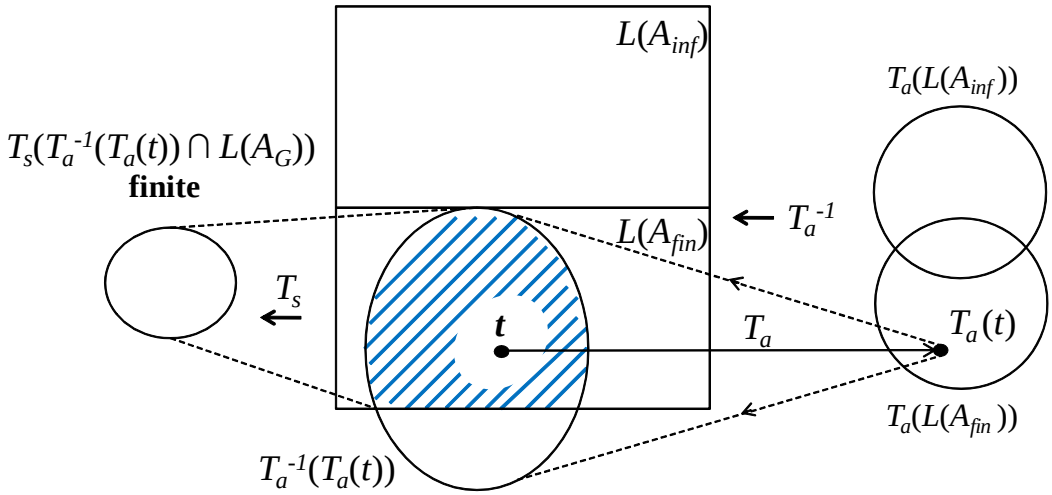


Figure 14. Lemma 3

Summarizing, for any tree $t \in L(A_{inf})$, there is another tree $t' \in L(A_{inf})$ such that $T_a(t') = T_a(t)$ and $|T_s(t')| \geq |T_s(t)| + 1$. Thus, $T_s(T_a^{-1}(T_a(t)) \cap L(A_G))$ is infinite (see Figure 13).

Now assume that $T_a(\tau) \in T_a(L(A_{inf}))$, then there is $t \in L(A_{inf})$ such that $T_a(t) = T_a(\tau)$. Applying the above argument to this specific t, we know $T_s(T_a^{-1}(T_a(\tau)) \cap L(A_G))$ is infinite. □

Lemma 3 *If $t \in L(A_{fin})$ and $T_a(t) \notin T_a(L(A_{inf}))$ then $T_s(T_a^{-1}(T_a(t)) \cap L(A_G))$ is finite (Figure 14).*

Proof. Let t be an arbitrary tree in $L(A_{fin})$. Then, for any node v_1 and any descendant v_2 of v_1 in t such that

- $r(A_I, t)(v_1) = r(A_I, t)(v_2)$ and
- every node v that appears in the path from v_1 to v_2 is mapped to a state in $\mathcal{Q}_a^D$ by $r(A_I, t)$,

it holds not only that $T_a(t) = T_a(t[v_1 \leftarrow t/v_2])$ but also that $T_s(t) = T_s(t[v_1 \leftarrow t/v_2])$. The reason is as follows: Let $\bar{q} = r(A_I, t)(v_1) = r(A_I, t)(v_2)$, then $\bar{q}$ is recursive through a path such that all of the states in the path are in $\mathcal{Q}_a^D$. Since A_{fin} does not have any states in Q', by the construction of Q' and A_{fin} in Steps 4

and 5, we must have that $r(A_I, t)(v_1) = r(A_I, t)(v_2) \in \mathcal{Q}_s^D$ and $r(A_I, t)(v) \in \mathcal{Q}_s^D$ for each v which appears in the path from v_1 to v_2. Thus, by Lemma 1, it holds that $T_s(t) = T_s(t[v_1 \leftarrow t/v_2])$.

Now consider an arbitrary tree $t \in L(A_{fin})$ such that $T_a(t) \notin T_a(L(A_{inf}))$, and let $t_0 = T_a(t)$. For any $t' \in T_a^{-1}(t_0)$, $|\{v \in V(t') \mid r(A_I, t')(v) \notin \mathcal{Q}_a^D\}| \leq |t_0|$ because a node labeled with an output symbol in Δ_1 is emitted every time a rule which contains $q \notin \mathcal{Q}_a^D$ in its left-hand side is applied. Therefore, for each $t' \in T_a^{-1}(t_0)$, there is a tree t'' such that $T_s(t'') = T_s(t')$ and the height of t'' is at most $|t_0| + (|t_0| + 1)|\mathcal{Q}_a^D|$. Thus, there is a subset L_f of the set of all trees of height at most $|t_0| + (|t_0| + 1)|\mathcal{Q}_a^D|$ such that $T_s(L_f) = T_s(T_a^{-1}(t_0))$. Since T_s is functional and L_f is finite, $T_s(T_a^{-1}(t_0) \cap L(A_G))$ is finite. □

We now give a lemma for correctness of our algorithm.

Lemma 4 *$T_a(L(A_{fin})) \subseteq T_a(L(A_{inf}))$ if and only if $T_s(T_a^{-1}(T_a(t)) \cap L(A_G))$ is infinite for every $t \in L(A_G)$.*

Proof. Assume that $T_a(L(A_{fin})) \subseteq T_a(L(A_{inf}))$. We have that $T_a(t) \in T_a(L(A_{inf}))$ for every $t \in L(A_G)$. By Lemma 2, $T_s(T_a^{-1}(T_a(t)) \cap L(A_G))$ is infinite for every $t \in L(A_G)$.

On the other hand, assume that $T_a(L(A_{fin})) \not\subseteq T_a(L(A_{inf}))$, and then there is some $t \in L(A_{fin}) \subseteq L(A_G)$ such that $T_a(t) \notin T_a(L(A_{inf}))$. Thus, by Lemma 3, $T_s(T_a^{-1}(T_a(t)) \cap L(A_G))$ is finite. □

By lemma 4, we obtain the following theorem.

Theorem 2 *∞-SSIA is decidable for linear deterministic top-down tree transducers.*

5.3 Time complexity

Now we show the time complexity of our algorithm. We estimate the time and space complexity of each step.

Step 1. Construct two TAs A_a and A_s from T_a and T_s:

- it takes $O(|T_a|)$ time to construct A_a, and $|A_a|$ is $O(|T_a|)$, and
- it takes $O(|T_s|)$ time to construct A_s, and $|A_s|$ is $O(|T_s|)$.

Step 2. Construct the product TA A_I from A_s, A_a and A_G:

- it takes $O(|A_a| \times |A_s| \times |A_G|)$ time, and
- $|A_I|$ is $O(|A_a| \times |A_s| \times |A_G|)$.

Step 3. Compute $\mathcal{Q}_a^D$ and $\mathcal{Q}_s^D$:

- D_a, D_s, and O can be computed in $O(|T_a|)$, $O(|T_s|)$, and $O(|A_G|)$ time, respectively, and $|D_a| = O(|T_a|)$, $|D_s| = O(|T_s|)$, $|O| = O(|A_G|)$.
- $\mathcal{Q}_a^D$ and $\mathcal{Q}_s^D$ can be computed in $O(|D_a| \times |O| \times |A_I|)$ and $O(|D_s| \times |O| \times |A_I|)$ time, respectively. Since they are subsets of Q_I, $|\mathcal{Q}_a^D| = O(|A_I|)$ and $|\mathcal{Q}_s^D| = O(|A_I|)$.

Step 4. Construct Q' from Q_I: To construct Q' takes $O(|A_I|^3)$ time, and its size is $O(|A_I|)$ because Q' is a subset of Q_I.

Step 5. Construct A_{fin} from Q' and A_I:

- To construct A_{fin} takes $O(|Q_I| \times |A_I|)$ time, and
- $|A_{fin}|$ is $O(|A_I|)$.

Note that A_{fin} can be computed as an unambiguous TA because A_{fin} is unambiguous.

Step 6. Construct A_{inf} such that $L(A_{inf}) = L(A_I) - L(A_{fin})$:

- To compute the complement TA of A_{fin} takes exponential time and its size is exponential in that of A_{fin} at the worst.

Step 7. Decide whether $T_a(L(A_{fin})) \subseteq T_a(L(A_{inf}))$:

- The tree automaton A'_{fin} recognizing $T_a(L(A_{fin}))$ can be constructed in $O(|T_a| \times |A_{fin}|)$ time. Similarly, the tree automaton A'_{inf} recognizing $T_a(L(A_{inf}))$ can be constructed in $O(|T_a| \times |A_{inf}|)$ time. Note that A'_{fin} and A'_{inf} are unambiguous because A_{inf} and A_{fin} are unambiguous and T_a is deterministic.

- To decide whether $L(A'_{fin}) \subseteq L(A'_{inf})$ holds, it takes $O((|A'_{fin}| + |A'_{inf}|)^c)$ time where c is a constant because inclusion for unambiguous TAs can be decided in polynomial time [9].

The bottleneck of the time complexity of our algorithm is Step 6. The total time complexity of our algorithm is in single exponential time. Thus, ∞-SSIA is decidable in exponential time.

5.4 Algorithm for other query classes

We next discuss the decidability of ∞-SSIA when queries are given by another class, LDBT and LDTT^R. Both for query classes, we just revise steps 1 and 3 of the decision algorithm given in section 5.1.2.

5.4.1 Linear deterministic bottom-up tree transducer

We show that ∞-SSIA is decidable for linear deterministic bottom-up tree transducers (LDBT), by giving the revision from the algorithm for LDTTs in section 5.1.2. The differences from the decision algorithm for LDTTs are only

- (step 1) the construction of A_a and A_s from authorized/unauthorized queries T_a and T_s and
- (step 3) the computation of $\mathcal{Q}_a^D$ and $\mathcal{Q}_s^D$ which mean deleted states on T_a and T_s, respectively.

The rest of the algorithm works the same.

Step 1. Construct the following TAs A from a given linear deterministic bottom-up tree transducer $T = (Q, \Sigma, \Delta_1, Q^f, \delta)$: $A = (Q \times \delta \cup Q^d, \Sigma, Q_f, \delta')$ where $Q^d = \{q^d \mid q \in Q\}$ and δ' is the smallest set such that

- for each $r = \sigma(q_1(x_1), q_2(x_2)) \to q(u) \in \delta$ where $q_1, q_2, q \in Q$ and $\sigma \in \Sigma$,
  - $q^d \to \sigma(q_1^d, q_2^d) \in \delta'$, and

– $(q, r) \to \sigma(\tilde{q}_1, \tilde{q}_2) \in \delta'$, where for $i \in 1, 2$, if u contains x_i then $\tilde{q}_i = (q_i, r_i)$ such that q_i appears in the right-hand side of $r_i \in \delta$; otherwise $\tilde{q}_i = q_i^d$.

- for $r = \# \to q(u) \in \delta$, $\# \to (q, r) \in \delta'$ and $\# \to q^d \in \delta'$.

Note that A is unambiguous but neither top-down nor bottom-up deterministic in general; The size of A is $O(|T|^2)$ (precisely, $O(|Q| \times |\delta|) \cup Q^d$). For $t \in L(A)$ and $v \in V(t)$,

- if $r(A, t)(v) = q^d \in Q^d$, the subtree of t rooted by v is deleted by T;
- if $r(A, t)(v) = (q, r) \in Q \times \delta$ and $r = \sigma(q_1(x_1), q_2(x_2)) \to q(x_1)$ (an erasing rule), the node v is erased by T.

Hence, we revise step 3 as follows:

Step 3. Let Q_I be the set of states of A_I. Compute two subsets $\mathcal{Q}_a^D$ and $\mathcal{Q}_s^D$ of Q_I as follows:

$$\begin{aligned}
\mathcal{Q}_a^D &= \{(q_a, q_s, q) \in Q_I \mid q_a \in Q_a^d\} \\
&\quad \cup \{(\tilde{q}_a, q_s, q) \in Q_I \mid (\tilde{q}_a, \sigma) \in D_a, (q, \sigma) \in O\}, \\
\mathcal{Q}_s^D &= \{(q_a, q_s, q) \in Q_I \mid q_s \in Q_s^d\} \\
&\quad \cup \{(q_a, \tilde{q}_s, q) \in Q_I \mid (\tilde{q}_s, \sigma) \in D_s, (q, \sigma) \in O\},
\end{aligned}$$

where

$$\begin{aligned}
D_a &= \{((q_a, r), \sigma) \mid \exists i \in \{1, 2\}. r = \sigma(q_1(x_1), q_2(x_2)) \to q'(x_i) \in \delta_a\}, \\
D_s &= \{((q_s, r), \sigma) \mid \exists i \in \{1, 2\}. r = \sigma(q_1(x_1), q_2(x_2)) \to q'(x_i) \in \delta_s\}, \\
O &= \{(q, \sigma) \mid q \to \sigma(q_1, q_2) \in \delta\}.
\end{aligned}$$

It is known that any linear bottom-up tree transducer preserves regularity [2]. As in the same way of section 5.2, we can show that the revised algorithm for LDBTs works correctly. Since the sizes of A_a and A_s are still polynomial in $|T_a|$ and $|T_s|$, respectively, the time complexity of the algorithm is in single exponential time.

5.4.2 Linear deterministic top-down tree transducer with regular look-ahead

We next show that ∞-SSIA is decidable for linear deterministic top-down tree transducers with regular look-ahead (LDTT^{R}s). In a similar way to the case of LDBTs, we just give the revision of the algorithm in section 5.2.

Step 1. Construct the following TAs A from a given linear deterministic top-down tree transducer with regular look-ahead $T = (Q, \Sigma, \Delta, Q_i, \delta)$:

Step 1.1 Let $\mathcal{A}$ be the set of all the TAs occurring as ranges of variables in the rules of T. Let $A_{r,i}$ denotes the TA of ranges of variable x_i in the rule r. For each $A_{r,i} \in \mathcal{A}$, construct a bottom-up deterministic [2] and complete TA $A'_{r,i}$ equivalent to $A_{r,i}$. Let $A'_{r,i} = (Q'_{r,i}, \Sigma, Q'_{F,r,i}, \delta_{r,i})$, and let $\mathcal{A}' = \{A'_{r,i} \mid A_{r,i} \in \mathcal{A}\}$.

Step 1.2 Construct a product TA A^{rl} of all the TAs $A'_{r,i}$. Let $A^{rl} = (Q^{rl}, \Sigma, Q^{rl}_F, \delta^{rl})$. For $\bar{q} \in Q^{rl}$, let $Y_{r,i}(\bar{q})$ be the state of $A'_{r,i}$ occurring in $\bar{q}$.

Step 1.3 Construct $A = (Q \times \delta \cup \{q_d\} \times Q^{rl}, \Sigma, Q_f \times \delta \times Q^{rl}, \delta')$ where δ' is the smallest set such that

- for each $r = q(\sigma(x_1 : A_1, x_2 : A_2)) \to C[q_1(x_1), q_2(x_2)] \in \delta$ where $q_1, q_2, q \in Q$ and $\sigma \in \Sigma$,

$$((q, r), \bar{q}) \to \sigma((\tilde{q}_1, \bar{q}_1), (\tilde{q}_2, \bar{q}_2)) \in \delta'$$

  where $\bar{q} \to \sigma(\bar{q}_1, \bar{q}_2) \in \delta^{rl}$, for $i \in \{1, 2\}, Y_{r,i}(\bar{q}_i) \in Q'_{F,r,i}$, and
  - if RHS(r) contains x_i and $r_i \in \delta$, then $\tilde{q}_i = (q_i, r_i)$, and
  - if RHS(r) does not contain x_i, then $\tilde{q}_i = q_d$.
- for each $r = q(\#) \to u \in \delta$, $((q, r), \bar{q}) \to \# \in \delta'$ where $\bar{q} \to \# \in \delta^{rl}$.
- for each $\bar{q} \to \sigma(\bar{q}_1, \bar{q}_2) \in \delta^{rl}, (q_d, \bar{q}) \to \sigma((q_d, \bar{q}_1), (q_d, \bar{q}_2)) \in \delta'$

Step 3. Let Q_I be the set of states of A_I. Compute two subsets $\mathcal{Q}^D_a$ and $\mathcal{Q}^D_s$ of Q_I as follows:

$$\begin{aligned} \mathcal{Q}^D_a &= \{((q_a, \bar{q}_a), (\tilde{q}_s, \bar{q}_s), q) \in Q_I \mid q_a \in Q^d_a\} \\ &\quad \cup \{((\tilde{q}_a, \bar{q}_a), (\tilde{q}_s, \bar{q}_s), q) \in Q_I \mid ((\tilde{q}_a, \bar{q}_a), \sigma) \in D_a, (q, \sigma) \in O\}, \end{aligned}$$

$$
\begin{aligned}
\mathcal{Q}_s^D &= \{((\tilde{q}_a, \bar{q}_a), (q_s, \bar{q}_s), q) \in Q_I \mid q_s \in Q_s^d\} \\
&\quad \cup \{((\tilde{q}_a, \bar{q}_a), (\tilde{q}_s, \bar{q}_s), q) \in Q_I \mid ((\tilde{q}_s, \bar{q}_s), \sigma) \in D_s, (q, \sigma) \in O\},
\end{aligned}
$$

where

$$
\begin{aligned}
D_a &= \{(((q_a, r), \bar{q}_a), \sigma) \mid \exists i \in \{1, 2\}. r = q_a(\sigma(x_1 : A_1, x_2 : A_2)) \to q'(x_i) \in \delta_a\}, \\
D_s &= \{(((q_s, r), \bar{q}_s), \sigma) \mid \exists i \in \{1, 2\}. r = q_s(\sigma(x_1 : A_1, x_2 : A_2)) \to q'(x_i) \in \delta_s\}, \\
O &= \{(q, \sigma) \mid q \to \sigma(q_1, q_2) \in \delta\}.
\end{aligned}
$$

It is also known that linear top-down tree transducers with regular look-ahead (LDTT^R) preserve regularity [3]. As in the same way of section 5.1.2, we can show that the revised algorithm for LDTT^{R}s works correctly. The sizes of A_a and A_s are exponential in $|T_a|$ and $|T_s|$, respectively, because $|A^{rl}|$ is exponential in $|\mathcal{A}|$ and $|A|$ is proportional to product of $|A^{rl}|$, $|Q|$, and $|\delta|$. Thus, the time complexity of the algorithm is in double exponential time.

5.5 Discussion for multiple authorized queries

We have discussed ∞-SSIA for a single authorized query. Generalizing the results to the case that there are multiple authorized queries is important because using multiple authorized queries attackers can narrow down the candidates of secret information to less than using them independently in general. However, we conjecture that ∞-SSIA for more than one authorized queries is much complicated and might be undecidable at worst. In fact, the approach in section 5.1.1 does not work well. Now we assume that LDTTs T_{a1} and T_{a2} are given as authorized queries, and T_s and A_G are given as an unauthorized query and a schema. Then, the following procedure which is almost the same as section 5.1.2 was likely to work:

Step 1. Construct A_{a1}, A_{a2}, and A_s from T_{a1}, T_{a2}, and T_s, respectively.

Step 2. Compute the product tree automaton of A_{a1}, A_{a2}, A_s, and A_G.

Step 3. Compute $\mathcal{Q}_{a1}^D$, $\mathcal{Q}_{a2}^D$, and $\mathcal{Q}_s^D$, respectively.

Step 4. Compute $Q' = \{q_I \in Q_I \mid q_I$ is recursive through a proper path such that all the states in the path are in "$\mathcal{Q}^D_{a1} \cap \mathcal{Q}^D_{a2}$" and some state in the path is not in $\mathcal{Q}^D_s$.$\}$.

Steps 5–6. These steps are the same as section 5.1.2.

Step 7. Let $(T_{a1}, T_{a2})(L(A_{fin})) = \{(T_{a1}(t), T_{a2}(t)) \mid t \in L(A_{fin})\}$ and $(T_{a1}, T_{a2})(L(A_{inf})) = \{(T_{a1}(t), T_{a2}(t)) \mid t \in L(A_{fin})\}$. Then, decide whether $(T_{a1}, T_{a2})(L(A_{fin})) \subseteq (T_{a1}, T_{a2})(L(A_{inf}))$.

Unfortunately, the above procedure does not work at step 7 because we do not know whether $(T_{a1}, T_{a2})(L(A_{fin})) \subseteq (T_{a1}, T_{a2})(L(A_{inf}))$ is decidable. At least, $(T_{a1}, T_{a2})(L(A_{fin}))$ cannot be captured by any regular tree language in general.

Similarly, even for a single authorized query, when the authorized query T_a is given by a *non-linear* top-down tree transducer, which allows to copy subtrees twice or more, $T_a(L)$ is not guaranteed to be regular for a regular tree language L. Thus, our algorithm cannot be extended directly to such cases.

6. Conclusion

In this paper, we have discussed solvability of the problem of deciding whether every database instance conforming to a given XML schema is safe against inference attack for given authorized and unauthorized queries in the sense of k-secrecy (k-SSIA). We assume an XML schema is given by a tree automaton and a query is given by a linear deterministic tree transducer. We have shown that k-SSIA schema is undecidable for any natural number $k > 1$ by reduction from PCP even when a query is given by a deterministic rational transducer on words (DRTW), as well as when queries are given by LDTT, LDBT or LDTT^R. For a positive result, we have shown that ∞-SSIA is decidable for LDTT, LDBT and LDTT^R.

There are several goals left as future work. Firstly, we should improve our algorithm to work as well for multiple authorized queries since this paper focused on only ∞-SSIA for a single authorized query. This extension is important because using multiple authorized queries attackers can narrow down the candidates of secret information to less than using them independently in general. Second, we would like to extend our initial ideas for handling the k-secrecy preserving through updates. For some schema, k-secrecy is too strong to satisfy. On the other hand, a database instance is frequently updated in the real world. Consider the property that for any instance t which is k-secret, every instance obtained from t through updates is also k-secret. We call this property k-secrecy preserving property through updates, abbreviated as k-SP. If a given schema, queries and update operations satisfy k-SP, then it suffices to guarantee that the initial instance is k-secret in order to keep every instance through updates k-secret.

Formally, k-SP is defined as follows. Given an XML schema A_G, an authorized query T_a, an unauthorized query T_s and an update operation T_u, decide whether the next condition holds: For each $t \in L(A_G)$ which is k-secret and for each updated instance $t' \in T_u(t)$, t' is also k-secret. By definition, if there is a k-secret instance $t \in L(A_G)$ such that $T_u(t) = L(A_G)$, then k-SP and k-SSIA become equivalent since by the assumption $T_u(t) = L(A_G)$, k-SSIA requires that every instance $t' \in L(A_G)$ is k-secret. From this observation, k-SP makes sense when we appropriately restrict the power of update operations. Investigating the decidability of k-SP in an appropriate model of update operations is interesting future work.

Acknowledgements

I would like to express the deepest thankfulness to my supervisor, Professor Hiroyuki Seki, who encouraged, supported me during my Master's Program. He continually and convincingly conveyed me taking place in challenging in regard to research. Without his guidance and frequently sustain this research would not have been possible.

I also thanks to my co-supervisor Professor Minoru Ito, who always made more than several suggestions to enhance this research. Furthermore, his suggestions always very much to the point of my presentation.

My sincere thankful are extended to Associate Professor Yuichi Kaji, my co-supervisor for his guidances and suggestions, for all his advices, without whose knowledges and assistances this study would not have been successful.

In particular Assistant Professor Kenji Hashimoto, my co-supervisor, who supported me from the beginning, provided an insightful view of my work, his enlightening conveyed strength that keeps me standing and for the hope that assures this research would be possible and meaningful.

Finally, my big thanks go to the APAL laboratory members, friends, and my family, I appreciate their encouragement, without them I would not have strength to push my self this hard.

References

[1] G. J. Bex, S. Maneth, and F. Neven. A formal model for an expressive fragment of XSLT. *Information Systems*, vol. 27, issue 1, pp.21–39, 2002.

[2] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. http://tata.gforge.inria.fr/, 2008.

[3] J. Engelfriet, Top-down tree transducers with regular look-ahead, *Theory of Computing Systems*, vol.10, no.1, pp.289-303, 1977.

[4] K. Hashimoto, K. Sakano, F. Takasuka, Y. Ishihara, and T. Fujiwara. Verification of the security against inference attacks on XML databases. *IEICE Transactions on Information and systems*, vol.E92-D, no.5, pp.1022-1032, 2009.

[5] H. Hosoya. Foundations of XML Processing - The Tree-automata Approach. *Cambridge University Press*, 2011.

[6] S. Kepser. A simple proof for the Turing-completeness of XSLT and XQuery. *Proc. of Extreme Markup Languages*, 2004.

[7] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. ℓ-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data, vol.1, issue 1, 2007, earlier version is in Proc. of the 22nd ICDE, 2006.*

[8] T. Schwentick. Automata for XML - A survey. *Journal of computer and system sciences*, vol.73, issue 3, pp.289–315, 2007.

[9] H. Seidl. Deciding equivalence of finite tree automata. *SIAM J. Comput.*, vol.19, issue 3, pp.424–437, 1990.

Publication list

[1] Chittaphone Phonharath, Kenji Hashimoto and Hiroyuki Seki. Static analysis for k-secrecy against Inference Attacks, Joint Workshop on Software Science and Engineering, IEICE Technical Report, SS2011-4, June 2011.