Toward Security Verification against Inference Attacks on Data Trees

Ryo Iwase Yasunori Ishihara Toru Fujiwara
Graduate School of Information Science and Technology, Osaka University
{r-iwase,ishihara,fujiwara}@ist.osaka-u.ac.jp

This paper describes our ongoing work on security verification against inference attacks on data trees. We focus on infinite secrecy against inference attacks, which means that attackers cannot narrow down the candidates for the value of the sensitive information to finite by available information to the attackers. Our purpose is to propose a model under which infinite secrecy is decidable. To be specific, we first propose tree transducers which are expressive enough to represent practical queries. Then, in order to represent attackers' knowledge, we propose data tree types such that type inference and inverse type inference on those tree transducers are possible with respect to data tree types, and infiniteness of data tree types is decidable.

1 Introduction

Nowadays, many organizations utilize and store information in databases. These databases may contain highly confidential information. One of the important problems on achieving database security for these database systems is to ensure the security against inference attacks. Inference attacks mean that users infer the information which they cannot access directly by using the authorized queries and the result of them. In order to ensure the security of databases, it is important to figure out the possibility in advance that the sensitive information can be leaked by inference attacks.

Example 1: We show an example of inference attacks on XML databases. We consider an XML document I (see Fig. 1) representing the correspondence between student name, origin, and the amount of the scholarship, and valid against the following schema:

$$\begin{aligned} \mathsf{faculty} &\rightarrow (\mathsf{student}\{@\mathsf{scholarship}\})^* \\ \mathsf{student}\{@\mathsf{scholarship}\} &\rightarrow \mathsf{name}\{@\mathsf{str}\}, \mathsf{origin}\{@\mathsf{str}\} \end{aligned}$$

That is, the faculty element has zero or more student elements as its children. Each student element has a scholarship as a data value which is a nonnegative integer without upper limit, and has a name element and an origin element as its children. Each of name and origin element has a str as a string data value.

Let T_1 be an authorized query extracting the name and the origin of each student. Let T_2 be an authorized query extracting the origin of the student who receives the most amount of the scholarship, and T_3 be an authorized query extracting the origin and scholarship of the student who receives the second most amount of the scholarship. Moreover, we set the sensitive information to the amount of the scholarship which a student of a given name receives, and let T_A, T_B, T_C, and T_D be the unauthorized queries extracting the amount of the scholarship of the student of the name A, B, C, and D, respectively (i.e., extracting the sensitive information). Now, we assume that the results of T_1, T_2, and T_3 are the trees shown in Figs. 2, 3, and 4, respectively.

S. Maneth (Ed.): Workshop on Trends in Tree Automata and Tree Transducers (TTATT 2013).
EPTCS 134, 2013, pp. 49–59, doi:10.4204/EPTCS.134.6

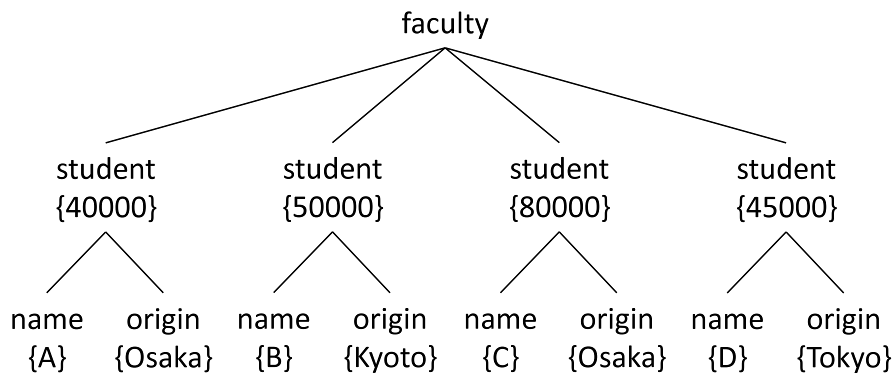

Figure 1: An XML Document I.

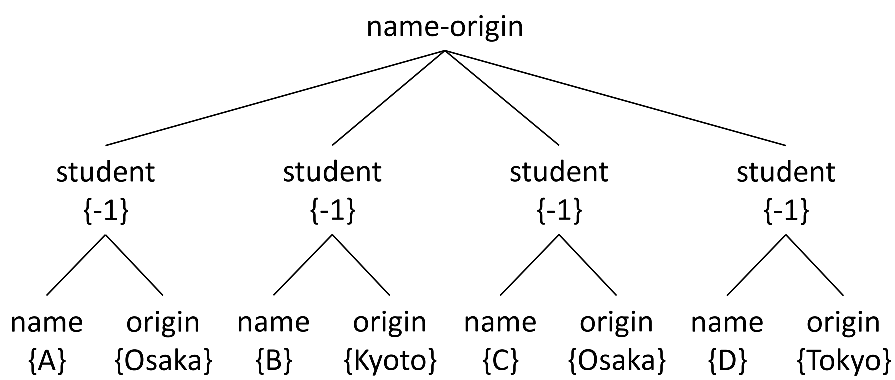

Figure 2: The result of T_1.

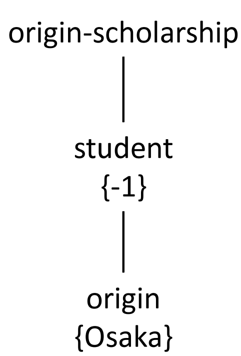

Figure 3: The result of T_2.

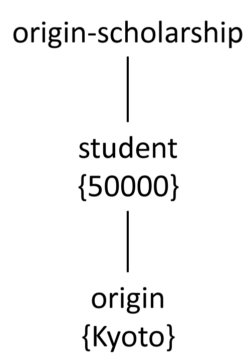

Figure 4: The result of T_3.

Then, we know that from the result of T_1, the student whose origin is Kyoto is only B, and from the result of T_3, the student whose origin is Kyoto receives 50,000 yen as a scholarship. Therefore, we find that B receives 50,000 yen. That is, the result $T_{\mathrm{B}}(I)$ of T_{B} is identified by inference attacks. Moreover, from the result of T_2, we know that a student whose origin is Osaka receives the most amount of the scholarship. Here, considering that the origin of the student who receives the most amount of the scholarship is Osaka, we find that D receives less than 50,000 yen. Therefore, we can narrow down the number of the candidates of $T_{\mathrm{D}}(I)$ to 50000. However, we cannot identify the person who receives the most amount of the scholarship because we know that there are two students whose origins are Osaka. Also, we do not know the most amount of the scholarship. Therefore, we cannot narrow down the number of the candidates of each of $T_{\mathrm{A}}(I)$ and $T_{\mathrm{C}}(I)$ to a finite number. □

The protection of sensitive information in XML databases has been studied in terms of access control. In [5], an access control model to protect information is proposed. In the model, information to be protected is represented by a path expression, and for each information, the authorizations of users are defined clearly. Ref. [4] discusses access control in the presence of insertions and updates of a database.

In our previous work [7], we formulated the security against inference attacks on XML databases and proposed a verification method of the security called infinite secrecy. The whole picture of our verification is shown in Fig. 5. The notion of infinite secrecy is as follows: Suppose that the following information is available to a user:

- The authorized queries T_1, T_2, …, T_n,
- The results $T_1(D)$, $T_2(D)$, …, $T_n(D)$ of the authorized queries on an XML document D,
- The schema A_G of D, and
- The query T_S to retrieve the sensitive information.

Then, the candidate set C of the values of the sensitive information inferred by the user is

$$C \quad = \quad \{T_S(D') \mid D' \in TL(A_G), T_1(D') = T_1(D), T_2(D') = T_2(D), \ldots, T_n(D') = T_n(D)\},$$

where $TL(A_G)$ denotes the set of trees valid against A_G. If $|C|$ is infinite, then we say that *D is infinitely secret with respect to T_S*. In the example above, I is infinitely secret with respect to T_{A} and T_{C}, and is not with respect to T_{B} and T_{D}. The proposed verification method can only handle queries represented by relabeling or deleting the specified nodes in an XML document. Since the set of labels is finite in the formulation, the verification method of the security with queries involving infinite data value comparisons has not been studied.

We consider the verification of infinite secrecy against inference attacks on data trees. The verification method consists of the following three steps:

1. Construct a candidate set of XML document D from the authorized queries T_1, T_2, …, T_n, their results $T_1(D)$, $T_2(D)$, …, $T_n(D)$, and the schema A_G.
2. Construct a candidate set C of the value of $T_S(D)$ from the candidate set of D and T_S.
3. Decide whether the number of the elements of C is infinite.

In the verification, *type inference* and *inverse type inference* on tree transducers are used at steps 2 and 1, respectively. *Type inference* is to construct the candidate set Doc'_{out} of output trees of a tree transducer from the tree transducer and the candidate set Doc_{in} of input trees. *Inverse type inference* is to construct the candidate set Doc'_{in} of input trees of a tree transducer from the tree transducer and the candidate set Doc_{out} of output trees. To verify the security according to these steps, the models must satisfy three

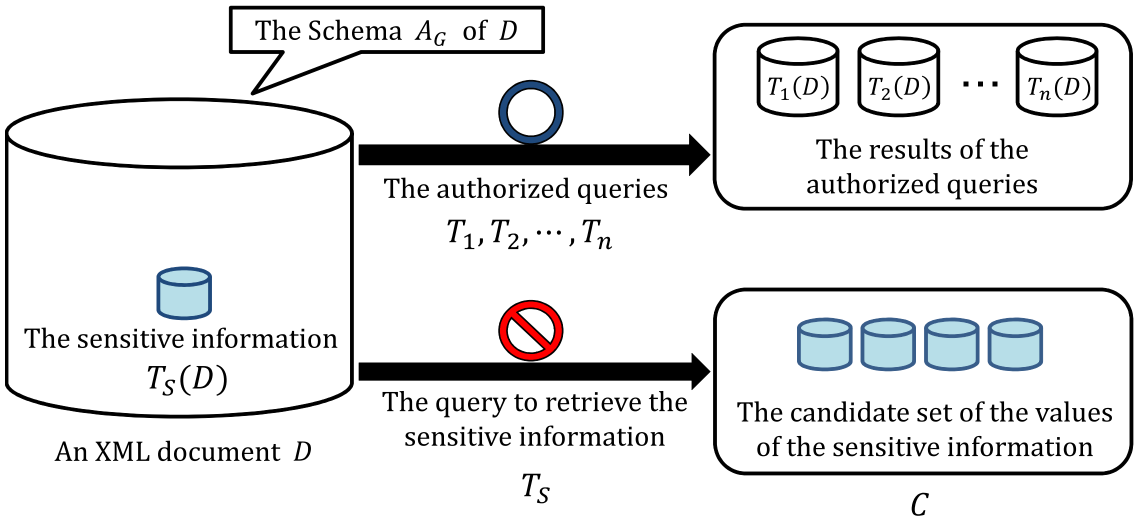

Figure 5: The whole picture of our verification.

requirements. First, tree transducers must be able to represent practically significant queries. Second, type inference and inverse type inference on tree transducers must be possible. Third, it must be decidable whether the number of the candidates of the sensitive information is infinite.

In this paper, we discuss a verification method of the security against inference attacks on data trees. We propose models satisfying the aforementioned three requirements. First, we propose tree transducers on data trees which are expressive enough to represent practical queries. Operations corresponding to projection, selection, and natural join in the relational algebra are allowed in queries by using those tree transducers. Then, in order to represent attackers' knowledge, we propose *data tree types* such that type inference and inverse type inference on those tree transducers are possible with respect to data tree types, and infiniteness of data tree types is decidable. A data tree type consists of a non-deterministic finite tree automaton, a mapping from the set of the pairs of the states of the tree automaton and the labels of the nodes to the set of variables, and a finite set of conditional expressions between variables or between a variable and a constant. Until now, we have provided inverse type inference on several tree transducers and type inference on data-rewriting transducers, and have provided an algorithm to decide infiniteness of data tree types.

2 Preliminaries

We use *data trees* [3][2] as a model of XML documents. We define data trees as follows.

Definition 1: Let Σ be a finite set of labels including special symbols $\sharp$ and \$, and D be a countable infinite set of data values on which a total order $<$ is defined. We assume that D is a set of integers or rational numbers. A data tree t is a 3-tuple $\langle T, l, \rho \rangle$, where

- T is a set of nodes, which is a prefix-closed finite subset of $\mathbb{N}^*$ such that for all $j < i$ and $v \in \mathbb{N}^*$, if $v \cdot i \in T$ then $v \cdot j \in T$,
- l is a mapping from T to Σ, and
- ρ is a mapping from T to D.

That is, each node has just one label and one data value. In T, ε is called the root node, and for any two nodes v, $v \cdot i \in T$, v is called the parent of $v \cdot i$, and $v \cdot i$ is called the i-th child of v. □

We use *non-deterministic finite tree automata* (NFTAs) to represent XML schemas.

Definition 2: An NFTA A is a 4-tuple (Q, Σ, q_0, R), where

- Q is a finite set of states,
- Σ is a finite set of labels,
- $q_0 \in Q$ is the initial state, and
- R is a set of transition rules in the form of (q, a, e), where $q \in Q$, $a \in \Sigma$, and e is a non-deterministic finite automaton over Q.

A run of an NFTA assigns states to nodes of an input tree according to the transition rules. Formally, we define a run r^t_A of $A = (Q, \Sigma, q_0, R)$ against $t = \langle T, l, \rho \rangle$ as a mapping from T to Q with the following properties:

- $r^t_A(\varepsilon) = q_0$.
- For each node $v \in T$ with n children, there exists a transition rule $(q, a, e) \in R$ such that $r^t_A(v) = q$, $l(v) = a$, and $r^t_A(v \cdot 1) r^t_A(v \cdot 2) \cdots r^t_A(v \cdot n)$ is in the string language represented by e.

t is accepted by A if there exists a run of A against t. Let $TL(A)$ denote the set of data trees accepted by A. □

3 Proposed Models

3.1 Queries

We use deterministic tree transducers to represent queries. We define seven types of tree transducers for our verification.

- A *deterministic top-down relabeling tree transducer* [7] relabels the current node according to the state of the node and assigns states to the children of the node, traversing in a top-down manner.
- A *deterministic bottom-up relabeling tree transducer* [7] relabels the current node according to the label of the node and the states of the children, traversing in a bottom-up manner.
- A *deterministic deleting tree transducer* [7] deletes nodes labeled by $\sharp$ and subtrees rooted by \$.
- A *deterministic data-rewriting tree transducer* rewrites the data value of all the nodes which have a specified label a to a specified value d. The operation by a data-rewriting tree transducer corresponds to projection in the relational algebra. Formally, given label a and data value d, a deterministic data-rewriting tree transducer transforms $t = \langle T, l, \rho \rangle$ into $t' = \langle T, l, \rho' \rangle$, where ρ' is a mapping defined as follows:

$$\rho'(v) = \begin{cases} d & \text{if } l(v) = a, \\ \rho(v) & \text{otherwise.} \end{cases}$$

- A *deterministic data-relabeling tree transducer* relabels all the nodes which have a specified label a and data value d to a specified label a'. The operation by a data-relabeling tree transducer corresponds to selection in the relational algebra. Formally, given labels a, a', and data value d, a deterministic data-relabeling tree transducer transforms $t = \langle T, l, \rho \rangle$ into $t' = \langle T, l', \rho \rangle$, where l' is a mapping defined as follows:

$$l'(v) = \begin{cases} a' & \text{if } l(v) = a \text{ and } \rho(v) = d, \\ l(v) & \text{otherwise.} \end{cases}$$

- A *deterministic min-data-relabeling tree transducer* relabels all the nodes which have the minimum value of the nodes labeled by a specified label a to a specified label a'. A deterministic min-data-relabeling tree transducer is used for representing an operation like natural join in the relational algebra. Formally, given labels a and a', a deterministic min-data-relabeling tree transducer transforms $t = \langle T, l, \rho \rangle$ into $t' = \langle T, l', \rho \rangle$, where l' is a mapping defined as follows:

$$l'(v) = \begin{cases} a' & \text{if } l(v) = a \text{ and } \rho(v) = \min\{\rho(v') \mid l(v') = a\}, \\ l(v) & \text{otherwise.} \end{cases}$$

- A *deterministic max-data-relabeling tree transducer* is a counterpart of a deterministic min-data-relabeling tree transducer. Formally, given labels a and a', a deterministic max-data-relabeling tree transducer transforms $t = \langle T, l, \rho \rangle$ into $t' = \langle T, l', \rho \rangle$, where l' is a mapping defined as follows:

$$l'(v) = \begin{cases} a' & \text{if } l(v) = a \text{ and } \rho(v) = \max\{\rho(v') \mid l(v') = a\}, \\ l(v) & \text{otherwise.} \end{cases}$$

The procedures of natural join by min/max-data-relabeling tree transducers are as follows. We consider an XML document which have the information of two relations A and B. First, choose a pair of nodes p and q, where p and q correspond to a tuple of A and B, respectively. Second, relabel their nodes to a new label a' by a bottom-up/top-down relabeling tree transducer. Third, relabel all the nodes labeled by a' to a new label b' by a min/max-data-relabeling tree transducer. If the values of p and q are the same, then both p and q are labeled by b', and we can join these two nodes. Otherwise, we need to choose another pair because we cannot join these two nodes.

A *query* is a composition of these tree transducers satisfying the following restrictions. First, every query must be a composition of zero or more tree transducers except deleting tree transducers followed by a deleting tree transducer. Second, no constituent tree transducers of the unauthorized query T_{S} relabels a node to $\sharp$. These restrictions are necessary for type inference to be possible. Without these restrictions, candidates for the value of the sensitive information cannot be represented by an NFTA even if we do not consider data values. For example, consider an NFTA $A = (Q, \Sigma, q_0, R)$, where

- $Q = \{q_0, q_a, q_b\}$, and
- $R = \{(q_0, r, q_a q_b q_a), (q_a, a, \varepsilon), (q_b, b, q_a q_b q_a | \varepsilon)\}$.

For trees accepted by A, consider the trees obtained by relabeling the nodes labeled by b to $\sharp$ and then deleting the nodes labeled by $\sharp$. The resulting trees have the root node labeled by r, and the strings obtained by the concatenation of labels of its children is the form of $a^n b a^n$, which cannot be represented by an NFTA.

Queries appeared in Example 1 can be represented by the proposed tree transducers. For example, T_{A} can be represented as follows.

1. Relabel the name nodes which have the value "A" to name′ by a data-relabeling tree transducer.
2. Relabel the student nodes which have a name node as a child to \$ by a bottom-up relabeling tree transducer.
3. Relabel the name′ nodes and origin nodes to $\sharp$ by a top-down/bottom-up relabeling tree tranducer.
4. Delete nodes by a deleting tree transducer.

The others can also be represented similarly.

3.2 Data tree types

We introduce *data tree types* to represent sets of data trees, which model user's knowledge during inference attacks. A data tree type is defined as a finite union of *atomic data tree types*. Similarly to existing research on incomplete information like [1], we use variables and conditional expressions on them to represent undetermined data values. However, there are several novelties in our model. First, each variable is associated with a pair of a state of an NFTA and a label, rather than a node of a fixed tree. Hence, one atomic data tree type can handle infinitely many variations of tree shapes (up to the expressive power of NFTAs). This is useful for type inference involving relabeling and deleting tree transducers. Next, our model uses two kinds of variables. *S-variables* are ordinary ones, and all data values of nodes which have the same s-variable must be the same. *M-variables* are novel ones, and all data values of nodes which have the same m-variable are not necessarily the same, but satisfy conditional expressions on the m-variable. M-variables are useful for inverse type inference involving data-rewriting tree transducers. For example, consider inverse type inference on data-rewriting tree transducers which rewrites the data value of all the nodes labeled by a to value d. We cannot know data values of the nodes labeled by a in input trees since these values have already been rewritten to d. By the following definition of θ, these nodes may have the same variable if a run of an NFTA assigns the same state to these nodes. On the other hand, these nodes may have different values. Since m-variable can have more than one value, we can represent values of these nodes by an m-variable.

In what follows, s-variables and m-variables are written as $\ddot{x}$ and $\tilde{x}$, respectively. We also write simply like x when we do not differentiate two kinds of variables.

Definition 3: An *atomic data tree type Doc* is a 3-tuple $\langle A, \theta, E\rangle$, where

- $A = (Q, \Sigma, q_0, R)$ is an NFTA;
- θ is a mapping from $Q \times \Sigma$ to the set of variables; and
- E is a finite set of conditional expressions in the following form:
  - x op $\ddot{y}$, where op $\in \{<, >, \subseteq\}$,
  - d op y, where op $\in \{\in, \notin\}$ and d is a constant, and
  - $\tilde{x} \subseteq \tilde{y}$. □

The semantics of an atomic data tree type is defined as follows. Let σ be a mapping from a set of variables to the power set of values satisfying the following conditions:

- for all s-variables $\ddot{x}$, $|\sigma(\ddot{x})| = 1$, and
- for all m-variables $\tilde{x}$, $|\sigma(\tilde{x})| \geq 0$.

We extend the definition of σ toward conditional expressions as follows:

- $\sigma(x \text{ op } \ddot{y}) = \{a \text{ op } b \mid a \in \sigma(x), b \in \sigma(y)\}$, where op $\in \{<, >\}$,

- $\sigma(x \subseteq \ddot{y}) = \{\sigma(x) \subseteq \sigma(\ddot{y})\}$,
- $\sigma(d \text{ op } y) = \{d \text{ op } \sigma(y)\}$, where op $\in \{\in, \notin\}$ and d is a constant,
- $\sigma(\tilde{x} \subseteq \tilde{y}) = \{\sigma(\tilde{x}) \subseteq \sigma(\tilde{y})\}$, and
- $\sigma(E) = \bigcup_{x \text{ op } y \in E} \sigma(x \text{ op } y)$.

We assume that D is a set of integers or rational numbers. Therefore, each meaning of $<$ and $>$ is the same as that in sets of integers or rational numbers.

A data tree $t = \langle T, l, \rho \rangle$ *belongs to* the set of data trees represented by an atomic data tree type $Doc = \langle A, \theta, E \rangle$ if there exist a run r^t_A and a mapping σ such that for all $v \in T$, $\rho(v) \in \sigma(\theta(r^t_A(v), l(v)))$, and all the conditional expressions in $\sigma(E)$ hold. Let $TL(Doc)$ denote the set of data trees which belongs to Doc.

4 Decidability

In this section, we refer to the detail of type inference and inverse type inference, and provide an algorithm to decide infiniteness of data tree types.

4.1 Type inference and inverse type inference

As stated already, we have proved the correctness of inverse type inference on several tree transducers and type inference on data-rewriting transducers. In this section, we show the detail of inverse type inference on data-rewriting tree transducers.

For a data-rewritng tree transducer which rewrites the data value of all the nodes labeled by a to value d, the data tree type $Doc' = \langle A', \theta', E' \rangle$ of input trees is constructed from the data tree type $Doc = \langle A, \theta, E \rangle$ of output trees, where $A = (Q, \Sigma, q_0, R)$. The detail of $\langle A', \theta'.E' \rangle$ is as follows.

- $A' = A$.
- $\theta'(q,c) = \begin{cases} \tilde{x}' & \text{if } c = a, \\ \theta(q,c) & \text{otherwise.} \end{cases}$
  Here, $\tilde{x}'$ is a variable which do not appear in Doc.
- $E' = E \cup \{d \in \theta(q,a) \mid (q,a,e) \in R\}$.

Proof. First, let $t' = \langle T, l, \rho' \rangle \in TL(\langle A', \theta', E' \rangle)$ be an input tree. There exist a run $r^{t'}_{A'}$ and a mapping σ', and the following properties hold:

- for all $v \in T$, $\rho(v) \in \sigma'(\theta'(r^{t'}_{A'}(v), l(v)))$, and
- all the conditional expressions in $\sigma'(E')$ hold.

Then, by the definition of the tree transducer, the output tree $t = \langle T, l, \rho \rangle$ is accepted by A since there exists a run r^t_A such that $r^t_A = r^{t'}_{A'}$, and for all $v \in T$ satisfying $l(v) = a$, we have $\rho(v) = d$. Here, let $\sigma = \sigma'$. Since all conditional expressions in $\sigma'(E')$ hold, for all $v \in T$ satisfying $l(v) = a$, $\rho(v) \in \sigma(\theta(r^t_A(v), l(v)))$. Moreover, $E \subseteq E'$. Therefore, for r^t_A and σ, the following properties hold:

- for all $v \in T$, $\rho(v) \in \sigma(\theta(r^t_A(v), l(v)))$, and
- all the conditional expressions in $\sigma(E)$ hold.

Hence, $t \in TL(\langle A, \theta, E\rangle)$.

Inversely, let $t = \langle T, l, \rho\rangle \in TL(\langle A, \theta, E\rangle)$ be an output tree. Then, for all $v \in T$ satisfying $l(v) = a$, t must satisfy $\rho'(v) = d$. There exist a run r_A^t and a mapping σ, and the following properties hold:

- for all $v \in T$, $\rho(v) \in \sigma(\theta(r_A^t(v), l(v)))$, and
- all the conditional expressions in $\sigma(E)$ hold.

Then, by the definition of the tree transducer, the input tree $t' = \langle T, l, \rho'\rangle$ is accepted by A' since there exists a run $r_{A'}^{t'}$ such that $r_{A'}^{t'} = r_A^t$. Moreover, we define σ' as follows:

$$\sigma'(x) = \begin{cases} \{\rho(v) \mid l(v) = a\} & \text{if } x = \tilde{x}, \\ \sigma(x) & \text{otherwise.} \end{cases}$$

Then, for all $v \in T$, $\rho(v) \in \sigma'(\theta'(r_{A'}^{t'}(v), l(v)))$. Moreover, since for all $v \in T$ satisfying $l(v) = a$, $d \in \sigma(\theta(r_A^t(v), l(v)))$, all conditional expression in $\sigma'(E')$ hold. Hence, $t' \in TL(\langle A', \theta', E'\rangle)$. $\square$

4.2 Infiniteness of data tree types

The following algorithm is to decide infiniteness of an input data tree type $Doc = (A, \theta, E)$, where $A = (Q, \Sigma, q_0, R)$. Before discussing the detail of the algorithm, we declare in advance that we can assume that there is no conditional expression which cannot be satisfied in E for the following reason. If there existed some conditional expressions which cannot be satisfied in E, then by the definition of data tree types, $TL(Doc)$ would be an emptyset. However, this assumption is contradictory since $TL(Doc)$ must contain at least the sensitive information.

The detail of the algorithm for deciding infiniteness is as follows.

1. Separate the number line into zones by constants in E. If D is an integer set, then we consider only integers in each zone.

2. Find the assignment σ satisfying all conditional expressions in E as follows.

   2-1 For each s-variable in E, assign a zone non-deterministically and then break up zones according to the assignment.

   2-2 Assign a set of zones to each m-variable in E non-deterministically.

3. For each (q, a) such that $\sigma(\theta(q, a)) = \emptyset$, construct R_σ from R by rewriting each $(q, a, e) \in R$ to $(q, a, \emptyset)$, and check whether $TL(A_\sigma)$ is infinite where $A_\sigma = (Q, \Sigma, q_0, R_\sigma)$. If $TL(A_\sigma)$ is infinite, then output "Yes (i.e., $TL(Doc)$ is infinite)." Otherwise, if there exists a pair (q, a) such that $\sigma(\theta(q, a))$ is infinite and there is $t = \langle T, l, \rho\rangle \in TL(A_\sigma)$ such that $(q, a) = (r_{A_\sigma}^t(v), l(v))$ for some $v \in T$, then output "Yes." If there does not exist such pair (q, a), then output "No (i.e., $TL(Doc)$ is finite)."

We show an example of an assignment and a breakup of zones. Consider the set $E = \{1 \in \ddot{x}_1, 2 \notin \ddot{x}_2, 3 \in \tilde{x}_1, \tilde{x}_1 \subseteq \tilde{x}_2\}$. First, as shown in Fig. 6, the number line is broken up into seven zones. Next, as shown in Fig. 7, zone 2 and zone 7 are assigned to $\ddot{x}_1$ and $\ddot{x}_2$, respectively, and then zone 7 is broken up into three zones 7-1, 7-2, and 7-3 by the assignment. Finally, as shown in Fig. 8, zones 5 and 6 are assigned to $\tilde{x}_1$, and zones 1, 5, and 6 are assigned to $\tilde{x}_2$.

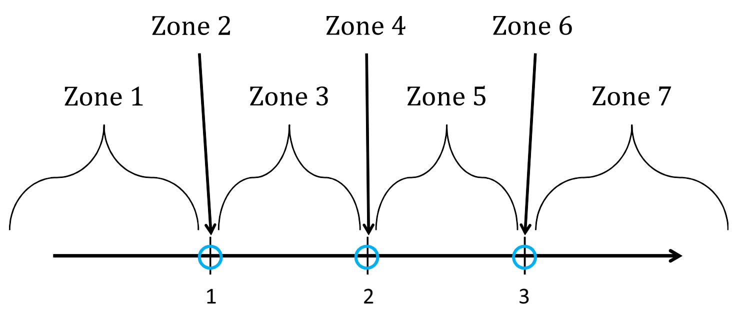

Figure 6: The breakup of the number line into zones.

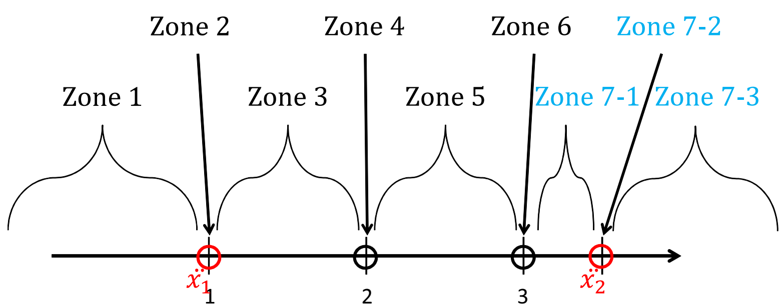

Figure 7: The assignment of zones to s-variable.

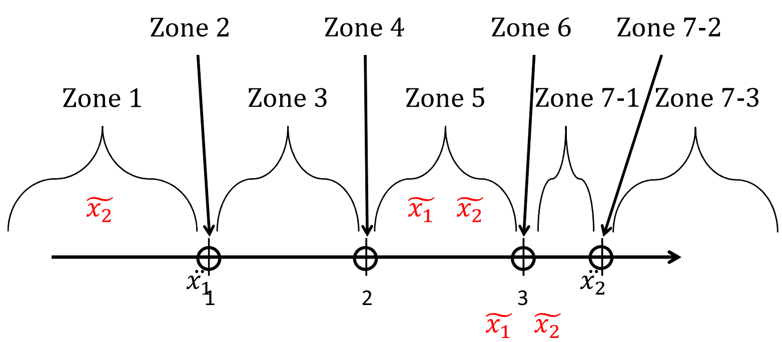

Figure 8: The assignment of zones to m-variable.

5 Ongoing and Future Work

This paper has discussed security verification against inference attacks on data trees. We have proposed tree transducers on data trees which can represent projection, selection, and natural join in the relational algebra. Moreover, we have proposed data tree types for representing the candidate set of the value of the sensitive information.

We are now trying to prove that type inference and inverse type inference are possible on queries with respect to data tree types. We have done inverse type inference on several tree transducers and type inference on data-rewriting transducers until now. One of our future work is to evaluate the complexity of our method. Another future work is to consider inference attacks using functional dependencies [6] on data trees.

References

[1] Serge Abiteboul, Luc Segoufin & Victor Vianu (2006): *Representing and querying XML with incomplete information*. *ACM Trans. Database Syst.* 31(1), pp. 208–254, doi:10.1145/1132863.1132869.

[2] Noga Alon, Tova Milo, Frank Neven, Dan Suciu & Victor Vianu (2001): *XML with data values: typechecking revisited*. In: *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '01, ACM, pp. 138–149, doi:10.1145/375551.375570.

[3] Mikoaj Bojańczyk, Anca Muscholl, Thomas Schwentick & Luc Segoufin (2009): *Two-variable logic on data trees and XML reasoning*. *J. ACM* 56(3), pp. 13:1–13:48, doi:10.1145/1516512.1516515.

[4] James Cheney (2013): *Static Enforceability of XPath-Based Access Control Policies*. In: *Proceedings of the 14th International Symposium on Database Programming Languages*. Available at http://arxiv.org/abs/1308.0502.

[5] Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi & Pierangela Samarati (2002): *A fine-grained access control system for XML documents*. *ACM Trans. Inf. Syst. Secur.* 5(2), pp. 169–202, doi:10.1145/505586.505590.

[6] Kenji Hashimoto, Hiroto Kawai, Yasunori Ishihara & Toru Fujiwara (2012): *Decidability of the Security against Inference Attacks using a Functional Dependency on XML Databases*. *IEICE Transactions on Information and Systems* E95-D(5), pp. 1365–1374, doi:10.1587/transinf.E95.D.1365.

[7] Kenji Hashimoto, Kimihide Sakano, Fumikazu Takasuka, Yasunori Ishihara & Toru Fujiwara (2009): *Verification of the Security against Inference attacks on XML Databases*. *IEICE Transactions on Information and Systems* E92-D(5), pp. 1022–1032, doi:10.1587/transinf.E92.D.1022.