Electronic Preprint for Journal of Information Processing Vol.25

Regular Paper

Towards Practical Typechecking for Macro Forest Transducers

Kazuhiro Abe[1,a)] Keisuke Nakano[1,b)]

Received: February 4, 2017, Accepted: August 4, 2017

Abstract: Macro tree transducers (MTTs) and macro forest transducers (MFTs) have been used as good models of tree-structured data transformations such as XML transformations. Typechecking of transformations in these models is performed to verify if any tree of an input type is always transformed into a tree of an output type, which is useful for validating XML transformations against given XML schemata. In typechecking problems for MTTs and MFTs, each "type" is usually given by a tree automaton. A naive implementation of a typechecking algorithm is very inefficient because its time complexity is beyond exponential to the number of states of tree automata, and a large number of equivalence checking operations over finite maps are required. For typechecking of MTTs, Frisch and Hosoya proposed an efficient and practical algorithm by using alternating tree automata as an internal representation of types and reducing the problem to satisfiability checking over first-order logic formulae. In this paper, we extend their typechecking method to apply it to MFTs that are more expressive than MTTs. Our implementation of the proposed method shows that it performs typechecking for relatively simple cases in a reasonable time.

Keywords: macro forest transducers, typechecking, alternating tree automata

1. Introduction

Macro tree transducers (MTTs) [1] have been studied as a model of transformations between trees. However, since trees in MTTs have a fixed number of children, MTTs cannot deal with JSON and XML, called *forest*, which have an arbitrary number of children. To solve this problem, macro forest transducers (MFTs) [2] have been proposed by extending MTTs so that they can be applied to actual XML transformations [3], [4].

One of the important properties of tree transducers such as MTTs and MFTs is the decidability of typechecking. Given a specific input and output as sets $\mathcal{L}_{\text{in}}$ and $\mathcal{L}_{\text{out}}$ of trees, typechecking of a transformation $\mathcal{T}$ induced by a tree transducer is performed to verify if any tree in $\mathcal{L}_{\text{in}}$ is always transformed by transformation $\mathcal{T}$ into a tree in $\mathcal{L}_{\text{out}}$. In general, sets of trees are specified as tree automata (TAs). Typechecking of tree transducers is applied to validate XML transformations against XML schemata [5].

Typechecking methods of tree transducers are roughly divided into the following two groups.

Backward typechecking First, we calculate the inverse image $\mathcal{T}^{-1}(\mathcal{L}_{\text{out}}^{\complement})$ of the complement of the output type $\mathcal{L}_{\text{out}}$ under the transformation $\mathcal{T}$, and then we check $\mathcal{T}^{-1}(\mathcal{L}_{\text{out}}^{\complement}) \cap \mathcal{L}_{\text{in}} = \emptyset$, which is equivalent to $\mathcal{T}(\mathcal{L}_{\text{in}}) \subseteq \mathcal{L}_{\text{out}}$.

Forward typechecking First, we calculate the image $\mathcal{T}(\mathcal{L}_{\text{in}})$ of the input type $\mathcal{L}_{\text{in}}$ under the transformation $\mathcal{T}$, and then we check directly $\mathcal{T}(\mathcal{L}_{\text{in}}) \subseteq \mathcal{L}_{\text{out}}$.

For MTTs and MFTs, backward typechecking has been intensively studied [1], [2]. Frisch and Hosoya [6] proposed an efficient backward typechecking algorithm for MTTs by using alternating tree automata (ATAs) [7] for inverse images instead of TAs and implemented it [8]. Since any MFT can be represented by a composition of two MTTs, any MFT can be typechecked by two-fold computation of the inverse images of the MTTs [2]. However, this *indirect* typechecking method for MFTs is inefficient due to the number of states of an ATA expressing the inverse image used in the middle of the typechecking process. As a direct typechecking method for MFTs, Perst and Seidl's naive backward typechecking [2] is known. Their method is based on the fact that MFTs extend MTTs just by adding a concatenation operator for forests in the same way as for lists. In the method, the inverse image of an MFT is given by a TA in which each state is given by binary relation, that is, a set of pairs, representing state transitions caused by forest concatenation. Their proposed typechecking method showed the worst-case complexity. However, there has been no proposal or implementation of a practical typechecking method for MFTs. On the other hand, Kobayashi et al. [9] proposed a faster forward typechecking method for linear higher-order multi-parameter tree transducers that are as expressive as compositions of MTTs and implemented it. In particular, since an MFT can be converted to 4-order linear higher-order multi-parameter tree transducers, it is possible to typecheck any MFT. However, their typechecking method is not specialized for MFTs, so there may be room for improvement. In this paper, we propose a new backward typechecking method for MFTs based on Perst and Seidl's idea to compute the inverse image while paying attention to output state transitions by concatenating forests. We give a different construction of a TA in which a state is a pair of output states while they use a set of pairs. Based on their

[1] Graduate School of Informatics and Engineering, the University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[a)] abe@ipl.cs.uec.ac.jp
[b)] ksk@cs.uec.ac.jp

paper, we extend Frisch and Hosoya's faster backward typechecking method for MTTs to apply it to MFTs.

The contributions of this paper are as follows. First, based on Perst and Seidl's idea, this paper extended Frisch and Hosoya's backward typechecking method for MTT to a direct backward typechecking method for MFT. Second, this paper showed the correctness of this extended typechecking method. Finally, an implementation of this method showed that typechecking will be completed quickly even if the worst-case complexity increases.

The structure of this paper is as follows. Section 2 provides the terms and definitions required for our typechecking method. Section 3 introduces basic algorithms related to TAs or ATAs. Section 4 shows our typechecking method and its correctness. Section 5 discusses the implementation of our method. Section 6 presents the experimental results for our implementation. Section 7 compares the three approaches: Frisch and Hosoya's typechecking method for MTTs, Kobayashi et al.'s typechecking method for higher-order multi-parameter tree transducers, and our typechecking method for MFTs.

2. Preliminaries

In this section, we introduce the terms and definitions used in this paper.

2.1 Basic Notations

We denote the set of all natural numbers including zero by $\mathbb{N}$. For any natural number $n \in \mathbb{N}$, $[n]$ stands for the set of natural numbers from 1 to n; in particular, $[0]$ is the empty set.

From now on, for simplicity, the $i\,(1 \le i \le k)$-th element of k-vector $\boldsymbol{v}$ is denoted by v_i, and the 0-vector is denoted by $()$.

2.2 Trees and Forests

In this paper, *trees* are labeled binary trees which are defined below.

Definition 2.1. The set of all trees over an alphabet Σ is denoted by $\mathcal{B}_\Sigma$. A tree $t \in \mathcal{B}_\Sigma$ is syntactically given by

$$t ::= \varepsilon \mid a(t,t) \qquad (a \in \Sigma).$$

■

Here, ε is called a *leaf*, and symbol a in tree $a(t_1,t_2)$ is called a *node*. For trees ε and $a(t_1,t_2)$, their *roots* are ε and a, respectively. In addition, let the alphabet Σ be a finite set of symbols.

A *forest* is given as a list of unranked trees in which every node can have any number of children.

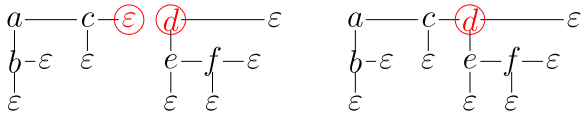
Definition 2.2. The set of all forests over an alphabet Σ is denoted by $\mathcal{F}_\Sigma$. A forest $f \in \mathcal{F}_\Sigma$ is syntactically given by

$$f ::= \varepsilon \mid a\langle f\rangle f \qquad (a \in \Sigma).$$

■

Here, ε means an *empty forest*. Intuitively, $a\langle f\rangle$ means an unranked tree whose root symbol is a, and a forest means a list of such trees.

For any forest $f = a\langle f_1\rangle f_2$, f_1 is called a *child forest* of f, and f_2 is called a *sibling forest* of f. The right most empty forest in forest f, that is, ε obtained by taking sibling forests repeatedly, is called a *hole* of f. In particular, the hole of ε is ε itself. For example, the hole of the forest $a\langle b\langle\varepsilon\rangle\varepsilon\rangle c\langle\varepsilon\rangle\varepsilon$ shown on the left of **Fig. 1** (a) is represented by the circled ε.



(a) Before concatenation (b) After concatenation

Fig. 1 Concatenation of forests.

By translating forests ε and $a\langle-\rangle-$ into trees ε and $a(-,-)$, respectively, every forest can be regarded as a tree. Therefore, we will treat forests as trees in this paper. In particular, tree automata and alternating tree automata are also used for forests.

However, forests and trees are different in the following sense. Since forests are lists, they can be concatenated. The concatenation $f_1 f_2$ of forests f_1, f_2 is the forest obtained by replacing the hole of f_1 with f_2.

Definition 2.3. For forests $f_1 = a_{11}\langle f_{11}\rangle\cdots a_{1n}\langle f_{1n}\rangle\varepsilon$ and $f_2 = a_{21}\langle f_{21}\rangle\cdots a_{2m}\langle f_{2m}\rangle\varepsilon$, the *concatenation* $f_1 f_2$ of f_1 and f_2 is defined by

$$f_1 f_2 = a_{11}\langle f_{11}\rangle\cdots a_{1n}\langle f_{1n}\rangle a_{21}\langle f_{21}\rangle\cdots a_{2m}\langle f_{2m}\rangle\varepsilon.$$

■

For example, the concatenation of the forests $a\langle b\langle\varepsilon\rangle\varepsilon\rangle c\langle\varepsilon\rangle\varepsilon$ and $d\langle e\langle\varepsilon\rangle f\langle\varepsilon\rangle\varepsilon\rangle\varepsilon$ in Fig. 1 (a) is the forest $a\langle b\langle\varepsilon\rangle\varepsilon\rangle c\langle\varepsilon\rangle d\langle e\langle\varepsilon\rangle f\langle\varepsilon\rangle\varepsilon\rangle\varepsilon$ in Fig. 1 (b)

2.3 Tree Automata

A *tree automaton* (TA) is a state machine dealing with trees.

Definition 2.4. Let Σ be an alphabet. A TA is a tuple $M = (Q, I, F, \Delta)$, where

- Q is a finite set of states,
- $I \subseteq Q$ is a finite set of initial states,
- $F \subseteq Q$ is a finite set of final states, and
- Δ is a finite set of transition rules of the form

$$a(q_1,q_2) \to q \qquad (a \in \Sigma,\ q_1,q_2,q \in Q).$$

■

A TA assigns a state $q \in Q$ to each node and leaf in a given tree. The assignment must follow the set Δ. For a symbol a and states q_1 and q_2, there may be more than one transition rule or no transition rules $a(q_1,q_2) \to q$. Therefore, a state assigned to a node of a tree is nondeterministically chosen according to the node symbol and states of children.

First, we define the *language* of a state of a TA. For a state $q \in Q$, the set of all trees *accepted* by q, that is, whose root is assigned q is called the *language of the state* q and denoted by $[\![q]\!]$.

Definition 2.5. Let q be a state of a TA $M = (Q, I, F, \Delta)$, and let $t_1, t_2 \in \mathcal{B}_\Sigma$ be trees. Then, $[\![q]\!]$ is the smallest set such that

- $\varepsilon \in [\![q]\!]$ if $q \in I$, and
- $a(t_1,t_2) \in [\![q]\!]$ if $\exists(a(q_1,q_2) \to q) \in \Delta.\ t_1 \in [\![q_1]\!] \ \wedge\ t_2 \in [\![q_2]\!]$.

■

Next, we define the language of a TA. For any tree $t \in \mathcal{B}_\Sigma$, it

can be determined whether a TA M accepts t. The set of all trees accepted by M, that is, whose root is assigned to any final state of M is called the *language of the TA* M and denoted by $\mathcal{L}(M)$. The language of the TA M is defined as the union of the languages of the final states of M.

Definition 2.6. Let $M = (Q, I, F, \Delta)$ be a TA. The language $\mathcal{L}(M)$ of the TA M is $\bigcup_{q \in F} [\![q]\!]$. ■

2.4 Deterministic Bottom-up Tree Automata

It is said that a TA M is *deterministic* if at most one state of a node is determined by the node symbol and states of children according to the transition rules of M.

Definition 2.7. Let Σ be an alphabet. A TA $M = (Q, I, F, \Delta)$ is deterministic if

- the set I of initial states is a singleton, and
- for any pair of states $q_1, q_2 \in Q$ and symbol $a \in \Sigma$, there exists at most one transition rule $a(q_1, q_2) \to q$ in Δ.

■

The 'at most one' condition allows nonexistence of transition rules for some symbols and states of children. Additionally, since the set I of initial states is a singleton, we will denote the deterministic TA by $(Q, \{q_I\}, F, \Delta)$.

Because of the properties of the transition rules of a deterministic TA, we have the following theorem that two distinct states cannot be assigned to the root for any tree.

Theorem 2.8. Let Σ be an alphabet, and let $M = (Q,), \{q_I\}, F, \Delta)$ be a deterministic TA. For any tree $t \in \mathcal{B}_\Sigma$ and states $q, q' \in Q$, $q = q'$ if $t \in [\![q]\!] \wedge t \in [\![q']\!]$. ■

Proof. By induction on the structure of t. □

On the other hand, it is said that a TA is *complete* if one or more states are always determined by a symbol and states of children.

Definition 2.9. Let Σ be an alphabet. A TA $M = (Q, I, F, \Delta)$ is complete if

- the set I of initial states has at least one state, and
- for any pair of states $q_1, q_2 \in Q$ and symbol $a \in \Sigma$, there exists at least one transition rule $a(q_1, q_2) \to q$ in Δ.

■

Because of the properties of the transition rules of a complete TA, the TA can always assign at least one state to the root for any tree.

Theorem 2.10. Let Σ be an alphabet, and let $M = (Q, I, F, \Delta)$ be a complete TA. For any tree $t \in \mathcal{B}_\Sigma$, there exists a state $q \in Q$ such that $t \in [\![q]\!]$. ■

Proof. By induction on the structure of t. □

A deterministic and complete TA uniquely assigns a state for each node of a given tree. We call it a *deterministic bottom-up tree automaton* (DBTA).

Definition 2.11. A TA M is a DBTA if M is deterministic and complete. ■

Because of the properties of the transition rules of a DBTA, exactly one state is assigned to the root for any tree.

2.5 Alternating Tree Automata

An *alternating tree automaton* (ATA) is a variant of the TA. Although the definition of ATAs is similar to that of TAs, the forms of their transition rules are different. In TAs, state transition is specified by a set of rules that assign a state to a node by its node symbol and states assigned to its children. In ATAs, state transition is specified by *conditional expressions* that assign a state to a node by its node symbol and all possible states assigned to its children.

Definition 2.12. Let Σ be an alphabet. An ATA is a tuple $M = (Q, I, F, \Phi)$, where

- Q is a finite set of states,
- $I \subseteq Q$ is a finite set of initial states,
- $F \subseteq Q$ is a finite set of final states, and
- Φ is a transition function $Q \times \Sigma \to \phi$ that gives a conditional expression to assign states to nodes of a tree. The conditional expression ϕ is syntactically given by

$$\phi ::= \phi \vee \phi \mid \phi \wedge \phi \mid \top \mid \bot \mid \downarrow_1 q \mid \downarrow_2 q.$$

■

The intuitive meaning of the conditional expression ϕ conforms to the logical expression. $\vee$ gives the logical OR of conditions, and $\wedge$ gives the logical AND of conditions. $\top$ and $\bot$ correspond to true and false, respectively. $\downarrow_i q$ means the condition that the root of the i-th child is assigned the state q.

Similarly to TAs, an ATA assigns states to each node and leaf of a tree. This assignment must follow the transition function Φ of the ATA.

First, we define the language of a state of an ATA. For a state $q \in Q$ of an ATA, the set of all trees *accepted* by q, that is, whose root is assigned q is called the *language of the state* q and denoted by $[\![q]\!]$. We also define the language of a conditional expression here. For a conditional expression ϕ of an ATA, a set of pairs of trees that satisfy the condition specified by ϕ is called the *language of the conditional expression* ϕ and denoted by $[\![\phi]\!]$.

Definition 2.13. Let $M = (Q, I, F, \Phi)$ be an ATA, $q \in Q$ be a state of M, and $t_1, t_2 \in \mathcal{B}_\Sigma$ be trees. Then, $[\![q]\!]$ is the smallest set such that

- $\varepsilon \in [\![q]\!]$ if $q \in I$, and
- $a(t_1, t_2) \in [\![q]\!]$ if $(t_1, t_2) \in [\![\Phi(q, a)]\!]$.

where $[\![\phi]\!]$ for a conditional expression ϕ is defined as follows. For a vector of trees $\boldsymbol{t} \in \mathcal{B}_\Sigma^k$ $(k = 0, 2)$, $\boldsymbol{t} \in [\![\phi]\!]$ holds if a judgement $\boldsymbol{t} \vdash \phi$ is derived by

- $\boldsymbol{t} \vdash \top$,
- $\boldsymbol{t} \vdash \phi_1 \wedge \phi_2$ if $\boldsymbol{t} \vdash \phi_1$ and $\boldsymbol{t} \vdash \phi_2$,
- $\boldsymbol{t} \vdash \phi_1 \vee \phi_2$ if $\boldsymbol{t} \vdash \phi_1$ or $\boldsymbol{t} \vdash \phi_2$, and
- $\boldsymbol{t} \vdash \downarrow_i q$ $(i = 1, 2)$ if $t_i \in [\![q]\!]$.

■

Next, we define the *language of an ATA*. Similarly to TAs, for any tree t, it can be determined whether an ATA accepts t. The set of all trees that are accepted by an ATA M is called the language of M and denoted by $\mathcal{L}(M)$. The language of an ATA M is defined as the union of the languages of the final states of M.

Definition 2.14. Let $M = (Q, I, F, \Phi)$ be an ATA. The language $\mathcal{L}(M)$ of the ATA M is $\bigcup_{q \in F} [\![q]\!]$. ■

Example 1. For trees over the alphabet $\Sigma = \{s\}$, an ATA M_{odd} that accepts a tree with an odd number of s is given by

$M_{\text{odd}} = (\{q_0, q_1\}, \{q_0\}, \{q_1\}, \Phi_{\text{odd}})$,
$\Phi_{\text{odd}}(q_0, s) = (\downarrow_1 q_0 \wedge \downarrow_2 q_1) \vee (\downarrow_1 q_1 \wedge \downarrow_2 q_0)$, and
$\Phi_{\text{odd}}(q_1, s) = (\downarrow_1 q_0 \wedge \downarrow_2 q_0) \vee (\downarrow_1 q_1 \wedge \downarrow_2 q_1)$.

q_0 accepts only trees with an odd number of s. q_1 accepts only trees with an even number of s. ■

2.6 Macro Forest Transducers

A *macro forest transducer* (MFT) is a collection of transformation rules over forests. The form of the transformation rules is restricted. Intuitively, the transformation induced by an MFT can be regarded as a multi-parameter mutually recursive function over forests. A function in an MFT can have multiple accumulating parameters, and the number of parameters is called *rank*. A transformation may output more than one forest or no forests. Therefore, each function induced by an MFT is regarded as a map from forests to sets of forests.

Definition 2.15. For forests over an alphabet Σ, an MFT $\mathcal{T}$ is a tuple (P, P_0, Π), where

- P is a finite set of ranked function names. $P^{(i)} \subseteq P$ is a set of function names each of which has rank $i \in \mathbb{N}$,
- $P_0 \subseteq P^{(0)}$ is a finite set of initial function names, and
- Π is a set of transformation rules of the form

$$p(a\langle x_1\rangle x_2, y_1, \ldots, y_k) \to e \quad (p \in P^{(k)},\ k \in \mathbb{N},\ a \in \Sigma)$$
$$p(\varepsilon, y_1, \ldots, y_k) \to e \qquad (p \in P^{(k)},\ k \in \mathbb{N})$$

where e is called a *right-hand-side expression*. The right-hand-side expression e for $p(a\langle x_1\rangle x_2, y_1, \ldots, y_k)$ is syntactically given by

$$\begin{aligned} e ::= {} & b\langle e_1\rangle e_2 && (b \in \Sigma)\\ \mid {} & \varepsilon \\ \mid {} & q(x_h,\ e_1, \ldots, e_l) && (q \in P^{(l)},\ l \in \mathbb{N},\ h \in [2])\\ \mid {} & y_i && (i \in [k])\\ \mid {} & e_1 e_2. \end{aligned}$$

The right-hand-side expression e for $p(\varepsilon, y_1, \ldots, y_k)$ is given in the same syntax excluding $q(x_h,\ e_1, \ldots, e_l)$. ■

We denote a function name p that has a rank k by $p^{(k)}$. The rank k shows the number of accumulating parameters $y_1, \ldots, y_k$ given to the function name $p^{(k)}$. Each syntax of the right-hand-side expressions of a transformation rule for $p^{(k)}$ has the following meaning: $b\langle e_1\rangle e_2$ and ε construct forests, $q^{(l)}(x_h,\ e_1, \ldots, e_l)$ means a function call, y_i refers to an accumulating parameter, and $e_1 e_2$ concatenates two forests. In particular, the first argument of a function call must be the child x_1 or the sibling x_2 of the first argument of the caller, and the other arguments can be arbitrary right-hand-side expressions. For each left-hand side, there may be more than one right-hand-side expression or no right-hand-side expressions. Therefore, a transformation induced by an MFT is nondeterministic, and its semantics is given by a function from forests to sets of forests.

In summary, P is the set of function names, Π is the set of transformation rules, P_0 is the set of initial function names whose rank must be 0, and functions in P_0 should be called first when applying an MFT to a given forest.

Next, we define the *semantics of an MFT* $[\![\cdot]\!]$. It is known through evaluation strategies that a nondeterministic MFT has two styles of semantics evaluation results obtained by using these two semantics are different in general. In this paper, we consider only one of them, called In-Out (IO). Intuitively, IO-semantics $[\![\cdot]\!]$ evaluates a right-hand-side expression from inside to outside. $[\![\cdot]\!]$ takes a function name $p^{(k)}$, an input forest f, and a k-dimensional vector $\boldsymbol{\tau}$ consisting of accumulating parameters τ_i given by function calls and then returns the set of forests as evaluation results of $p^{(k)}$. The set $[\![p^{(k)}]\!](f, \boldsymbol{\tau})$ is defined below.

Definition 2.16. For a forest $a\langle f_1\rangle f_2$ over an alphabet Σ, a vector $\boldsymbol{\tau} \in {\mathcal{F}_\Sigma}^k$ and an MFT $\mathcal{T} = (P, P_0, \Pi)$, the semantics $[\![p^{(k)}]\!]$ with $p \in P$ is defined by

$$[\![p^{(k)}]\!](a\langle f_1\rangle f_2, \boldsymbol{\tau}) = \bigcup_{(p^{(k)}(a\langle x_1\rangle x_2, y_1, \ldots, y_k) \to e) \in \Pi} [\![e]\!]((f_1, f_2), \boldsymbol{\tau}), \text{ and}$$
$$[\![p^{(k)}]\!](\varepsilon, \boldsymbol{\tau}) = \bigcup_{(p^{(k)}(\varepsilon, y_1, \ldots, y_k) \to e) \in \Pi} [\![e]\!]((), \boldsymbol{\tau})$$

where the *semantics* $[\![e]\!]$ *of the right-hand side* e is defined by

$$\begin{aligned} & [\![a\langle e_1\rangle e_2]\!](\boldsymbol{f}, \boldsymbol{\tau})\\ & = \{a\langle f_1'\rangle f_2' \mid f_1' \in [\![e_1]\!](\boldsymbol{f}, \boldsymbol{\tau}),\ f_2' \in [\![e_2]\!](\boldsymbol{f}, \boldsymbol{\tau})\},\\ & [\![\varepsilon]\!](\boldsymbol{f}, \boldsymbol{\tau}) = \{\varepsilon\},\\ & [\![p^{(l)}(x_h, e_1, \ldots, e_l)]\!](\boldsymbol{f}, \boldsymbol{\tau})\\ & = \{[\![p^{(l)}]\!](f_h, \boldsymbol{\tau'}) \mid \forall j \in [l].\ \tau_j' \in [\![e_j]\!](\boldsymbol{f}, \boldsymbol{\tau})\},\\ & [\![y_i]\!](\boldsymbol{f}, \boldsymbol{\tau}) = \{\tau_i\}, \text{ and}\\ & [\![e_1 e_2]\!](\boldsymbol{f}, \boldsymbol{\tau}) = \{f_1' f_2' \mid f_1' \in [\![e_1]\!](\boldsymbol{f}, \boldsymbol{\tau}),\ f_2' \in [\![e_2]\!](\boldsymbol{f}, \boldsymbol{\tau})\}. \end{aligned}$$

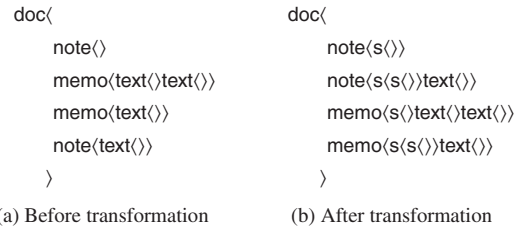
■

An MFT $\mathcal{T} = (P, P_0, \Pi)$ transforms a forest f by applying the semantics of the initial function names P_0 to f. Therefore, the *transformation by* $\mathcal{T}$ for a forest f is the union of $[\![p_0^{(0)}]\!](f, ())$ for all $p_0 \in P_0$.

Definition 2.17. The set of $\mathcal{T}(f)$ of all forests transformed from a forest f by an MFT $\mathcal{T} = (P, P_0, \Pi)$ is $\bigcup_{p \in P_0} [\![p]\!](f, ())$. ■

Example 2. Over an alphabet $\Sigma = \{\mathsf{doc}, \mathsf{note}, \mathsf{memo}, \mathsf{text}, \mathsf{s}\}$, we consider forests in which each of the children of doc is either memo or note and the children of memo and note are text. For these forests, an MFT $\mathcal{T}_{\text{ex}} = (\{p_0^{(0)}, p_{\text{note}}^{(1)}, p_{\text{memo}}^{(1)}, id^{(0)}\}, \{p_0^{(0)}\}, \Pi_{\text{ex}})$ with

$$\begin{aligned} \Pi_{\text{ex}} = \Big\{ & p_0^{(0)}(\mathsf{doc}\langle x_1\rangle x_2) \to \mathsf{doc}\langle p_{\text{note}}^{(1)}(x_1, \varepsilon)\ p_{\text{memo}}^{(1)}(x_1, \varepsilon)\rangle,\\ & p_{\text{note}}^{(1)}(\varepsilon, y_1) \to \varepsilon,\\ & p_{\text{note}}^{(1)}(\mathsf{note}\langle x_1\rangle x_2, y_1)\\ & \quad \to \mathsf{note}\langle \mathsf{s}\langle y_1\rangle id^{(0)}(x_1)\rangle p_{\text{note}}^{(1)}(x_2, \mathsf{s}\langle y_1\rangle),\\ & p_{\text{note}}^{(1)}(\mathsf{memo}\langle x_1\rangle x_2, y_1) \to p_{\text{note}}^{(1)}(x_2),\\ & p_{\text{memo}}^{(1)}(\varepsilon, y_1) \to \varepsilon,\\ & p_{\text{memo}}^{(1)}(\mathsf{memo}\langle x_1\rangle x_2, y_1)\\ & \quad \to \mathsf{memo}\langle \mathsf{s}\langle y_1\rangle id^{(0)}(x_1)\rangle p_{\text{memo}}^{(1)}(x_2, \mathsf{s}\langle y_1\rangle),\\ & p_{\text{memo}}^{(1)}(\mathsf{note}\langle x_1\rangle x_2, y_1) \to p_{\text{memo}}^{(1)}(x_2),\\ & id^{(0)}(\varepsilon) \to \varepsilon,\\ & id^{(0)}(\mathsf{text}\langle x_1\rangle x_2) \to \mathsf{text}\langle id^{(0)}(x_1)\rangle id^{(0)}(x_2)\Big\} \end{aligned}$$



(a) Before transformation (b) After transformation

Fig. 2 Transformation example of MFT $\mathcal{T}_{\text{ex}}$.

separates two kinds of children, memo and note, and adds a serial number with s (successor operator for Peano numbers) to each. Here, we denote an expression $(x, \boldsymbol{y})$ with 1-dimensional vector $\boldsymbol{y} = (y_1)$ by (x, y_1) and $a\langle\varepsilon\rangle\varepsilon$ by $a\langle\rangle$ briefly. By this MFT, for example, the forest in **Fig. 2** (a) is transformed into the forest in Fig. 2 (b).

The first argument of the function $p^{(1)}_{\text{note}}$ is a forest whose root is note or memo, and the second argument is a forest that represents a serial number that is added to note. The arguments of the function $p^{(1)}_{\text{memo}}$ are similar to those of $p^{(1)}_{\text{note}}$. The functions $p^{(1)}_{\text{note}}$ and $p^{(1)}_{\text{memo}}$ collect all forests whose roots are note and memo. The return value of the initial function $p^{(0)}_0$ is a concatenation of these results. $id^{(0)}$ imitates the identity function. ■

2.7 Typechecking of MFTs

In this paper, a *type* of a tree is specified by the language of a TA. In other words, a tree t is included in the type specified by a TA M if and only if t is accepted by M.

Definition 2.18. Let M be a TA. A tree t has the type specified by M if $t \in \mathcal{L}(M)$. ■

For tree transducers, *typechecking* is performed to verify if any tree that satisfies certain properties (that is, any tree of an input type M_{in}) is always transformed into a tree that satisfies other certain properties (that is, a tree of an output type M_{out}).

Although there are several kinds of type representation and tree transducers, we assume that the input type is specified by a TA, output type is specified by a DBTA, and tree transducer is given by an MFT. Typechecking for an MFT is formally defined as follows.

Definition 2.19. Let Σ be an alphabet, $\mathcal{T}$ be an MFT, M_{in} be an input type TA, and M_{out} be an output type DBTA. Then, typechecking of $\mathcal{T}$ against M_{in} and M_{out} is performed to verify the formula

$$\forall t \in \mathcal{F}_\Sigma. \quad t \in \mathcal{L}(M_{\text{in}}) \Rightarrow \mathcal{T}(t) \subseteq \mathcal{L}(M_{\text{out}}),$$

which is often denoted by $\mathcal{T}(\mathcal{L}(M_{\text{in}})) \subseteq \mathcal{L}(M_{\text{out}})$ simply. ■

The following fact about typechecking for most tree transducers is not limited to MFTs and is known.

Theorem 2.20. For any input type TA M_{in}, any output type DBTA M_{out}, and any MFT $\mathcal{T}$, we have

$$\mathcal{T}(\mathcal{L}(M_{\text{in}})) \subseteq \mathcal{L}(M_{\text{out}}) \iff \mathcal{T}^{-1}(\mathcal{L}(M_{\text{out}})^{\complement}) \cap \mathcal{L}(M_{\text{in}}) = \emptyset$$

where $\mathcal{L}(M_{\text{out}})^{\complement}$ is the complement $\mathcal{F}_\Sigma \setminus \mathcal{L}(M_{\text{out}})$ of the set $\mathcal{L}(M_{\text{out}})$, and $\mathcal{T}^{-1}(A)$ is the inverse image $\{t \mid \mathcal{T}(t) \cap A \neq \emptyset\}$ of a set A of trees under an MFT $\mathcal{T}$. ■

Proof. By the definition of inverse image and trivial deformations about sets. □

3. Algorithms about TAs and ATAs

This section introduces basic algorithms and construction methods for the TAs and ATAs that are used in this paper.

3.1 Construction of Complement TA of DBTA

In the case where a given TA is a DBTA, it is known that a TA that rejects all trees accepted by the original TA but accepts all trees rejected by the original TA can be constructed by swapping final states with non-final states [10].

Theorem 3.1. For a DBTA $M = (Q, I, F, \Delta)$, consider the TA $M' = (Q, I, Q \setminus F, \Delta)$. Then, M' accepts trees rejected by M. That is, for an alphabet Σ, $\mathcal{L}(M) \cup \mathcal{L}(M') = \mathcal{B}_\Sigma$, and $\mathcal{L}(M) \cap \mathcal{L}(M') = \emptyset$. ■

Because of the definition of a DBTA, all transitions are uniquely determined from the bottom. Therefore, we can obtain the complement as a language by only swapping final states with non-final states.

3.2 Transformation of TA into ATA

Any TA can be transformed into an ATA equivalent to the TA.

Theorem 3.2. For a TA $M = (Q, I, F, \Delta)$, let $M' = (Q, I, F, \Phi)$ be the ATA constructed by

$$\Phi(q, a) = \bigvee_{(a(q_1,q_2)\to q)\in\Delta} \downarrow_1 q_1 \wedge \downarrow_2 q_2.$$

Then, $\mathcal{L}(M') = \mathcal{L}(M)$ holds. ■

Proof. Let $[\![q]\!]_M$ be the language of a state q for the TA M, and let $[\![q]\!]_{M'}$ be the language of a state q for the ATA M'. $t \in [\![q]\!]_{M'} \iff t \in [\![q]\!]_M$ is proved by induction on the structure of $t \in \mathcal{B}_\Sigma$. Thus, $\mathcal{L}(M') = \mathcal{L}(M)$ holds. □

3.3 Intersection of ATAs

For two ATAs, M_1 and M_2, an ATA M such that $\mathcal{L}(M) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$ is called an ATA accepting the intersection of M_1 and M_2. The ATA M can be constructed as follows.

Theorem 3.3. Let $M_1 = (Q_1, I_1, F_1, \Phi_1)$ and $M_2 = (Q_2, I_2, F_2, \Phi_2)$ be ATAs. Without loss of generality, we assume $Q_1 \cap Q_2 = \emptyset$. Let $M = (Q, I, F, \Phi)$ be the ATA given by

$$Q = Q_1 \cup Q_2 \cup \{q_{\text{new}}\},$$

$$I = \begin{cases} I_1 \cup I_2 \cup \{q_{\text{new}}\} & (I_1 \cap F_1 \neq \emptyset \wedge I_2 \cap F_2 \neq \emptyset) \\ I_1 \cup I_2 & (\text{otherwise}), \end{cases}$$

$F = \{q_{\text{new}}\}$, and

$$\Phi(q, a) = \begin{cases} \Phi_1(q, a) & (q \in Q_1) \\ \Phi_2(q, a) & (q \in Q_2) \\ \displaystyle\bigvee_{(q_1,q_2)\in F_1\times F_2} \Phi_1(q_1, a) \wedge \Phi_2(q_2, a) & (q = q_{\text{new}}) \end{cases}$$

where q_{new} is a fresh state such that $q_{\text{new}} \notin Q_1 \cup Q_2$.

Then, $\mathcal{L}(M) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$ holds. ■

Proof. $t \in [\![q_{\text{new}}]\!] \iff t \in \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$ can be proved

by induction on the structure of $t \in \mathcal{B}_\Sigma$. Thus, $\mathcal{L}(M) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$ holds. □

4. Typechecking Method for MFTs

In this section, we show a typechecking method for MFTs using an ATA that extends the typechecking method of Frisch and Hosoya [6]. First, we propose additional states that are necessary for our extension of TAs and define the semantics of the language. Next, using the additional states, we extend the inverse type inference method for MTTs of Frisch and Hosoya to MFTs and show its correctness. Finally, we show the whole structure of our typechecking for MFTs.

4.1 Language of Hole-relational State

A TA is an automaton specifically designed for trees. When a TA is applied to forests, the states of the root of a forest f can be determined by transition rules with the states of the child and the sibling of the forest f, while the states of the root of a concatenation $f_1 f_2$ cannot be determined with states of forests, f_1 and f_2. The naive typechecking method for MFTs of Perst and Seidl [2] solves this problem by considering pairs of states. In this subsection, we propose assigning special states for concatenation of forests to a TA, which was inspired by the method of Perst and Seidl, and define the semantics of this language. By using these special states, we can immediately find the states of the root of a forest after concatenation from the states of the roots of two forests to be concatenated.

For states $q', q \in Q$ of a TA $M = (Q, I, F, \Delta)$, $q' \triangleleft q$ with $(q', q) \in Q^2$ is called a *hole-relational state*. Intuitively, assigning the state $q' \triangleleft q$ to the root of a forest f corresponds to the fact that q' can be assigned to the root of f by applying the transition rules when q is assigned to the hole of f. The set of all forests whose root can be assigned a hole-relational state $q' \triangleleft q$ is called the *language of the hole-relational state* for $q' \triangleleft q$ and denoted by $[\![q' \triangleleft q]\!]$.

Definition 4.1. For a TA $M = (Q, I, F, \Delta)$, states $q', q \in Q$, and a forest f, $f \in [\![q' \triangleleft q]\!]$ holds if we have

$$\forall f' \in [\![q]\!].\ f f' \in [\![q']\!].$$

■

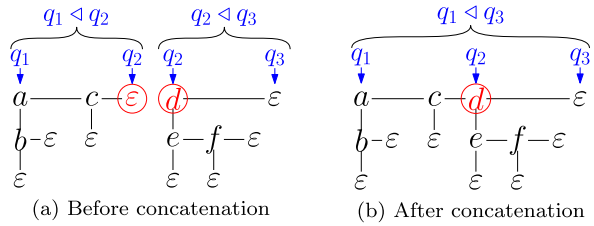
Briefly, for a vector $\boldsymbol{f} \in {\mathcal{F}_\Sigma}^k$ and vectors of states $\boldsymbol{q'}, \boldsymbol{q} \in Q^k$, we denote $\forall i \in [k].\ f_i \in [\![q'_i \triangleleft q_i]\!]$ by $\boldsymbol{f} \in [\![\boldsymbol{q'} \triangleleft \boldsymbol{q}]\!]$.

Example 3. For states q_1, q_2, and q_3 of a TA, consider forests $f_1 = a\langle b\langle\varepsilon\rangle\varepsilon\rangle c\langle\varepsilon\rangle\varepsilon$ and $f_2 = d\langle e\langle\varepsilon\rangle f\langle\varepsilon\rangle\varepsilon\rangle\varepsilon$. When q_1 and q_2 are assigned to the root and the hole of f_1, respectively, $q_1 \triangleleft q_2$ is assigned to the root of f_1 as shown in **Fig. 3** (a). Additionally, when $q_2 \triangleleft q_3$ is assigned to the root of f_2, $q_1 \triangleleft q_3$ is assigned to the root of $f_1 f_2$ as shown Fig. 3 (b). ■

In the language of a hole-relational state, we have the following theorem related to transition rules and concatenation of forests.

Theorem 4.2. For states $q_1, q_2, q_3 \in Q$ of a TA $M = (Q, I, F, \Delta)$, an initial state $q_I \in I$, and forests $f_1, f_2 \in \mathcal{F}_\Sigma$,

$$a\langle f_1\rangle f_2 \in [\![q \triangleleft q_3]\!] \iff \exists (a(q_1, q_2) \to q) \in \Delta.\ f_1 \in [\![q_1]\!] \wedge f_2 \in [\![q_2 \triangleleft q_3]\!]$$



(a) Before concatenation (b) After concatenation

Fig. 3 Assignment of hole-relational state.

holds. ■

Proof. By the definitions of the language of a hole-relational state of a TA and the language of a state of a TA, we obtain

$$\begin{aligned}
&a\langle f_1\rangle f_2 \in [\![q \triangleleft q_3]\!] \\
&\iff \forall f' \in [\![q_3]\!].\ a\langle f_1\rangle f_2 f' \in [\![q]\!] \\
&\iff \forall f' \in [\![q_3]\!].\ \exists (a(q_1, q_2) \to q) \in \Delta. \\
&\qquad\qquad f_1 \in [\![q_1]\!] \wedge f_2 f' \in [\![q_2]\!] \\
&\iff \exists (a(q_1, q_2) \to q) \in \Delta.\ f_1 \in [\![q_1]\!] \wedge f_2 \in [\![q_2 \triangleleft q_3]\!].
\end{aligned}$$

□

If a TA is deterministic, when an initial state is assigned to the hole, we have the following theorem related to the languages of a state and a hole-relational state.

Theorem 4.3. For any state $q \in Q$ of a deterministic TA $M = (Q, \{q_I\}, F, \Delta)$ and any forest $f \in \mathcal{F}_\Sigma$, we have $f \in [\![q \triangleleft q_I]\!] \iff f \in [\![q]\!]$. ■

Proof. By induction on the structure of f.

We consider the case $f = \varepsilon$. We prove this case by the definition of the language of a hole-relational state of a TA, definition of a language of a state of a TA, and Theorem 2.8. Then, we find

$$\begin{aligned}
&\varepsilon \in [\![q \triangleleft q_I]\!] \\
&\iff \forall f' \in [\![q_I]\!].\ f' \in [\![q]\!] \\
&\iff q = q_I \\
&\iff \varepsilon \in [\![q]\!].
\end{aligned}$$

Next, we consider the case $f = a\langle f_1\rangle f_2$. We have

$$\begin{aligned}
&a\langle f_1\rangle f_2 \in [\![q \triangleleft q_I]\!] \\
&\iff \quad \{\text{The definition of a hole-relational state of a TA}\} \\
&\quad \forall f' \in [\![q_I]\!].\ a\langle f_1\rangle f_2 f' \in [\![q]\!] \\
&\iff \quad \{\text{The definition of a state of a TA}\} \\
&\quad \forall f' \in [\![q_I]\!].\ \exists (a(q_1, q_2) \to q). \\
&\qquad f_1 \in [\![q_1]\!] \wedge f_2 f' \in [\![q_2]\!] \\
&\iff \quad \{\text{The definition of a hole-relational state of a TA}\} \\
&\quad \exists (a(q_1, q_2) \to q).\ f_1 \in [\![q_1]\!] \wedge f_2 \in [\![q_2 \triangleleft q_I]\!] \\
&\iff \quad \{\text{The induction hypothesis}\} \\
&\quad \exists (a(q_1, q_2) \to q).\ f_1 \in [\![q_1]\!] \wedge f_2 \in [\![q_2]\!] \\
&\iff \quad \{\text{The definition of a state of a TA}\} \\
&\quad a(f_1, f_2) \in [\![q]\!].
\end{aligned}$$

Thus, Theorem 4.3 is proved by induction on the structure of f. □

From Theorem 4.3, if a TA is deterministic, we obtain the following theorem by transformation of Theorem 4.2 immediately.

Theorem 4.4. For states $q_1, q_2, q_3 \in Q$ of a deterministic TA $M = (Q, \{q_I\}, F, \Delta)$ and forests $f_1, f_2 \in \mathcal{F}_\Sigma$, we have

$$a\langle f_1\rangle f_2 \in [\![q \triangleleft q_3]\!] \iff \exists (a(q_1, q_2) \to q) \in \Delta. \\ f_1 \in [\![q_1 \triangleleft q_I]\!] \wedge f_2 \in [\![q_2 \triangleleft q_3]\!].$$

■

Proof. By Theorem 4.2 and Theorem 4.3, we obtain

$$\begin{aligned} & a\langle f_1\rangle f_2 \in [\![q \triangleleft q_3]\!] \\ & \iff \exists (a(q_1, q_2) \to q) \in \Delta.\, f_1 \in [\![q_1]\!] \wedge f_2 \in [\![q_2 \triangleleft q_3]\!] \\ & \iff \exists (a(q_1, q_2) \to q) \in \Delta.\, f_1 \in [\![q_1 \triangleleft q_I]\!] \wedge f_2 \in [\![q_2 \triangleleft q_3]\!]. \end{aligned}$$

□

On the other hand, if a TA is complete, for any forest f, there exists a hole-relational state whose language includes f, which is similar to the language of a state of a TA.

Theorem 4.5. Let Σ be an alphabet and $M = (Q, I, F, \Delta)$ be a TA. If M is complete, for any forest $f \in \mathcal{F}_\Sigma$, there exist states $q', q \in Q$ such that $f \in [\![q' \triangleleft q]\!]$. ■

Proof. By induction on the structure of $f \in \mathcal{F}_\Sigma$.

Case $t = \varepsilon$

In accordance with the definition of completeness, I has at least one state. Let q_I be one of these states. By the definition of the language of a hole-relational state and the definition of the language of a state of a TA, $\varepsilon \in [\![q_I \triangleleft q_I]\!]$ holds.

Case $t = a\langle f_1\rangle f_2$

By Theorem 2.10, there exists $q_1 \in Q$ such that $f_1 \in [\![q_1]\!]$. By the induction hypothesis, there exist $q_2, q_3 \in Q$ such that $f_2 \in [\![q_2 \triangleleft q_3]\!]$. Since there exists a transition rule $a(q_1, q_2) \to q$ ($q \in Q$) in Δ because the TA is complete, $a\langle f_1\rangle f_2 \in [\![q \triangleleft q_3]\!]$ holds by Theorem 4.2.

Thus, Theorem 4.5 is proved by induction on the structure of the forest f. □

4.2 Inverse Type Inference for MFT with ATA

Given an MFT $\mathcal{T}$ and a DBTA M, inferring the inverse image of $\mathcal{L}(M)$ under $\mathcal{T}$ is called *inverse type inference*. Frisch and Hosoya proposed an inverse type inference method for MTTs. In their method, given an MTT $\mathcal{T} = (P, P_0, \Pi)$ and a DBTA $M = (Q, \{q_I\}, F, \Delta)$, they construct an ATA that has a state $\langle p^{(k)}, q, \boldsymbol{s}\rangle$ for $p^{(k)} \in P$, $q \in Q$, and $\boldsymbol{s} \in Q^k$. Intuitively, $t \in [\![\langle p^{(k)}, q, \boldsymbol{s}\rangle]\!]$ for a tree t means that there exists at least one tree t' such that $t' \in [\![q]\!]$ in the transformed forests $[\![p^{(k)}]\!](t, \boldsymbol{\tau})$ by a function $p^{(k)}$ with a vector $\boldsymbol{\tau}$ such that $\forall i \in [k].\, \tau_i \in [\![s_i]\!]$ are used as parameters. The set $\bigcup_{p_0^{(0)} \in P_0,\, q_F \in F} [\![\langle p_0^{(0)}, q_F, ()\rangle]\!]$ represents all trees whose root can be assigned a final state q_F after the transformation by a initial function $p_0^{(0)}$ so that the inverse image under $\mathcal{T}$ can be represented by the set.

However, in their method, an excepted return value of $\langle p^{(k)}, q, \boldsymbol{s}\rangle$ obtained by a transition function cannot be represented in the case that a right-hand-side expression of function $p^{(k)}$ represents a concatenation of forests for a state $\langle p^{(k)}, q, \boldsymbol{s}\rangle$. That is caused by the fact that the language of a state of a TA does not support state transition by concatenation. To solve this problem, in this paper, we use hole-relational states instead of original states. By using hole-relational states, for example, we can find immediately that $q_1 \triangleleft q_3$ can be assigned to the concatenation $f_1 f_2$ if there exist forests f_1 and f_2 such that $f_1 \in [\![q_1 \triangleleft q_2]\!]$ and $f_2 \in [\![q_2 \triangleleft q_3]\!]$ hold for states q_1, q_2, and q_3 of a TA.

Based on the above, we propose an inverse type inference method that constructs an ATA accepting the inverse image under an MFT by extending their method. While they construct an ATA related to states of a DBTA, we construct an ATA related to hole-relational states of a DBTA in a similar way.

Definition 4.6. For an MFT $\mathcal{T} = (P, P_0, \Pi)$ and a DBTA $M = (Q, \{q_I\}, F, \Delta)$, the ATA $\mathrm{Iv}(\mathcal{T}, M) = (\Xi, \Xi_I, \Xi_F, \Phi)$ is defined as below.

$$\Xi = \{\langle p^{(k)}, (q', q), (\boldsymbol{s}', \boldsymbol{s})\rangle \mid p^{(k)} \in P,\, q', q \in Q,\, \boldsymbol{s}', \boldsymbol{s} \in Q^k\}$$

$$\Xi_I = \left\{ \langle p^{(k)}, (q', q), (\boldsymbol{s}', \boldsymbol{s})\rangle \in \Xi \;\middle|\; () \in \left[\!\!\left[\bigvee_{(p^{(k)}(\varepsilon, y_1, \ldots, y_k) \to e) \in \Pi} \mathsf{Inf}(e, (q', q), (\boldsymbol{s}', \boldsymbol{s})) \right]\!\!\right] \right\}$$

$$\Xi_F = \{\langle p^{(0)}, (q_F, q_I), ((), ())\rangle \mid p^{(0)} \in P_0,\, q_F \in F,\, q_I \in I\}$$

$$\Phi(\langle p^{(k)}, (q', q), (\boldsymbol{s}', \boldsymbol{s})\rangle, a) = \bigvee_{(p^{(k)}(a\langle f_1\rangle f_2, y_1, \ldots, y_k) \to e) \in \Pi} \mathsf{Inf}(e, (q', q), (\boldsymbol{s}', \boldsymbol{s}))$$

Here, Inf is the function that takes a right-hand-side expression, a pair of states of M, and a pair of vectors of states of M and returns a conditional expression of the ATA. The function Inf is defined by cases about the right-hand-side expression as follows.

$$\mathsf{Inf}(\varepsilon, (q', q), (\boldsymbol{s}', \boldsymbol{s})) = \begin{cases} \top & (q' = q) \\ \bot & (\text{otherwise}) \end{cases}$$

$$\mathsf{Inf}(a\langle e_1\rangle e_2, (q', q), (\boldsymbol{s}', \boldsymbol{s})) = \bigvee_{(a(r_1, r_2) \to q') \in \Delta} \mathsf{Inf}(e_1, (r_1, q_I), (\boldsymbol{s}', \boldsymbol{s})) \wedge \mathsf{Inf}(e_2, (r_2, q), (\boldsymbol{s}', \boldsymbol{s}))$$

$$\mathsf{Inf}(p^{(l)}(x_h, e_1, \ldots, e_l), (q', q), (\boldsymbol{s}', \boldsymbol{s})) = \bigvee_{\boldsymbol{t}', \boldsymbol{t} \in Q^l} \left(\downarrow_h \langle p^{(l)}, (q', q), (\boldsymbol{t}', \boldsymbol{t})\rangle \wedge \bigwedge_{j \in [l]} \mathsf{Inf}(e_j, (t'_j, t_j), (\boldsymbol{s}', \boldsymbol{s})) \right)$$

$$\mathsf{Inf}(y_j, (q', q), (\boldsymbol{s}', \boldsymbol{s})) = \begin{cases} \top & (q' = s'_j \wedge q = s_j) \\ \bot & (\text{otherwise}) \end{cases}$$

$$\mathsf{Inf}(e_1 e_2, (q', q), (\boldsymbol{s}', \boldsymbol{s})) = \bigvee_{r \in Q} \mathsf{Inf}(e_1, (q', r), (\boldsymbol{s}', \boldsymbol{s})) \wedge \mathsf{Inf}(e_2, (r, q), (\boldsymbol{s}', \boldsymbol{s}))$$

■

Iv constructs an ATA that accepts the inverse image of $\mathcal{L}(M)$ under $\mathcal{T}$. Intuitively, for a forest f, $\langle p^{(k)}, (q', q), (\boldsymbol{s}', \boldsymbol{s})\rangle$ can be assigned to the root of f when $[\![q' \triangleleft q]\!]$ includes a tree in the set $[\![p^{(k)}]\!](f,\ f_1, \ldots, f_k)$ of transformed trees by $p^{(k)}$ taking forests $f_i \in [\![s'_i \triangleleft s_i]\!]$. We show that $\mathrm{Iv}(\mathcal{T}, M)$ accepts the inverse image, that is, $\mathcal{L}(\mathrm{Iv}(\mathcal{T}, M)) = \mathcal{T}^{-1}(\mathcal{L}(M))$ and prove the correctness of the proposed construction.

First, we prove Lemma 4.7. Second, using Lemma 4.7, we prove Lemma 4.8. Finally, using Lemma 4.8, we prove Theorem 4.9 which provides the correctness of Iv.

Lemma 4.7. For the function Inf for any MFT $\mathcal{T} = (P, P_0, \Pi)$ and any DBTA $M = (Q, \{q_I\}, F, \Delta)$, we have

$$
\begin{aligned}
&\forall n \in \{0,2\}.\, \forall \boldsymbol{f} \in {\mathcal{F}_\Sigma}^n.\\
&(\forall h \in [n].\, \forall p^{(k)} \in P.\, \forall \boldsymbol{\tau} \in {\mathcal{F}_\Sigma}^k.\, \forall q', q \in Q.\\
&\ (\exists \boldsymbol{s'}, \exists \boldsymbol{s} \in Q^k.\\
&\ \boldsymbol{\tau} \in [\![\boldsymbol{s'} \triangleleft \boldsymbol{s}]\!] \ \wedge\ f_h \in [\![\langle p^{(k)}, (q', q), (\boldsymbol{s'}, \boldsymbol{s})\rangle]\!])\\
&\ \iff [\![p^{(k)}]\!](f_h, \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!] \neq \emptyset) \quad \cdots\cdots (H)\\
&\Longrightarrow\\
&(\forall e : \text{right-hand-side expression of } \mathcal{T}.\\
&\forall k \in \mathbb{N}.\, \forall \boldsymbol{\tau} \in {\mathcal{F}_\Sigma}^k.\, \forall q', q \in Q.\\
&\ (\exists \boldsymbol{s'}, \exists \boldsymbol{s} \in Q^k.\\
&\ \boldsymbol{\tau} \in [\![\boldsymbol{s'} \triangleleft \boldsymbol{s}]\!] \ \wedge\ \boldsymbol{f} \in [\![\mathsf{Inf}(e, (q', q), (\boldsymbol{s'}, \boldsymbol{s}))]\!])\\
&\ \iff [\![e]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!] \neq \emptyset).
\end{aligned}
$$

■

Proof. By induction of the structure of the right-hand-side expression e.

Consider the case $e = b\langle e_1\rangle e_2$. For any $\boldsymbol{\tau} \in {\mathcal{F}_\Sigma}^k$, we have

$$
\begin{aligned}
&\exists \boldsymbol{s'}, \exists \boldsymbol{s}.\, \boldsymbol{\tau} \in [\![\boldsymbol{s'} \triangleleft \boldsymbol{s}]\!]\\
&\quad \wedge\ \boldsymbol{f} \in [\![\mathsf{Inf}(b\langle e_1\rangle e_2, (q', q), (\boldsymbol{s'}, \boldsymbol{s}))]\!]\\
&\overset{(1)}{\iff} \exists \boldsymbol{s'}, \exists \boldsymbol{s}.\, \boldsymbol{\tau} \in [\![\boldsymbol{s'} \triangleleft \boldsymbol{s}]\!]\\
&\quad \wedge\ \boldsymbol{f} \in \left[\!\!\left[\bigvee_{(b(r_1,r_2)\to q') \in \Delta} \mathsf{Inf}(e_1, (r_1, q_I), (\boldsymbol{s'}, \boldsymbol{s})) \right.\right.\\
&\qquad\qquad \left.\left. \wedge\ \mathsf{Inf}(e_2, (r_2, q), (\boldsymbol{s'}, \boldsymbol{s})) \right]\!\!\right]\\
&\overset{(2)}{\iff} \exists \boldsymbol{s'}, \exists \boldsymbol{s}.\, \boldsymbol{\tau} \in [\![\boldsymbol{s'} \triangleleft \boldsymbol{s}]\!]\\
&\quad \wedge\ (\exists (b(r_1, r_2) \to q') \in \Delta.\\
&\qquad \boldsymbol{f} \in [\![\mathsf{Inf}(e_1, (r_1, q_I), (\boldsymbol{s'}, \boldsymbol{s}))]\!]\\
&\qquad \wedge\ \boldsymbol{f} \in [\![\mathsf{Inf}(e_2, (r_2, q), (\boldsymbol{s'}, \boldsymbol{s}))]\!])\\
&\overset{(3)}{\iff} \exists (b(r_1, r_2) \to q') \in \Delta.\\
&\quad [\![e_1]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![r_1 \triangleleft q_I]\!] \neq \emptyset\\
&\quad \wedge\ [\![e_2]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![r_2 \triangleleft q]\!] \neq \emptyset\\
&\overset{(4)}{\iff} [\![b\langle e_1\rangle e_2]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!] \neq \emptyset
\end{aligned}
$$

where each of the deformations is obtained as follows.

(1) By the definition of the function Inf.

(2) By the definition of the semantics of MFT.

(3) By the induction hypothesis for the right-hand-side expressions, e_1 and e_2.

(4) ($\Rightarrow$) We consider a forest $b\langle f_1\rangle f_2$ with $f_1 \in ([\![e_1]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![r_1 \triangleleft q_I]\!])$ and $f_2 \in ([\![e_2]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![r_2 \triangleleft q]\!])$. From $f_1 \in [\![e_1]\!](\boldsymbol{f}, \boldsymbol{\tau})$ and $f_2 \in [\![e_2]\!](\boldsymbol{f}, \boldsymbol{\tau})$, we obtain $b\langle f_1\rangle f_2 \in [\![b\langle e_1\rangle e_2]\!](\boldsymbol{f}, \boldsymbol{\tau})$ by the definition of the semantics $[\![\cdot]\!]$ of a right-hand-side expression. From $f_1 \in [\![r_1 \triangleleft q_I]\!]$, $f_2 \in [\![r_2 \triangleleft q]\!]$ and the transition rule $b(r_1, r_2) \to q'$, we obtain $b\langle f_1\rangle f_2 \in [\![q' \triangleleft q]\!]$ by Theorem 4.4.

($\Leftarrow$) We consider a forest $f' \in ([\![b\langle e_1\rangle e_2]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!])$. From $f' \in [\![b\langle e_1\rangle e_2]\!](\boldsymbol{f}, \boldsymbol{\tau})$, there exist $f'_1 \in [\![e_1]\!](\boldsymbol{f}, \boldsymbol{\tau})$ and $f'_2 \in [\![e_2]\!](\boldsymbol{f}, \boldsymbol{\tau})$ and we obtain $f' = b\langle f'_1\rangle f'_2$ by the definition of semantics $[\![\cdot]\!]$ of a right-hand-side expression. From $f' = b\langle f'_1\rangle f'_2 \in [\![q' \triangleleft q]\!]$, there exists a transition rule $b(r_1, r_2) \to q'$ with some states $r_1, r_2 \in Q$ such that $f'_1 \in [\![r_1 \triangleleft q_I]\!]$ and $f'_2 \in [\![r_2 \triangleleft q]\!]$ by Theorem 4.4. Thus, we have $\exists r_1, \exists r_2 \in Q.\, \exists (b(r_1, r_2) \to q').\, f'_1 \in ([\![e_1]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![r_1 \triangleleft q_I]\!]) \wedge f'_2 \in ([\![e_2]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![r_2 \triangleleft q]\!])$.

Consider the case $e = \varepsilon$. For any $\boldsymbol{\tau} \in {\mathcal{F}_\Sigma}^k$, we have

$$
\begin{aligned}
&\exists \boldsymbol{s'}, \exists \boldsymbol{s}.\, \boldsymbol{\tau} \in [\![\boldsymbol{s'} \triangleleft \boldsymbol{s}]\!] \ \wedge\ \boldsymbol{f} \in [\![\mathsf{Inf}(\varepsilon, (q', q), (\boldsymbol{s'}, \boldsymbol{s}))]\!]\\
&\overset{(1)}{\iff} \exists \boldsymbol{s'}, \exists \boldsymbol{s}.\, \boldsymbol{\tau} \in [\![\boldsymbol{s'} \triangleleft \boldsymbol{s}]\!] \ \wedge\ q' = q\\
&\overset{(2)}{\iff} [\![\varepsilon]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!] \neq \emptyset
\end{aligned}
$$

where each of the deformations is obtained as follows.

(1) By the definition of the function Inf.

(2) ($\Rightarrow$) We consider the forest ε. We obtain $\varepsilon \in [\![\varepsilon]\!](\boldsymbol{f}, \boldsymbol{\tau})$ by the definition of the semantics $[\![\cdot]\!]$ of a right-hand-side expression. From $q' = q$, we obtain $\varepsilon \in [\![q' \triangleleft q]\!]$ by the definition of the language of a hole-relational state of a TA. Thus, we have $\varepsilon \in ([\![\varepsilon]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!])$.

($\Leftarrow$) We consider a forest $f' \in ([\![\varepsilon]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!])$. From $f' \in [\![\varepsilon]\!](\boldsymbol{f}, \boldsymbol{\tau})$, we obtain $f' = \varepsilon$ by the definition of the semantics of a right-hand-side expression. From $f' = \varepsilon \in [\![q' \triangleleft q]\!]$, we obtain $q' = q$ by the definition of the language of a hole-relational state and the definition of a DBTA. In addition, since M is a DBTA, there exist $\boldsymbol{s'}$ and $\boldsymbol{s}$ such that $\boldsymbol{\tau} \in [\![\boldsymbol{s'} \triangleleft \boldsymbol{s}]\!]$ by Theorem 4.5.

Consider the case $e = p^{(l)}(x_h, e_1, \ldots, e_l)$. Since there is no expression for $n = 0$ in any right-hand-side expression, we consider only the case $n = 2$ i.e. $h \in [2]$. For any $\boldsymbol{\tau} \in {\mathcal{F}_\Sigma}^k$, we have

$$
\begin{aligned}
&\exists \boldsymbol{s'}, \exists \boldsymbol{s}.\, \boldsymbol{\tau} \in [\![\boldsymbol{s'} \triangleleft \boldsymbol{s}]\!]\\
&\quad \wedge\ \boldsymbol{f} \in [\![\mathsf{Inf}(p^{(l)}(x_h, e_1, \ldots, e_l), (q', q), (\boldsymbol{s'}, \boldsymbol{s}))]\!]\\
&\overset{(1)}{\iff} \exists \boldsymbol{s'}, \exists \boldsymbol{s}.\, \boldsymbol{\tau} \in [\![\boldsymbol{s'} \triangleleft \boldsymbol{s}]\!]\\
&\quad \wedge\ \boldsymbol{f} \in \left[\!\!\left[\bigvee_{\boldsymbol{t'}, \boldsymbol{t} \in Q^l} \left(\downarrow_h \langle p^{(l)}, (q', q), (\boldsymbol{t'}, \boldsymbol{t})\rangle \right.\right.\right.\\
&\qquad\qquad \left.\left.\left. \wedge \bigwedge_{j \in [l]} \mathsf{Inf}(e_j, (t'_j, t_j), (\boldsymbol{s'}, \boldsymbol{s})) \right) \right]\!\!\right]\\
&\overset{(2)}{\iff} \exists \boldsymbol{s'}, \exists \boldsymbol{s}.\, \boldsymbol{\tau} \in [\![\boldsymbol{s'} \triangleleft \boldsymbol{s}]\!]\\
&\quad \wedge \left(\exists \boldsymbol{t'}, \exists \boldsymbol{t} \in Q^l.\, f_h \in [\![\langle p^{(l)}, (q', q), (\boldsymbol{t'}, \boldsymbol{t})\rangle]\!] \right.\\
&\qquad \left. \wedge \bigwedge_{j \in [l]} \boldsymbol{f} \in [\![\mathsf{Inf}(e_j, (t'_j, t_j), (\boldsymbol{s'}, \boldsymbol{s}))]\!] \right)\\
&\overset{(3)}{\iff} \exists \boldsymbol{t'}, \exists \boldsymbol{t} \in Q^l.\, f_h \in [\![\langle p^{(l)}, (q', q), (\boldsymbol{t'}, \boldsymbol{t})\rangle]\!]\\
&\quad \wedge \bigwedge_{j \in [l]} [\![e_j]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![t'_j \triangleleft t_j]\!] \neq \emptyset\\
&\overset{(4)}{\iff} \exists \boldsymbol{t'}, \exists \boldsymbol{t} \in Q^l.\, (f_h \in [\![\langle p^{(l)}, (q', q), (\boldsymbol{t'}, \boldsymbol{t})\rangle]\!])\\
&\quad \wedge\ (\exists \boldsymbol{\tau'}.\, \forall j \in [l].\, \tau'_j \in [\![e_j]\!](\boldsymbol{f}, \boldsymbol{\tau}) \ \wedge\ \tau'_j \in [\![t'_j \triangleleft t_j]\!])
\end{aligned}
$$

$$\overset{(5)}{\iff} \exists \boldsymbol{\tau}'. [\![p^{(l)}]\!](f_h, \boldsymbol{\tau}') \cap [\![q' \triangleleft q]\!] \neq \emptyset \\ \wedge\ (\forall j \in [l].\ \tau'_j \in [\![e_j]\!](\boldsymbol{f}, \boldsymbol{\tau}))$$

$$\overset{(6)}{\iff} [\![p^{(l)}(x_h, e_1, \ldots, e_l)]\!] \cap [\![q' \triangleleft q]\!] \neq \emptyset$$

where each of the deformations is obtained as follows.

(1) By the definition of the function Inf.
(2) By the definition of the semantics $[\![\cdot]\!]$ of MFT.
(3) By the induction hypothesis for the right-hand-side expression e_j.
(4) By introduction $\boldsymbol{\tau}'$ such that $\tau'_j \in [\![e_j]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![t'_j \triangleleft t_j]\!]$.
(5) By the hypothesis H.
(6) By the definition of the semantics $[\![\cdot]\!]$ of a right-hand-side expression.

Consider the case $e = y_i$. For any $\boldsymbol{\tau} \in {\mathcal{F}_\Sigma}^k$, we have

$$\exists \boldsymbol{s}', \exists \boldsymbol{s}. \boldsymbol{\tau} \in [\![\boldsymbol{s}' \triangleleft \boldsymbol{s}]\!] \wedge \boldsymbol{f} \in [\![\mathsf{Inf}(y_i, (q', q), (\boldsymbol{s}', \boldsymbol{s}))]\!]$$

$$\overset{(1)}{\iff} \exists \boldsymbol{s}', \exists \boldsymbol{s}. \boldsymbol{\tau} \in [\![\boldsymbol{s}' \triangleleft \boldsymbol{s}]\!] \wedge q' = s'_i \wedge q = s_i$$

$$\overset{(2)}{\iff} \tau_i \in [\![q' \triangleleft q]\!]$$

$$\overset{(3)}{\iff} [\![y_i]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!] \neq \emptyset$$

where each of the deformations is obtained as follows.

(1) By the definition of the function Inf.
(2) ($\Rightarrow$) By replacing s'_i and s_i in the hypothesis $\tau_i \in [\![s'_i \triangleleft s_i]\!]$ with q' and q, respectively.
($\Leftarrow$) Since M is a DBTA, by Theorem 4.5, there exist $\boldsymbol{t}'$ and $\boldsymbol{t}$ such that $\boldsymbol{\tau} \in [\![\boldsymbol{t}' \triangleleft \boldsymbol{t}]\!]$. From $\tau_i \in [\![q' \triangleleft q]\!]$, we obtain $\boldsymbol{\tau} \in [\![\boldsymbol{s}' \triangleleft \boldsymbol{s}]\!]$, $q' = s'_i$, and $q = s_i$ where $\boldsymbol{s}' = (t'_1, \ldots, t'_{i-1}, q', t'_{i+1}, \ldots, t'_k)$ and $\boldsymbol{s} = (t_1, \ldots, t_{i-1}, q, t_{i+1}, \ldots, t_k)$
(3) By the definition of the semantics of a right-hand-side expression.

Consider the case $e = e_1 e_2$. For any $\boldsymbol{\tau} \in {\mathcal{F}_\Sigma}^k$, we have

$$\exists \boldsymbol{s}', \exists \boldsymbol{s}. \boldsymbol{\tau} \in [\![\boldsymbol{s}' \triangleleft \boldsymbol{s}]\!] \\ \wedge \boldsymbol{f} \in [\![\mathsf{Inf}(e_1 e_2, (q', q), (\boldsymbol{s}', \boldsymbol{s}))]\!]$$

$$\overset{(1)}{\iff} \exists \boldsymbol{s}', \exists \boldsymbol{s}. \boldsymbol{\tau} \in [\![\boldsymbol{s}' \triangleleft \boldsymbol{s}]\!] \\ \wedge \boldsymbol{f} \in \left[\!\!\left[\bigvee_{r \in Q} \mathsf{Inf}(e_1, (q', r), (\boldsymbol{s}', \boldsymbol{s})) \right.\right. \\ \left.\left. \wedge\ \mathsf{Inf}(e_2, (r, q), (\boldsymbol{s}', \boldsymbol{s})) \right]\!\!\right]$$

$$\overset{(2)}{\iff} \exists \boldsymbol{s}', \exists \boldsymbol{s}. \boldsymbol{\tau} \in [\![\boldsymbol{s}' \triangleleft \boldsymbol{s}]\!] \\ \wedge\ (\exists r \in Q. \boldsymbol{f} \in [\![\mathsf{Inf}(e_1, (q', r), (\boldsymbol{s}', \boldsymbol{s}))]\!] \\ \wedge \boldsymbol{f} \in [\![\mathsf{Inf}(e_2, (r, q), (\boldsymbol{s}', \boldsymbol{s}))]\!])$$

$$\overset{(3)}{\iff} \exists r \in Q. [\![e_1]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![q' \triangleleft r]\!] \neq \emptyset \\ \wedge\ [\![e_2]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![r \triangleleft q]\!] \neq \emptyset$$

$$\overset{(4)}{\iff} [\![e_1 e_2]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!] \neq \emptyset$$

where each of the deformations is obtained as follows.

(1) By the definition of the function Inf.
(2) By the definition of the semantics $[\![\cdot]\!]$ of MFT.
(3) By the induction hypothesis for right-hand-side expressions, e_1 and e_2.
(4) ($\Rightarrow$) We consider a forest $f'_1 f'_2$ with $f'_1 \in ([\![e_1]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![q' \triangleleft r]\!])$ and $f'_2 \in ([\![e_2]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![r \triangleleft q]\!])$. From $f'_1 \in [\![e_1]\!](\boldsymbol{f}, \boldsymbol{\tau})$ and $f'_2 \in [\![e_2]\!](\boldsymbol{f}, \boldsymbol{\tau})$, we obtain $f'_1 f'_2 \in [\![e_1 e_2]\!](\boldsymbol{f}, \boldsymbol{\tau})$ by the definition of the semantics $[\![\cdot]\!]$ of a right-hand-side expression. In addition, From $f'_1 \in [\![q' \triangleleft r]\!]$ and $f'_2 \in [\![r \triangleleft q]\!]$, we obtain $f'_1 f'_2 \in [\![q' \triangleleft q]\!]$ by the definition of the language of a hole-relational state of a TA and the definition of a concatenation for forests. Thus, we have $f'_1 f'_2 \in ([\![e_1 e_2]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!])$.
($\Leftarrow$) We consider a forest $f' \in ([\![e_1 e_2]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!])$. From $f' \in [\![e_1 e_2]\!](\boldsymbol{f}, \boldsymbol{\tau})$, there exist $f'_1 \in [\![e_1]\!](\boldsymbol{f}, \boldsymbol{\tau})$ and $f'_2 \in [\![e_2]\!](\boldsymbol{f}, \boldsymbol{\tau})$ and we obtain $f' = f'_1 f'_2$ by the definition of the semantics $[\![\cdot]\!]$ of a right-hand-side expression. In addition, from $f' = f'_1 f'_2 \in [\![q' \triangleleft q]\!]$, there exists $r \in Q$ such that $f'_1 \in [\![q' \triangleleft r]\!]$ and $f'_2 \in [\![r \triangleleft q]\!]$. Thus, we have $\exists r \in Q. f'_1 \in ([\![e_1]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![q' \triangleleft r]\!]) \wedge f'_2 \in ([\![e_2]\!](\boldsymbol{f}, \boldsymbol{\tau}) \cap [\![r \triangleleft q]\!])$.

Based on the above, Lemma 4.7 is proved by induction of the structure of the right-hand-side expression e. □

We obtain Lemma 4.8 with Lemma 4.7.

Lemma 4.8. Let $\mathcal{T} = (P, P_0, \Pi)$ and $M = (Q, \{q_I\}, F, \Delta)$ be an MFT and a DBTA, respectively. For states in the ATA $\mathrm{Iv}(\mathcal{T}, M) = (\Xi, \Xi_I, \Xi_F, \Phi)$,

$$\forall f \in \mathcal{F}_\Sigma. \forall p^{(k)} \in P. \forall \boldsymbol{\tau} \in {\mathcal{F}_\Sigma}^k. \forall q', q \in Q. \\ (\exists \boldsymbol{s}', \exists \boldsymbol{s} \in Q^k. \boldsymbol{\tau} \in [\![\boldsymbol{s}' \triangleleft \boldsymbol{s}]\!] \wedge f \in [\![\langle p^{(k)}, (q', q), (\boldsymbol{s}', \boldsymbol{s})\rangle]\!]) \\ \iff [\![p^{(k)}]\!](f, \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!] \neq \emptyset$$

holds. ■

Proof. By induction on the structure of the forest f.

Consider the case $f = \varepsilon$. For any $p^{(k)} \in P$, any $\boldsymbol{\tau} \in {\mathcal{F}_\Sigma}^k$ and any $q', q \in Q$, we have

$$\exists \boldsymbol{s}', \exists \boldsymbol{s}. \boldsymbol{\tau} \in [\![\boldsymbol{s}' \triangleleft \boldsymbol{s}]\!] \wedge \varepsilon \in [\![\langle p^{(k)}, (q', q), (\boldsymbol{s}', \boldsymbol{s})\rangle]\!]$$

$\iff$ {The definition of the language of an ATA}

$$\exists \boldsymbol{s}', \exists \boldsymbol{s}. \boldsymbol{\tau} \in [\![\boldsymbol{s}' \triangleleft \boldsymbol{s}]\!] \wedge \langle p^{(k)}, (q', q), (\boldsymbol{s}', \boldsymbol{s})\rangle \in \Xi_I$$

$\iff$ {The definition of Ξ_I in Iv}

$$\exists \boldsymbol{s}', \exists \boldsymbol{s}. \boldsymbol{\tau} \in [\![\boldsymbol{s}' \triangleleft \boldsymbol{s}]\!] \\ \wedge\ \exists (p^{(k)}(\varepsilon, y_1, \ldots, y_k) \to e) \in \Pi. \\ () \in [\![\mathsf{Inf}(e, (q', q), (\boldsymbol{s}', \boldsymbol{s}))]\!]$$

$\iff$ {Lemma 4.7}

$$\exists (p^{(k)}(\varepsilon, y_1, \ldots, y_k) \to e) \in \Pi. \\ [\![e]\!](), \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!] \neq \emptyset$$

$\iff$ {The definition of the semantics $[\![\cdot]\!]$ of MFT}

$$[\![p^{(k)}]\!](\varepsilon, \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!] \neq \emptyset.$$

Consider the case $f = a\langle f_1\rangle f_2$. For any $p^{(k)} \in P$, any $\boldsymbol{\tau} \in {\mathcal{F}_\Sigma}^k$, and any $q', q \in Q$, we have

$$\exists \boldsymbol{s}', \exists \boldsymbol{s}. \boldsymbol{\tau} \in [\![\boldsymbol{s}' \triangleleft \boldsymbol{s}]\!] \\ \wedge\ a\langle f_1\rangle f_2 \in [\![\langle p^{(k)}, (q', q), (\boldsymbol{s}', \boldsymbol{s})\rangle]\!]$$

$\iff$ {The definition of the language of an ATA}

$$
\begin{aligned}
&\exists \boldsymbol{s}', \exists \boldsymbol{s}.\ \boldsymbol{\tau} \in [\![\boldsymbol{s}' \triangleleft \boldsymbol{s}]\!] \\
&\quad \wedge\ (f_1, f_2) \in [\![\Phi(\langle p^{(k)}, (q', q), (\boldsymbol{s}', \boldsymbol{s})\rangle, a)]\!] \\
\Longleftrightarrow\ & \quad \{\text{The definition of } \Phi \text{ in Iv}\} \\
&\exists \boldsymbol{s}', \exists \boldsymbol{s}.\ \boldsymbol{\tau} \in [\![\boldsymbol{s}' \triangleleft \boldsymbol{s}]\!] \\
&\quad \wedge\ \exists (p^{(k)}(a\langle f_1\rangle f_2, y_1, \ldots, y_k) \to e) \in \Pi. \\
&\qquad (f_1, f_2) \in [\![\mathsf{Inf}(e, (q', q), (\boldsymbol{s}', \boldsymbol{s}))]\!] \\
\Longleftrightarrow\ & \quad \{\text{The induction hypothesis for } f_1 \text{ and } f_2 \\
&\qquad \text{and Lemma 4.7}\} \\
&\exists (p^{(k)}(a\langle f_1\rangle f_2, y_1, \ldots, y_k) \to e) \in \Pi. \\
&[\![e]\!]((f_1, f_2), \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!] \neq \emptyset \\
\Longleftrightarrow\ & \quad \{\text{The definition of the semantics } [\![\cdot]\!] \text{ of MFT}\} \\
&[\![p^{(k)}]\!](a\langle f_1\rangle f_2, \boldsymbol{\tau}) \cap [\![q' \triangleleft q]\!] \neq \emptyset
\end{aligned}
$$

Thus, Lemma 4.8 is proved by induction on the structure of f. □

This lemma provides that the ATA constructed by Iv accepts the inverse image.

Theorem 4.9. Let $\mathcal{T} = (P, P_0, \Pi)$ and $M = (Q, \{q_I\}, F, \Delta)$ be an MFT and a DBTA, respectively. For the ATA $\mathrm{Iv}(\mathcal{T}, M)$,

$$\mathcal{L}(\mathrm{Iv}(\mathcal{T}, M)) = \mathcal{T}^{-1}(\mathcal{L}(M))$$

holds. ■

Proof. We find

$$
\begin{aligned}
&\mathcal{L}(\mathrm{Iv}(\mathcal{T}, M)) \\
&= \quad \{\text{The definition of the language of an ATA}\} \\
&\quad \bigcup_{p_0{}^{(0)} \in P_0,\, q_F \in F} [\![\langle p_0{}^{(0)}, (q_F, q_I), ((), ())\rangle]\!] \\
&= \quad \{\text{Lemma 4.8}\} \\
&\quad \{f \mid [\![p_0{}^{(0)}]\!](f, ()) \cap [\![q_F \triangleleft q_I]\!] \neq \emptyset,\ p_0{}^{(0)} \in P_0,\ q_F \in F\} \\
&= \quad \{\text{The definition of the language of a hole-relational} \\
&\qquad \text{state and the language of a state}\} \\
&\quad \{f \mid [\![p_0{}^{(0)}]\!](f, ()) \cap [\![q_F]\!] \neq \emptyset,\ p_0{}^{(0)} \in P_0,\ q_F \in F\} \\
&= \quad \{\text{The definitions of the semantics of MFT}\} \\
&\quad \{f \mid \mathcal{T}(f) \cap \mathcal{L}(M) \neq \emptyset,\ p_0{}^{(0)} \in P_0,\ q_F \in F\} \\
&= \quad \{\text{The definitions of an inverse image} \\
&\qquad \text{and the language of a TA}\} \\
&\quad \mathcal{T}^{-1}(\mathcal{L}(M)).
\end{aligned}
$$

□

4.3 Typechecking Method for MFTs

In this section, we show a typechecking method for MFTs using the inverse type inference method with an ATA. Typechecking for an MFT $\mathcal{T}$, an input type TA M_{in}, and an output type DBTA M_{out} is performed as follows.

(1) We obtain a TA $M_{\mathrm{out}}^{\complement}$ such that $\mathcal{L}(M_{\mathrm{out}}^{\complement}) = \mathcal{L}(M_{\mathrm{out}})^{\complement}$ by the construction method of a complement TA of a DBTA.

(2) For $\mathcal{T}$ and $M_{\mathrm{out}}^{\complement}$, we obtain an ATA A such that $\mathcal{L}(A) = \mathcal{T}^{-1}(\mathcal{L}(M_{\mathrm{out}}^{\complement}))$ by Iv.

(3) We obtain an ATA A_{in} such that $\mathcal{L}(A_{\mathrm{in}}) = \mathcal{L}(M_{\mathrm{in}})$ by the transformation method of a TA into an ATA.

(4) We obtain an ATA A' such that $\mathcal{L}(A') = \mathcal{L}(A) \cap \mathcal{L}(A_{\mathrm{in}})$ by the construction method of an intersection of ATAs.

(5) We check whether the language of A' is empty.

Here, the result of the emptiness test of A' can be used as the result of typechecking as required.

Corollary 4.10. For an ATA A' constructed as above, $\mathcal{L}(A') = \emptyset \iff \mathcal{T}(\mathcal{L}(M_{\mathrm{in}})) \subseteq \mathcal{L}(M_{\mathrm{out}})$ holds. ■

Proof. From the construction method of A',

$$\mathcal{L}(A') = \mathcal{T}^{-1}(\mathcal{L}(M_{\mathrm{out}})^{\complement}) \cap \mathcal{L}(M_{\mathrm{in}})$$

is obtained. In addition, by Lemma 2.20,

$$\mathcal{L}(A') = \emptyset \iff \mathcal{T}(\mathcal{L}(M_{\mathrm{in}})) \subseteq \mathcal{L}(M_{\mathrm{out}})$$

holds. □

4.4 Computational Complexity of Our Typechecking Method

We show the complexity of our typechecking method in Section 4.3 for an MFT $\mathcal{T}$, an output type DBTA M_{out}, and an input type TA M_{in}, following steps (1) to (5) of the method. Let $|P|$, $|e|$, and k be the number of function names, the maximum size and the maximum rank of the MFT $\mathcal{T}$, respectively. In addition, let $|Q_{\mathrm{out}}|$ be the number of states of the DBTA M_{out} and $|Q_{\mathrm{in}}|$ and $|\Delta_{\mathrm{in}}|$ be the numbers of states and transition rules of the TA M_{in}, respectively. Notice that since M_{out} is a DBTA, the number of transition rules is $O(|Q_{\mathrm{out}}|^2)$.

(1) For a DBTA M, a DBTA that is the complement of M can be computed in linear time for the number of states of M according to the construction. In addition, the number of states of M' is equal to the number of states of M. Therefore, (1) can be computed in linear time for $|Q_{\mathrm{out}}|$, and the number of states of the DBTA $M_{\mathrm{out}}^{\complement}$ is $|Q_{\mathrm{out}}|$.

(2) We consider the construction time of the ATA $A = \mathrm{Iv}(\mathcal{T}, M_{\mathrm{out}}^{\complement})$ for the DBTA $M_{\mathrm{out}}^{\complement}$ constructed at (1). One calculation time of $\Phi(q, a)$ is $O(k|Q_{\mathrm{out}}|^{2k|e|})$ because the number of states of $M_{\mathrm{out}}^{\complement}$ is $O(|Q_{\mathrm{out}}|^2)$. In addition, the number of states of A is $O(|P|\,|Q_{\mathrm{out}}|^{2(k+1)})$. Therefore, (2) can be computed in time $O(k|P|\,|Q_{\mathrm{out}}|^{4k(k+1)|e|})$.

(3) For a TA M that has $|Q|$ states and $|\Delta|$ transition rules, a transformation from M to an ATA can be done in time $O(|Q_{\mathrm{in}}| + |\Delta_{\mathrm{in}}|)$, and then the ATA has $|Q|$ states. Therefore, the ATA A_{in} can be computed in time $O(|Q_{\mathrm{in}}| + |\Delta_{\mathrm{in}}|)$ and has $|Q_{\mathrm{in}}|$ states.

(4) A construction of an ATA that accepts the intersection of ATAs can be computed in linear time for the numbers of states of original ATAs, according to the construction. Therefore, the ATA A' can be computed in time $O(|P|\,|Q_{\mathrm{out}}|^{2(1+k)} + |Q_{\mathrm{in}}|)$ and has $O(|P|\,|Q_{\mathrm{out}}|^{2(1+k)} + |Q_{\mathrm{in}}|)$ states.

(5) An emptiness test for an ATA can be computed in exponential time [11]. Therefore, (5) can be computed in exponential time for $|P| + |Q_{\mathrm{out}}| + |Q_{\mathrm{in}}|$.

In summary, since the complexities (1) to (4) can be neglected

for (5), the typechecking method can be computed in exponential time for $|P| + |Q_{\text{out}}| + |Q_{\text{in}}|$.

5. Optimization and Implementation

In the inverse type inference method of Frisch and Hosoya that we extend, they proposed an optimization technique [6] that shortens conditional expressions obtained from the transition function of an ATA constructed by the inverse type inference method and then implemented a typechecker for MTTs [8] using this optimization. To use their optimization simply, we modified their implementation on the basis of our inverse type inference method for MFTs so as to obtain a typechecker for MFTs equipped with their optimization. Because of the optimization, our implementation computes a conditional expression $\mathsf{Inf}(e, (\overline{q}, \overline{q'}), (s', s))$ for sets $\overline{q}$ and $\overline{q'}$ of states instead of states q and q'. The conditional expression $\mathsf{Inf}(e, (\overline{q}, \overline{q'}), (s', s))$ represents $\bigvee_{q' \in \overline{q'},\, q \in \overline{q}} \mathsf{Inf}(e, (q', q), (s', s))$. Since we use a set $\overline{q}$ of states instead of a state q, the number of states of a constructed ATA grows exponentially. However, their optimization shortens conditional expressions obtained from Inf, and then typechecking can be performed faster in practice.

6. Evaluation

In this section, we evaluate and consider our typechecking implementation by comparing it with other typechecking implementations. The implementation of Frisch [8] and our implementation that extends his were compiled by OCaml 4.04.0, respectively. In addition, these implementations were run on an Intel Core i5-4590 PC with 4 GB RAM.

The details of typechecking examples for MFTs used in the evaluation were defined as follows. We used DTD to represent the input and output types for simplicity. A line `type Input = a[T1],T2` declares and defines an element `Input` in which a tree has the symbol `a` at the root, the child in the element `T1` and the sibling in the element `T2`. In addition, a leaf ε represented by `()` can be omitted. For example, $a\langle\varepsilon\rangle\varepsilon$ is represented by `a[()]()` or `a[]`.

appT, appF

These are MFTs that represent XML transformations used in evaluations of Kobayashi et al. [9] and Frisch and Hosoya [6]. A given input forest is a document with `doc` as the root. In particular, an input type is represented by the following DTD.

```
type Input   = doc[Preface, (Div|P|Note)*]
type Preface = preface[Header, P*]
type Header  = header[]
type P       = p[]
type Div     = div[(Div|P|Note)*]
type Note    = note[P*]
```

appT and **appF** append `appendix` to the child of `doc`, append the children of `preface` and `note` to the child of `appendix`, and eliminate the children of `preface` and `note`. Typechecking for these MFTs verify the correctness of transformations for input forests. For typechecking, the output type is defined as follows.

```
type Output   = doc[(Div|P)*, Appendix]
type P        = p[]
type Div      = div[(Div|P)*]
type Appendix = appendix[Header, P*]
type Header   = header[]
```

where **appT** is defined without concatenationas an MTT, and **appF** is defined with concatenation as an MFT so that its function names have smaller ranks. In this way, these implementations are different.

echild

A given DTD as an input type is the same as for the **appT** case. **echild**, for example, transforms a forest `a[b[]],c[]` into a forest `a[b[empty[]]],c[empty[]]` using a forest `empty[]` that evinces the emptiness of the child. Typechecking for **echild** verifies the correctness of transformations for input forests. For typechecking, the output type is defined as follows.

```
type Output  = doc[Preface, (Div|P|Note)*]
type Preface = preface[Header, P*]
type Header  = header[Empty]
type P       = p[Empty]
type Div     = div[(Empty|(Div|P|Note)+)]
type Note    = note[(Empty|P+)]
type Empty   = empty[]
```

fib

fib is an MFT that uses concatenation and is based on Faßbender et al.'s [12] top-down tree transducer [13] to obtain Fibonacci numbers. Given a forest representing n, **fib** transforms into an output forest representing the n-th Fibonacci number. An input forest has a sequence of `s` in the child direction and represents a natural number including zero. For example, an input forest $\mathtt{s}\langle\mathtt{s}\langle\varepsilon\rangle\varepsilon\rangle\varepsilon$ represents 2. An output forest has a sequence of `s` in the sibling direction. For example, an output forest $\mathtt{s}\langle\varepsilon\rangle\mathtt{s}\langle\varepsilon\rangle\varepsilon$ represents 2. It is known that a $3n$-th Fibonacci number is even. Typechecking for **fib** verifies this fact.

book

book is the MFT based on an example of an MFT created by Nakano [14]. An input forest represents information of a book whose root is `book`. In particular, an input type for **book** is defined as follows.

```
type Input   = book[Title, Chapter*]
type Title   = title[]
type Chapter = chapter[Title, Item*]
type Item    = key[] | word[]
```

book adds a serial section number represented by a successor `s` and a zero `z` to the child of `chapter`, transforms `title` into `name` under an element `Chapter`, and enumerates the keywords `key` in the child of `book`. Typechecking for **book** verifies the correctness of transformations for input forests. For typechecking, the output type is defined as follows.

```
type Output  = book[Title, Chapter*, Index]
type Title   = title[]
```

Table 1 Execution times of typechecking by our implementation and Frisch's implementation.

	appT	appF	echild	book	fib
MFT/MTT	MTT	MFT	MTT	MFT	MFT
Frisch's implementation (ms)	11.95	213.0	3.116	4996	N/A
Our implementation (ms)	32.95	9.702	3.110	17.19	1071
No. of states of the ATA in Frisch's implementation	65	154	13	378	N/A
No. of states of the ATA in our implementation	84	50	13	83	33

```
type Chapter = chapter[Num, Name, Item*]
type Num     = s[Num] | z[]
type Name    = name[]
type Item    = key[] | word[]
```

We compared our implementation with Frisch's implementation with regard to execution times of typechecking for MFTs and MTTs defined above. In addition, MFTs are more expressive than MTTs, but a composition of two MTTs is more expressive than an MFT [2]. Based on this fact, we represent an MFT by a composition of two MTTs in Frisch's implementation. The results were as shown in **Table 1**. In this table, the number of states of an ATA is the number of states actually constructed in inverse type inference, not the number of all states of the ATA constructed by Iv.

For these typechecking examples, our implementation executed in reasonable time. In addition, we found out the following facts.

The number of states of an ATA The number of states of an ATA significantly affected the execution time of typechecking. The case concerned with this fact was considered to have the worst-case complexity of the typechecking algorithm, which is exponential to the number of states.

Concatenation of forests In our implementation, typechecking for **appF** generated fewer states of an ATA than typechecking for **appT**. **appF** was defined as the same MFT of **appT** with concatenation of forests. The number of states of an ATA constructed by our inverse type inference increases with the maximum rank of function names of an MFT significantly. A concatenation expression often deducts functions and decreases the maximum rank of functions. As a result, a concatenation like above seemed to decrease the number of states of the ATA significantly and improve the execution times of typechecking. In other words, even if a transformation can be represented as an MTT, we can improve the performance of typechecking by describing it in an MFT with concatenation expressions.

Comparison with Frisch's implementation In typechecking for MTTs and in particular for **appT**, our implementation made more states of an ATA and took a long time for typechecking. Our implementation treats pairs of states instead of states to be specifically designed for MFTs. As a result, our implementation seemed to consider more needless states and to cause worse results than above. On the other hand, in typechecking for MFTs of **appF**, **book**, and **fib**, our implementation made fewer states of an ATA and took a shorter time for typechecking. In particular, typechecking for **fib** in Frisch's implementation did not end within ten minutes. In Frisch's implementation, since an MFT is represented by a composition of MTTs, this result seemed to be caused by two-fold computation of inverse type inference. In our implementation, since inverse type inference for MFTs is direct, there is no inverse image in the middle. Based on the above, our implementation seems to specialize in typechecking for MFTs.

7. Related Work

7.1 Typechecking for MTT with ATA

A typechecking method for MTTs with an ATA was proposed by Frisch and Hosoya [6] and was implemented [8]. It is known that an MFT can be represented by the composition of two MTTs [2]. We can convert any MFT $\mathcal{T}$ into an MTT by replacing a concatenation $f_1 f_2$ with $@\langle f_1 \rangle f_2$ using the special symbol $@$. In addition, we can construct an MTT $\mathcal{T}_@$ that transforms $@\langle f_1 \rangle f_2$ into $f_1 f_2$. Therefore, assuming that $\mathcal{T}'$ is an MTT with $@$ converted from an MFT $\mathcal{T}$, we can express $\mathcal{T}$ as a composition of MTTs such that $\mathcal{T}(f) = \mathcal{T}_@(\mathcal{T}'(f))$ for any forest f. Based on this fact, their inverse type inference method for MTTs as it stands provides inverse type inference for MFTs. However, in this method, typechecking for $\mathcal{T}_@$ has an ATA accepting the inverse image under $\mathcal{T}'$ as an output type and then increases the number of states of an ATA accepting the inverse image under $\mathcal{T}$. Let $|Q|$ and $|P|$ be the numbers of states of a DBTA representing an output type and functions of an MFT, respectively, and let k be the maximum rank of functions of the MFT. In particular, the number of states of an ATA constructed is $O(\exp(2|P|\,|Q|^{1+k}))$.

In contrast, we extended Frisch and Hosoya's method to MFTs and proposed a direct typechecking method for MFTs. This method constructs $O(|P|\,|Q|^{2(1+k)})$ states of an ATA, that is, the number of states is reduced compared to their method. Actually, Section 6 shows the superiority of our method in typechecking for MFTs when comparing execution times.

7.2 Typechecking for Higher-order Multi-parameter Tree Transducers

Kobayashi et al. [9] proposed higher-order multi-parameter tree transducers and a forward typechecking method for them and implemented a typechecker. Their typechecking method is incomplete, but it is complete for linear higher-order multi-parameter tree transducers whose expression is restricted. By using continuation-passing style, they supposed each partially applied function of an MTT is an argument of a continuation and showed a method to convert from any MTT into a linear higher-order multi-parameter tree transducer. A linear higher-order multi-parameter tree transducer that was created by this method is 3-order because of (1) using continuation and (2) an argument of continuation is a partially applied function of an MTT. Similarly, supposing a forest is an unary function, an MFT

is converted into a 4-order linear height-order multi-parameter tree transducer and can be typechecked by their typechecking method. For a n-order higher-order multi-parameter tree transducer, let $|T|$, $|Q_{\text{in}}|$, and $|Q_{\text{out}}|$ be the size of the transducer, number of states of an input type TA and number of states of an output type TA, respectively. Their typechecking method runs in time $O(|T| \exp_n((|Q_{\text{out}}| + |Q_{\text{in}}|)^{1+\epsilon}))$ for any positive number ϵ and a function $\exp_n$ defined by $\exp_0(x) = x$, $\exp_n(x) = 2^{\exp_{n-1}(x)}$. Thus, for an MFT with $|P|$ functions, their typechecking method runs in time $O(|P| \exp_4((|Q_{\text{out}}| + |Q_{\text{in}}|)^{1+\epsilon}))$. In addition, this time complexity for a linear higher-order multi-parameter tree transducer converted from an MFT is refined into $O(|P| \exp_2((|Q_{\text{out}}| + |Q_{\text{in}}|)^{1+\epsilon}))$ because of (1) using each continuation only once and (2) considering only deterministic TAs as input and output types. In contrast, since our typechecking method runs in exponential time for $|P| + |Q_{\text{out}}| + |Q_{\text{in}}|$, it is superior to their method with regard to complexity with the number of states of TAs. However, their method runs in reasonable time despite high complexity. For example, their typechecker takes 19 ms, 12 ms, and 6 ms for typechecking for MFTs of **appF**, **book**, and **fib**, respectively.

8. Conclusion

In this paper, we extended an inverse type inference method for MTTs of Frisch and Hosoya to an inverse type inference method for MFTs with an ATA and showed the correctness of it. In addition, we showed a direct typechecking method for MFTs by this inverse type inference method and found that an implementation of it can typecheck in reasonable time.

Typechecking for more expressive tree transducers is our future work. *Macro tree transducers with Holes* (HMTTs) [15] are known as tree transducers that are more expressive than MFTs. While MFTs deal with forests, that is, tree-structured data with just a single 'hole', HMTTs deal with tree-structured data with an arbitrary number of holes. Concatenation of forests in MFTs is generalized by HMTTs as 'hole-application'. We expect that an ATA for an inverse image under an HMTT could be constructed similarly to the inverse type inference method for MFTs in this paper. Following our implementation technique, the emptiness of the ATA could be efficiently checked as shown in the experiments of Frisch and Hosoya [6]. Thereby, we might be able to implement a practical typechecker for HMTTs.

Acknowledgments We are grateful to Naoki Kobayashi for giving fruitful information about a typechecking method for higher-order multi-parameter tree transducers in Section 7.2. Further, we wish to express our gratitude to Alain Frisch for providing his source code of a typechecker for MTTs, and Hiroshi Unno for providing a typechecker for higher-order multi-parameter tree transducers. We also thank Hideya Iwasaki and anonymous reviewers for their valuable comments. This work was supported by JSPS KAKENHI Grant Number JP25730002 and JP17K00007.

References

[1] Engelfriet, J. and Vogler, H.: Macro tree transducers, *Journal of Computer and System Sciences*, Vol.31, No.1, pp.71–146, Academic Press (1985).
[2] Perst, T. and Seidl, H.: Macro forest transducers, *Information Processing Letters*, Vol.89, No.3, pp.141–149, Elsevier (2004).
[3] Nakano, K. and Mu, S.-C.: A pushdown machine for recursive XML processing, *Proc. 4th Asian Conference on Programming Languages and Systems* (*APLAS*), pp.340–356 (2006).
[4] Hakuta, S., Maneth, S., Nakano, K. and Iwasaki, H.: XQuery streaming by Forest Transducers, *Proc. IEEE 30th International Conference on Data Engineering* (*ICDE*), pp.952–963 (2014).
[5] Maneth, S., Berlea, A., Perst, T. and Seidl, H.: XML type checking with macro tree transducers, *Proc. 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (*PODS*), pp.283–294 (2005).
[6] Frisch, A. and Hosoya, H.: Towards practical typechecking for macro tree transducers, *Proc. 11th International Symposium on Database Programming Languages* (*DBPL*), pp.246–260 (2007).
[7] Slutzki, G.: Alternating tree automata, *Theoretical Computer Science*, Vol.41, pp.305–318, Elsevier (1985).
[8] Frisch, A.: Experiment in exact type-checking for XML transformations with accumulators, available from ⟨https://github.com/alainfrisch/mtt⟩.
[9] Kobayashi, N., Tabuchi, N. and Unno, H.: Higher-order multi-parameter tree transducers and recursion schemes for program verification, *Proc. 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (*POPL*), pp.495–508 (2010).
[10] Hosoya, H.: *Foundations of XML processing*, Cambridge University Press (2010).
[11] Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S. and Tommasi, M.: *Tree automata techniques and applications*, available from ⟨http://www.grappa.univ-lille3.fr/tata⟩ (2007).
[12] Faßbender, H. and Maneth, S.: A strict border for the decidability of E-unification for recursive functions, *Proc. 5th International Conference on Algebraic and Logic Programming* (*ALP*), pp.194–208 (1996).
[13] Rounds, W.C.: Mappings and grammars on trees, *Mathematical Systems Theory*, Vol.4, No.3, pp.257–287, Springer (1970).
[14] Nakano, K.: Automatic derivation of XML stream processors from macro forest transducers (in Japanese), *Proc. 22nd JSSST Annual Conference*, pp.1–11 (2005).
[15] Maneth, S. and Nakano, K.: XML type checking for macro tree transducers with holes, *Proc. International Workshop on Programming Language Technologies for XML* (*PLAN-X*) (2008).

Kazuhiro Abe was born in Tokyo, 1994. He received his B.E. degree in Informatics and Engineering from the University of Electro-Communications, Japan, in 2016. He has been a student in the Graduate School of Informatics and Engineering, the University of Electro-Communications since the same year. He is interested in functional programming languages, formal languages, and tree transducers.

Keisuke Nakano is an associate professor at the University of Electro-Communications. He received his Ph.D. degree from Kyoto University in 2006. He worked as a researcher at University of Tokyo from 2003 to 2008, and as an assistant professor at the present university from 2008 to 2012. His research interests include formal language theory, programming language theory, and functional programming. He is a member of ACM, IPSJ, and JSSST.