# Languages Modulo Normalization

Hitoshi Ohsaki[1] and Hiroyuki Seki[2]

[1] National Institute of Advanced Industrial Science and Technology
ohsaki@ni.aist.go.jp

[2] Nara Institute of Science and Technology
seki@is.naist.jp

**Abstract.** We propose a new class of tree automata, called *tree automata with normalization* (TAN). This framework extends equational tree automata, and improved the results of them: recognized tree languages modulo the *idempotency* $f(x, x) = x$ are closed under complement, besides we do not lose an important decidability result. In the paper, first we investigate the closure properties of this class for Boolean operations and the decidability relative to the equational tree automata. Next we consider the relationship to other automata frameworks, in particular, *hedge automata*, which is a class of unranked tree automata. Hedge automata have been recognized in the XML database community as a theoretical basis for modeling the manipulation of semi-structured data. Through the observation about transformations from XML schema to tree automata, we discuss advantages in the expressiveness and complexity of TAN. As an application of our framework, we show an example of XML schema and constraints that can be dealt with by TAN.

*Keywords*: tree automata modulo axioms, equational rewriting, Boolean closedness, decidability, regularity, XML schema and constraints.

## 1 Introduction

Tree automata accept trees as word automata accept words. The class of tree automata hence inherits the benefit from word automata that guarantees important bases for automated reasoning, including closure properties for Boolean operations and the positive decidability results. Propertywise there is no doubt in regular tree automata being a generalization of regular word automata. In fact, several verification tools are designed based on tree automata [1, 11, 12].

However, tree automata accept *irregular* words. For instance, the set of trees $t_{i+1} = \mathsf{f}(\mathsf{a}, \mathsf{f}(t_i, \mathsf{b}))$ and $t_1 = \mathsf{f}(\mathsf{a}, \mathsf{b})$ is regular in the sense of tree automata, while $\mathsf{a}^i\,\mathsf{b}^i$, which is the leaves of $t_i$, are no longer regular in words. This is a natural consequence from the fact that trees accepted by regular tree automata are the derivation trees of context-free grammar, and thus, Pumping Lemma [4] for regular tree automata can be viewed as a variant of $uvwxy$-lemma for context-free grammar, where for every context-free grammar $\mathcal{G}$, there exists a natural number $k$ such that, whenever a word $z$ whose length $|z|$ is greater than $k$ is generated by $\mathcal{G}$, $z$ can be decomposed to be $uvwxy$ containing non-empty words $v, x$ and for every $n \geqslant 0$, $uv^n wx^n y$ is generated by $\mathcal{G}$.

In the setting of equational tree automata, this becomes much more clear. We showed in [22] that the class of regular tree automata modulo associativity axioms is closely related to the class of context-free grammar. Moreover, *monotone* tree automata modulo associativity which are the super-class of regular tree automata modulo associativity are context-sensitive. These observations are obtained from the property that regular tree automata modulo associativity (for short, regular A-TA) are *not* closed under intersection and complement, and also from that the class of monotone tree automata modulo associativity (monotone A-TA) is *not* decidable in the emptiness, universality and inclusion problems. Furthermore, looking at the transition rules, regular rules $f(\alpha_1(x_1), \ldots, \alpha_n(x_n)) \to \beta(f(x_1, \ldots, x_n))$ in tree automata are the production rules $\beta \to \alpha_1 \cdots \alpha_n$ of context-free grammar, and monotone rules $f(\alpha_1(x_1), \ldots, \alpha_n(x_n)) \to f(\beta_1(x_1), \ldots, \beta_n(x_n))$ in tree automata are the rules $\beta_1 \cdots \beta_n \to \alpha_1 \cdots \alpha_n$ of context-sensitive grammar.

What should be then the counterpart of regular word languages in tree languages? In the paper, we introduce a new tree automata framework, called *tree automata with normalization*, that extends equational tree automata and generalizes the results of them. This class of tree automata includes regular tree automata with associativity normalization (regular TA + $R_{\mathsf{A}}$) that accept the counterpart of regular word languages.

In equational tree automata, depending on the equational theory, we may lose the Boolean closedness or an important decidability result. In addition to the above-mentioned problems, it is known that regular tree automata modulo ACI theory (*associativity, commutativity and idempotency*) are not closed under complement. There is the same problem in the classes of tree automata modulo *exclusive-or* and *Abelian group* theories. In the left-table below, we summarize the closure properties. The class of languages accepted by *regular tree automata with associativity normalization* is closed under associativity ($=_{\mathsf{A}}$), union ($\cup$), intersection ($\cap$) and complement ($(\ )^{\mathsf{c}}$). Moreover, the membership ($\in?$), emptiness ($=\varnothing?$), universality ($=T?$) and inclusion ($\subseteq?$) problems are decidable. In the tables, the positive results are indicated by ✓, and the negative results are –. These results for regular TA + $R_{\mathsf{A}}$ and for other useful classes are the consequence of Theorems 1–5 in the following Sections 3 and 4.

| | $=_{\mathsf{A}}$ | $\cup$ | $\cap$ | $(\ )^{\mathsf{c}}$ | $\in?$ | $=\varnothing?$ | $=T?$ | $\subseteq?$ |
|---|---|---|---|---|---|---|---|---|
| monotone A-TA (CSG) | ✓ | ✓ | ✓ | ✓ | ✓ | – | – | – |
| regular A-TA (CFG) | ✓ | ✓ | – | – | ✓ | ✓ | – | – |
| regular TA + $R_{\mathsf{A}}$ (RG) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

The class of the above regular TA + $R_{\mathsf{A}}$ is closely related to *hedge automata*. Hedge automata, originated from [27], are the automata which accepts *unranked* trees, whose transition rules have special patterns which matches regular sequence of state symbols. In practice, every hedge automaton can be minimized,

determinized and completely defined. Moreover, though the instances of transition rules are essentially not finite, most of the decidability questions are solvable. In Section 5, we investigate the relationship between the two frameworks.

As an early work of bi-directional translation between ranked and unranked tree languages, we should address the work of Carme, Niehren and Tommasi [3]. They showed that the hedges over the alphabet $\Sigma$ are manipulated by a special class of regular tree automata, called *stepwise tree automata*, whose signature is constants from $\Sigma$ and the binary symbol @. In our tree automata framework, their interpretation of hedges and the recognition in stepwise automata are performed by normalization and transition. The detailed discussion on related work and the applications of our framework are summarized in Section 6.

## 2 Preliminaries

We assume the reader is familiar with term rewriting [2] and tree automata [4]. An *equational theory* is a pair $\mathcal{E} = (F, E)$ in which $F$ is a finite set of function symbols, each with an associated *arity*, and $E$ is a finite set of (orientation sensitive) axioms over the function symbols in $F$ with possibly some variables from $V$. The binary relation $\rightarrow_{\mathcal{E}}$ induced by $\mathcal{E}$ is the rewrite relation, i.e. $s \rightarrow_{\mathcal{E}} t$ if there is an axiom $l = r$ in $E$, a context $C$ and a substitution $\sigma$ such that $s = C[l\sigma]$ and $t = C[r\sigma]$. The equivalence closure of $\rightarrow_{\mathcal{E}}$ is denoted by $=_{\mathcal{E}}$. Associative and commutative theory is an equational theory whose axioms are the associativity and commutativity for some of the binary function symbols. Given a binary function symbol $f \in F$, $f(f(x, y), z) = f(x, f(y, z))$ is an associativity ($\mathsf{A}$) axiom, and $f(x, y) = f(y, x)$ is a commutativity ($\mathsf{C}$) axiom.

An *equational rewriting system* is a pair $\mathcal{R} = (\mathcal{E}, R)$ in which $R$ is a finite set of rewrite rules over $F$ with $V$. The binary relation $\rightarrow_{\mathcal{R}}$ is a rewrite relation modulo axioms, i.e. $s \rightarrow_{\mathcal{R}} t$ if there is a rewrite rule $l \rightarrow r$ in $R$, a context $C$ and a substitution $\sigma$ such that $s =_{\mathcal{E}} C[l\sigma]$ and $t =_{\mathcal{E}} C[r\sigma]$. The reflexive transitive closure and the transitive closure of $\rightarrow_{\mathcal{R}}$ are denoted by $\rightarrow^*_{\mathcal{R}}$ and $\rightarrow^+_{\mathcal{R}}$, respectively. Given a term $s$, $s \rightarrow^!_{\mathcal{R}} t$ if $s \rightarrow^*_{\mathcal{R}} t$ and $t$ is a normal form. We write $(s)\downarrow^!_{\mathcal{R}}$ to denote such $t$. For the set $\mathcal{T}(F)$ of ground terms over the signature $F$, $\mathrm{NF}_F(\mathcal{R})$ represents the set of normal forms of $\mathcal{R}$ over $F$. An equational rewrite system $\mathcal{R}$ is weakly normalizing if for every term $t$, there exists some $(t)\downarrow^!_{\mathcal{R}}$. We say $\mathcal{R}$ ($R$ modulo $\mathcal{E}$) is confluent/strongly normalizing if $\rightarrow_{\mathcal{R}}$ is confluent/strongly normalizing.

Given an equational theory $\mathcal{E} = (F, E)$ and an equational rewriting system $\mathcal{R} = (\mathcal{E}, R)$, the *$\mathcal{E}$-closure* of a subset $L$ of terms, denoted by $\mathcal{E}(L)$, is the smallest set containing $L$ closed under $=_{\mathcal{E}}$, i.e. if $s =_{\mathcal{E}} t$ and $s \in \mathcal{E}(L)$ then $t \in \mathcal{E}(L)$. The *$\mathcal{R}$-descendant closure* of $L$, $\mathcal{R}^*(L)$, is the smallest set containing $L$ closed under $\rightarrow^*_{\mathcal{R}}$. Similarly, the *$\mathcal{R}$-ascendant closure* $(\mathcal{R}^{-1})^*(L)$ is the smallest set containing $L$ closed under $\rightarrow^*_{\mathcal{R}^{-1}}$ and the $\mathcal{R}$-equivalence closure $(\mathcal{R} \cup \mathcal{R}^{-1})^*(L)$ is the smallest set containing $L$ closed under $=_{\mathcal{R}}$.

A *tree automaton with normalization* (TAN) is a tuple $\mathcal{A} = (\mathcal{R}, Q, Q_{\mathrm{fin}}, \Delta)$, consisting of the equational rewrite system $\mathcal{R} = (\mathcal{E}, R)$ with $\mathcal{E} = (F, E)$, a finite

set $Q$ of states disjoint from the symbols in $F$, a subset $Q_{\mathrm{fin}}$ of $Q$, and a finite set $\Delta$ of transition rules whose shapes are in the following forms:

$$\text{(REGULAR)} \qquad f(\alpha_1(x_1), \ldots, \alpha_n(x_n)) \to \beta(f(x_1, \ldots, x_n)) \qquad \text{(EPSILON)} \qquad \alpha(x_1) \to \beta(x_1)$$

for some $f \in F$ with $\mathsf{arity}(f) = n$ and $\alpha_1, \ldots, \alpha_n, \alpha, \beta \in Q$. Every state is a unary symbol, and variables $x_i, x_j$ in the transition rule are different if $i \neq j$.

A *move relation* of $\mathcal{A}$ is concatenation of the two kinds of relations over $\mathcal{T}(F \cup Q)$, the normalization $\to_{\mathcal{R}}^{!}$ and the transition $\to_{\Delta}$ modulo $\mathcal{E}$, that means: $s \to_{\mathcal{A}} t$ if (1) there is a normal form $u$ of $s$ with respect to $\to_{\mathcal{R}}$ and (2) there is a transition rule $l \to r$ in $\Delta$, a context $C$ and a substitution $\sigma$ such that $u =_{\mathcal{E}} C[l\sigma]$ and $t =_{\mathcal{E}} C[r\sigma]$. If there is no normal form $(s)\downarrow_{\mathcal{R}}^{!}$, the transition fails.

A tree $t$ is *accepted* by $\mathcal{A}$ if $t \in \mathcal{T}(F)$ and $t \to_{\mathcal{A}}^{*} \alpha(t')$ for some $\alpha \in Q_{\mathrm{fin}}$ and $t' \in \mathcal{T}(F)$. The set of trees accepted by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$. We say a TAN $\mathcal{A} = (\mathcal{R}, Q, Q_{\mathrm{fin}}\Delta)$ with $\mathcal{R} = (F, E, R)$ is *regular* if transition rules in $\Delta$ are all in the REGULAR form. In case of $R = \varnothing$ and $E = \varnothing$, regular TAN are called regular tree automata. A tree language accepted by a regular tree automaton is called *regular*. These notions of regularity coincide with, if $R = \varnothing$ and $E = \varnothing$, the standard definition, e.g. [4].

As an example, we consider the TAN $\mathcal{A}_1$ with

$$\begin{aligned}
R_1\colon&\ \mathsf{f}(\mathsf{h}(x), \mathsf{h}(x)) \to \mathsf{h}(x)\\
E_1\colon&\ \mathsf{f}(\mathsf{f}(x, y), z) = \mathsf{f}(x, \mathsf{f}(y, z)) \quad \mathsf{f}(x, y) = \mathsf{f}(y, x)\\
\Delta_1\colon&\ \mathsf{a} \to \alpha(\mathsf{a}) \quad \mathsf{g}(\alpha(x)) \to \alpha(\mathsf{g}(x)) \quad \mathsf{h}(\alpha(x)) \to \beta(\mathsf{h}(x))
\end{aligned}$$

where $\beta$ is the final state. Then $\mathsf{h}(\mathsf{g}(\mathsf{a}))$ is accepted by $\mathcal{A}_1$, because $\mathsf{h}(\mathsf{g}(\mathsf{a})) \to_{\mathcal{R}_1}^{!} \mathsf{h}(\mathsf{g}(\mathsf{a}))$ and $\mathsf{h}(\mathsf{g}(\mathsf{a})) \to_{\Delta_1} \mathsf{h}(\mathsf{g}(\alpha(\mathsf{a})))$, and this turns out $\mathsf{h}(\mathsf{g}(\mathsf{a})) \to_{\mathcal{A}_1}^{*} \beta(\mathsf{h}(\mathsf{g}(\mathsf{a})))$.

Regarding the language accepted by this example, we observe that

> if a tree $t$ consists of nodes $\mathsf{f}$ with the same tree $\mathsf{h}(\mathsf{g}^i(\mathsf{a}))$ at each leaf ($i \geqslant 0$),
> $t$ is accepted by $\mathcal{A}_1$,

because for every such tree $t$, $t \to_{\mathcal{R}_1}^{!} \mathsf{h}(\mathsf{g}^i(\mathsf{a}))$, and then $\mathsf{h}(\mathsf{g}^i(\mathsf{a})) \to_{\mathcal{A}_1}^{*} \beta(\mathsf{h}(\mathsf{g}^i(\mathsf{a})))$. For instance, $\mathsf{f}(\mathsf{h}(\mathsf{g}(\mathsf{a})), \mathsf{h}(\mathsf{g}(\mathsf{a}))) \to_{\mathcal{R}_1}^{!} \mathsf{h}(\mathsf{g}(\mathsf{a}))$, and thus $\mathsf{h}(\mathsf{g}(\mathsf{a})) \to_{\mathcal{A}_1}^{*} \beta(\mathsf{h}(\mathsf{g}(\mathsf{a})))$ from the above example.

In the next example, we compare the expressiveness between TAN and ETA. ETA (equational tree automata [23]) are the special class of TAN whose set $R$ of rewrite rules is empty. We take the TAN $\mathcal{A}_2$ with the following components:

$$\begin{aligned}
R_2\colon&\ \mathsf{f}(\mathsf{h}(x), \mathsf{h}(x)) \to \mathsf{h}(x)\\
E_2\colon&\ \mathsf{f}(\mathsf{f}(x, y), z) = \mathsf{f}(x, \mathsf{f}(y, z)) \quad \mathsf{f}(x, y) = \mathsf{f}(y, x)\\
\Delta_2\colon&\ \mathsf{a} \to \alpha(\mathsf{a}) \quad \mathsf{g}(\alpha(x)) \to \alpha(\mathsf{g}(x)) \quad \mathsf{h}(\alpha(x)) \to \beta(\mathsf{h}(x))\\
&\ \mathsf{f}(\beta(x), \beta(y)) \to \gamma(\mathsf{f}(x, y))
\end{aligned}$$

In this example, $\mathcal{A}_2$ has the the additional transition rule $\mathsf{f}(\beta(x), \beta(y)) \to \gamma(\mathsf{f}(x, y))$ with the final state $\gamma$. Similar to the previous example, we observe that

if a tree $t$ consists of nodes $\mathsf{f}$ with two different kinds of trees $\mathsf{h}(\mathsf{g}^i(\mathsf{a}))$ and $\mathsf{h}(\mathsf{g}^j(\mathsf{a}))$ at its leaves ($i \neq j$), $t$ is accepted by $\mathcal{A}_2$.

For instance, $\mathsf{f}(\mathsf{h}(\mathsf{a}), \mathsf{f}(\mathsf{h}(\mathsf{g}(\mathsf{a})), \mathsf{h}(\mathsf{a})))$ is accepted: $\mathsf{f}(\mathsf{h}(\mathsf{a}), \mathsf{f}(\mathsf{h}(\mathsf{g}(\mathsf{a})), \mathsf{h}(\mathsf{a}))) \to^!_{\mathcal{R}_2} \mathsf{f}(\mathsf{h}(\mathsf{a}), \mathsf{h}(\mathsf{g}(\mathsf{a})))$, and therefore $\mathsf{f}(\mathsf{h}(\mathsf{a}), \mathsf{f}(\mathsf{h}(\mathsf{g}(\mathsf{a})), \mathsf{h}(\mathsf{a}))) \to^*_{\mathcal{A}_2} \gamma(\mathsf{f}(\mathsf{h}(\mathsf{a}), \mathsf{h}(\mathsf{g}(\mathsf{a}))))$. However, $\mathsf{f}(\mathsf{h}(\mathsf{a}), \mathsf{h}(\mathsf{a}))$ is not accepted by $\mathcal{A}_2$, because $\mathsf{f}(\mathsf{h}(\mathsf{a}), \mathsf{h}(\mathsf{a})) \to^!_{\mathcal{R}_2} \mathsf{h}(\mathsf{a})$ and $\mathsf{h}(\mathsf{a}) \to^*_{\mathcal{A}_2} \beta(\mathsf{h}(\mathsf{a}))$. In fact, $\mathsf{f}(\mathsf{h}(\mathsf{a}), \mathsf{h}(\mathsf{a})) \to^*_{\mathcal{A}_2} \beta(\mathsf{h}(\mathsf{a}))$.

*Remark 1.* The previous language $\mathcal{L}(\mathcal{A}_2)$ can not be represented by ETA if $E = \{\, \mathsf{f}(\mathsf{h}(x), \mathsf{h}(x)) = \mathsf{h}(x),\ \mathsf{f}(\mathsf{f}(x, y), z) = \mathsf{f}(x, f(y, z)),\ \mathsf{f}(x, y) = \mathsf{f}(y, x) \,\}$.

Furthermore, the language $\mathcal{L}(\mathcal{A}_2)$ is not accepted by multitree automata [14] or two-way equational tree automata [28] either.

## 3 Relative Decidability

We begin with the membership problem for the class of TAN. Hereafter in the following two sections, we suppose $\mathcal{E} = (F, E)$ and $\mathcal{R} = (\mathcal{E}, R)$.

**Theorem 1.** *The membership problem for $\mathcal{A}$ is decidable if $\mathcal{R}$ is strongly normalizing over $\mathcal{T}(F)$ and the membership problem for an equational tree automaton $\mathcal{A}'$ is decidable, where $\mathcal{A}'$ is defined by replacing $R$ in $\mathcal{A}$ with $\varnothing$.* □

In this framework, the decidability results depend upon that of equational tree automata, because we have $(\to^!_{\mathcal{R}} \cdot \to_{\mathcal{A}'})^+ \subseteq \to^!_{\mathcal{R}} \cdot \to^+_{\mathcal{A}'}$. To be precise, we state the following lemma.

**Lemma 1.** *Given $\mathcal{A} = (\mathcal{R}, Q, Q_{\mathrm{fin}}, \Delta)$, let $\mathcal{A}' = (\mathcal{R}', Q, Q_{\mathrm{fin}}, \Delta)$ with $\mathcal{R}' = (\mathcal{E}, \varnothing)$, then for every $t \in T(F)$, $t \to^+_{\mathcal{A}} t'$ implies $(t)\!\downarrow^!_{\mathcal{R}} \to^*_{\mathcal{A}'} t'$.* □

Accordingly, the question if $t \in \mathcal{L}(\mathcal{A})$ is determined by testing $(t)\!\downarrow^!_{\mathcal{R}} \in \mathcal{L}(\mathcal{A}')$ for all $(t)\!\downarrow^!_{\mathcal{R}}$. From König's Lemma, when $\mathcal{R}$ is strongly normalizing, there are only finitely many candidates for $(t)\!\downarrow^!_{\mathcal{R}}$. Strong normalization in Theorem 1, however, can not be weakened to weak normalization (Appendix A).

Every language accepted by a TAN $\mathcal{A} = (\mathcal{R}, Q, Q_{\mathrm{fin}}, \Delta)$ is the $\mathcal{R}$-ascendant closure of some subset of normal forms: Let $\mathcal{A}' = ((\mathcal{E}, \varnothing), Q, Q_{\mathrm{fin}}, \Delta)$,

(i) $\mathcal{L}(\mathcal{A}) = (\mathcal{R}^{-1})^*(\mathrm{NF}_F(\mathcal{R}) \cap \mathcal{L}(\mathcal{A}'))$.

Moreover,

(ii) $\mathcal{E}(\mathcal{L}(\mathcal{A})) = \mathcal{L}(\mathcal{A})$,

(iii) $(\mathcal{R} \cup \mathcal{R}^{-1})^*(\mathcal{L}(\mathcal{A})) = \mathcal{R}^*(\mathcal{L}(\mathcal{A})) = \mathcal{L}(\mathcal{A})$ if $\mathcal{R}$ is confluent.

For instance, we consider the TAN $\mathcal{A}_1$ with

$R$: $\mathsf{f}(\mathsf{h}(x), \mathsf{h}(x)) \to \mathsf{h}(x)$

$E$: $\mathsf{f}(\mathsf{f}(x, y), z) = \mathsf{f}(x, \mathsf{f}(y, z))$ $\quad$ $\mathsf{f}(x, y) = \mathsf{f}(y, x)$

$\Delta$: $\mathsf{a} \to \alpha(\mathsf{a})$ $\quad$ $\mathsf{g}(\alpha(x)) \to \alpha(\mathsf{g}(x))$ $\quad$ $\mathsf{h}(\alpha(x)) \to \beta(\mathsf{h}(x))$

We have $\mathcal{L}(\mathcal{A}_1) = (\mathcal{R}^{-1})^*(\mathcal{L}(\mathcal{A}'_1))$. This follows from Property (i) and $\mathrm{NF}_F(\mathcal{R}) \cap \mathcal{L}(\mathcal{A}'_1) = \mathcal{L}(\mathcal{A}'_1)$. Furthermore, $(\mathcal{R} \cup \mathcal{R}^{-1})^*(\mathcal{L}(\mathcal{A}'_1)) = \mathcal{L}(\mathcal{A}_1)$, because $\mathcal{L}(\mathcal{A}'_1) \subseteq \mathcal{L}(\mathcal{A}_1)$ from $\mathcal{L}(\mathcal{A}_1) = (\mathcal{R}^{-1})^*(\mathcal{L}(\mathcal{A}'_1))$. Since $\mathcal{R}$ is confluent, $(\mathcal{R} \cup \mathcal{R}^{-1})^*(\mathcal{L}(\mathcal{A}_1)) = \mathcal{L}(\mathcal{A}_1)$ from Property (iii). Therefore, $(\mathcal{R}^{-1})^*(\mathcal{L}(\mathcal{A}'_1)) \subseteq (\mathcal{R} \cup \mathcal{R}^{-1})^*(\mathcal{L}(\mathcal{A}'_1)) \subseteq (\mathcal{R} \cup \mathcal{R}^{-1})^*(\mathcal{L}(\mathcal{A}_1))$.

**Theorem 2.** *The emptiness problem is decidable for tree automata with normalization if*

*(1) there effectively exists an equational tree automaton (with the same equational theory) which accepts the set of normal forms of $\mathcal{R}$,*

*(2) this class of equational tree automata is closed under the intersection.*

*Proof.* Let $\mathcal{A}_R = ((\mathcal{E}, R), P, P_{\mathrm{fin}}, \Delta)$ and $\mathcal{A}_\varnothing = ((\mathcal{E}, \varnothing), P, P_{\mathrm{fin}}, \Delta)$. The ETA $\mathcal{A}_\varnothing$ is obtained from the TAN $\mathcal{A}_R$ by replacing $R$ with $\varnothing$. From the assumption (1), we obtain an ETA $\mathcal{B}$, equipped with the same theory $\mathcal{E} = (F, E)$, that accepts $\mathrm{NF}_F(\mathcal{R})$. This implies from Property (i) that

$$\mathcal{L}(\mathcal{A}_R) = \varnothing \;\Leftrightarrow\; \mathcal{L}(\mathcal{A}_\varnothing) \cap \mathcal{L}(\mathcal{B}) = \varnothing.$$

From the assumption (2), we have an ETA $\mathcal{C} = ((F, E, \varnothing), Q, Q_{\mathrm{fin}}, \Lambda)$ that accepts $\mathcal{L}(\mathcal{A}_\varnothing) \cap \mathcal{L}(\mathcal{B})$. From Lemma 9 (Appendix A), $\mathcal{L}(\mathcal{C})$ is the $\mathcal{E}$-closure of the language accepted by $\mathcal{C}' = ((F, \varnothing, \varnothing), Q, Q_{\mathrm{fin}}, \Lambda)$. Hence, $\mathcal{L}(\mathcal{A}_R) = \varnothing \;\Leftrightarrow\; \mathcal{L}(\mathcal{C}') = \varnothing$. Because $\mathcal{C}'$ is in the class of regular tree automata, the emptiness question to $\mathcal{C}'$ is decidable. □

The first equivalence in the above proof can be rephrased as follows: $\mathcal{L}(\mathcal{A}_R) \neq \varnothing$ if and only if $\mathcal{L}(\mathcal{A}_\varnothing)$ contains a normal form of $\mathcal{R}$. When $E = \varnothing$ in Theorem 2, the latter question is known to be EXPTIME-complete regardless of $R$ [5].

**Corollary 1.** *If $E = \varnothing$, the conditions (1) and (2) can be omitted.* □

**Theorem 3.** *The inclusion problem is decidable for tree automata with normalization if the conditions (1),(2) hold and the inclusion problem is decidable for this class of equational tree automata.* □

It is known that, if $\mathcal{E}$ is AC theory for a binary symbol $f$, the inclusion problem for ETA is decidable. Even if $\mathcal{E}$ also contains the axioms of $f(x, 0) = x$ and/or $g(f(x, y)) = f(g(x), g(y))$, the inclusion problem is decidable [28, 29].

Let $\mathcal{B}$ and $\mathcal{D}$ be ETA equipped with the equational theory $\mathcal{E}$, such that $\mathcal{B}$ accepts $\mathrm{NF}_F(\mathcal{R})$ and $\mathcal{D}$ accepts trees which are not normalizable. The universality question if $\mathcal{L}(\mathcal{A}_R) = \mathcal{T}(F)$ is equivalent to ask if $\mathcal{L}(\mathcal{B}) \subseteq \mathcal{L}(\mathcal{A}_\varnothing)$ and $\mathcal{L}(\mathcal{D}) = \varnothing$:

**Theorem 4.** *The universality is decidable for tree automata with normalization if the conditions (1),(3) hold and the inclusion problem is decidable for this class of equational tree automata:*

*(3) there effectively exists an equational tree automaton (with the same equational theory) which accepts the set of non-normalizable trees.* □

**Corollary 2.** *If $\mathcal{R}$ is weakly normalizing, the condition (3) can be omitted.* □

## 4 Closedness under Boolean Operations

**Lemma 2.** *For every equational rewrite system $\mathcal{R}$, the class of tree automata with $\mathcal{R}$-normalization is closed under complement if $\mathcal{R}$ is weakly normalizing and confluent over $\mathcal{T}(F)$ and this class of equational tree automata is closed under complement.* □

In the previous lemma, weak normalization is essential. Let us consider $\mathcal{R}_1 = ((\{\mathsf{a}, \mathsf{b}\}, \varnothing), \{\mathsf{a} \to \mathsf{a}\})$. Because $\mathsf{a}$ is not normalized, by definition, $\mathsf{a}$ is not accepted by TAN with $\mathcal{R}_1$. This implies that though the language $\{\mathsf{b}\}$ is accepted by $\mathcal{A}_1 = (\mathcal{R}_1, \{\alpha\}, \{\alpha\}, \{\mathsf{b} \to \alpha(\mathsf{b})\})$, the complement of $\mathcal{L}(\mathcal{A}_1)$ is not accepted by TAN with $\mathcal{R}_1$.

Also, confluence is necessary for closure under complement. We consider $\mathcal{R}_2 = ((\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}, \varnothing), \{\mathsf{a} \to \mathsf{b}, \mathsf{a} \to \mathsf{c}\})$ and $\mathcal{A}_2 = (\mathcal{R}_2, \{\alpha\}, \{\alpha\}, \{\mathsf{b} \to \alpha(\mathsf{b})\})$. Because $\mathsf{a}$ is normalized to $\mathsf{b}$, $\mathsf{a}$ is accepted by $\mathcal{A}_2$ as $\mathsf{b}$ is accepted. Every TAN with $\mathcal{R}_2$ accepting $\mathsf{c}$ accepts $\mathsf{a}$, because $\mathsf{a} \to^!_{\mathcal{R}_2} \mathsf{c}$. This implies that the complement of $\mathcal{L}(\mathcal{A}_2)$ is not accepted by TAN with $\mathcal{R}_2$.

**Theorem 5.** *For every equational rewrite system $\mathcal{R}$, the class of tree automata with $\mathcal{R}$-normalization is closed under Boolean operations if $\mathcal{R}$ is weakly normalizing and confluent over $\mathcal{T}(F)$ and this class of equational tree automata is closed under Boolean operations.* □

In the above theorem, confluence is essential for closure under intersection: Consider the previous example $\mathcal{R}_2 = ((\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}, \varnothing), \{\mathsf{a} \to \mathsf{b}, \mathsf{a} \to \mathsf{c}\})$. Let $\mathcal{A}_2 = (\mathcal{R}_2, \{\alpha\}, \{\alpha\}, \{\mathsf{b} \to \alpha(\mathsf{b})\})$ and $\mathcal{A}_3 = (\mathcal{R}_2, \{\alpha\}, \{\alpha\}, \{\mathsf{c} \to \alpha(\mathsf{c})\})$. The intersection of the two languages $\mathcal{L}(\mathcal{A}_2)$ and $\mathcal{L}(\mathcal{A}_3)$, which is $\{\mathsf{a}, \mathsf{b}\} \cap \{\mathsf{a}, \mathsf{c}\}$, is not accepted by TAN with $\mathcal{R}_2$, because every TAN with $\mathcal{R}_2$ accepting $\mathsf{a}$ accepts $\mathsf{b}$ or $\mathsf{c}$.

However, weak normalization is not needed for closure under intersection. Moreover, weak normalization and confluence are not required for the union.

**Lemma 3.** *Let $\mathcal{E} = (F, \mathsf{AC})$ whose component $\mathsf{AC}$ is the set of AC axioms for $\mathsf{f}$. If $R \in \{R_\mathsf{U}, R_\mathsf{I}, R_\mathsf{N}, R_\mathsf{D}, R_{\mathsf{UI}}, R_{\mathsf{UN}}, R_{\mathsf{UD}}, R_{\mathrm{AG}}\}$ defined below, $(\mathcal{E}, R)$ is strongly normalizing and confluent over $\mathcal{T}(F)$:*

$R_\mathsf{U}\colon\ \mathsf{f}(x, \mathsf{0}) \to x$ $\quad R_{\mathsf{UI}}\colon\ R_\mathsf{U} \cup R_\mathsf{I}$ $\quad R_{\mathsf{UN}}\colon\ R_\mathsf{U} \cup R_\mathsf{N}$ $\quad R_{\mathsf{UD}}\colon\ R_\mathsf{U} \cup R_\mathsf{D}$

$R_\mathsf{I}\colon\ \mathsf{f}(x, x) \to x$

$R_\mathsf{N}\colon\ \mathsf{f}(x, x) \to \mathsf{0}$

$R_\mathsf{D}\colon\ \mathsf{g}(\mathsf{f}(x, y)) \to \mathsf{f}(\mathsf{g}(x), \mathsf{g}(y))$ $\quad \mathsf{g}(\mathsf{0}) \to \mathsf{0}$ $\quad \mathsf{g}(\mathsf{g}(x)) \to x$

$R_\mathsf{G}\colon\ R_{\mathsf{UD}} \cup \{\mathsf{f}(x, \mathsf{g}(x)) \to \mathsf{0}\}$ $\quad$ (Abelian Group)

*Proof.* Strong normalization can easily be shown, and thus, confluence follows from Critical Pair Lemma [2]. Tool support, e.g. that of C*i*ME [6], is helpful in checking all AC critical pairs to be joinable. □

**Corollary 3.** *If* $R \in \{ R_{\mathsf{U}}, R_{\mathsf{I}}, R_{\mathsf{N}}, R_{\mathsf{D}}, R_{\mathsf{UI}}, R_{\mathsf{UN}}, R_{\mathsf{UD}}, R_{\mathsf{G}} \}$ *and* $E$ *is the set of AC axioms for* $\mathsf{f}$*, the class of tree automata with normalization is closed under Boolean operations. Moreover, the membership problem is decidable for this class of tree automata with normalization.* □

*Remark 2.* Equational tree automata with ACUI (resp. ACUN) axioms are *not* closed under complement. Applying a similar proof of Remark 1 (Appendix A), we can even show that equational tree automata with ACI (resp. ACN) axioms are also *not* closed under complement. The same problem is discussed in [29].

Furthermore, it is not obvious for ETA how to deal with Boolean operations in ACD and ACUD cases. Though it is shown in [29] that the class of tree languages modulo ACUD is closed under Boolean operations, the computation algorithm is quite involved. In our framework, Boolean computation for TAN with $R_{\mathsf{UD}}$ modulo AC can be manipulated by the operations for AC tree automata.

**Lemma 4.** *There effectively exists an AC-tree automaton which accepts the set of normal forms of* $R_{\mathsf{U}}$, $R_{\mathsf{D}}$, $R_{\mathsf{UD}}$ *modulo the* AC *axioms, respectively.*

*Proof.* We consider $R_{\mathsf{U}}$. Because of left-linearity, there effectively exists a tree automaton $\mathcal{A} = ((F, \varnothing, \varnothing), Q, Q_{\mathrm{fin}}, \Delta)$ which accepts normal forms of $R_{\mathsf{U}}$. The set of normal forms of $R_{\mathsf{U}}$ modulo AC is the AC-closure of $\mathcal{L}(\mathcal{A})$, which is accepted by $((F, \mathsf{AC}, \varnothing), Q, Q_{\mathrm{fin}}, \Delta)$. Similarly, we can prove for $R_{\mathsf{D}}$ and $R_{\mathsf{UD}}$. □

**Corollary 4.** *The emptiness, universality and inclusion problems are decidable for* $R_{\mathsf{U}}$, $R_{\mathsf{D}}$, $R_{\mathsf{UD}}$ *modulo the* AC *axioms.* □

**Corollary 5.** *The emptiness problem is decidable for* $R_{\mathsf{I}}$, $R_{\mathsf{N}}$, $R_{\mathsf{UI}}$, $R_{\mathsf{UN}}$ *modulo the* AC *axioms if the following question is decidable: Given a regular tree automaton* $\mathcal{A}$*, does its language contain a normal form of* $R_{\mathsf{I}}$ *modulo* AC*?* □

## 5 Regular Leaf Languages and Hedge Automata

When discussing the expressiveness of tree languages, we often take the images of them into word languages. Leaves of a tree $t$ are the sequence in the left-to-right order of constants occurring in $t$. We denote leaf for the mapping that takes a tree and returns the leaves, i.e. $\mathsf{leaf}(f(t_1, \ldots, t_n)) = \mathsf{leaf}(t_1) \cdots \mathsf{leaf}(t_n)$ if $n \geqslant 1$; $\mathsf{leaf}(a) = a$, otherwise. The leaf language associated to a tree language $L$ is the set of leaves obtained from $L$.

Given a signature $F$, we take the rewrite system $\mathcal{R}_{\mathsf{reg}}$ whose rewrite rules are for arbitrary function symbols $f, g$ in $F$,

$$f(x_1, \ldots, g(y_1, y_2, \ldots, y_k), x_i, \ldots, x_n) \to f(x_1, \ldots, y_1, g(y_2, \ldots, y_k, x_i), \ldots, x_n)$$

such that $\mathsf{arity}(f) \geqslant 2$ and $\mathsf{arity}(g) \geqslant 1$. Here $f$ and $g$ are possibly the same. Obviously, $\mathcal{R}_{\mathsf{reg}}$ is strongly normalizing if $E = \varnothing$. However, $\mathcal{R}_{\mathsf{reg}}$ is *not* confluent, because $\mathsf{f}(\mathsf{g}(\mathsf{h}(\mathsf{a}, \mathsf{b}), \mathsf{c}), \mathsf{d}) \to^{!}_{\mathcal{R}_{\mathsf{reg}}} \mathsf{f}(\mathsf{a}, \mathsf{h}(\mathsf{b}, \mathsf{g}(\mathsf{c}, \mathsf{d})))$ and $\mathsf{f}(\mathsf{g}(\mathsf{h}(\mathsf{a}, \mathsf{b}), \mathsf{c}), \mathsf{d}) \to^{!}_{\mathcal{R}_{\mathsf{reg}}} \mathsf{f}(\mathsf{a}, \mathsf{g}(\mathsf{b}, \mathsf{h}(\mathsf{c}, \mathsf{d})))$, though it preserves the leaves.

**Lemma 5.** *Suppose that for every $s, t$ in $\mathcal{T}(F)$, $s =_{\mathcal{E}} t$ implies $\mathsf{leaf}(s) = \mathsf{leaf}(t)$. Then, $t =_{\mathcal{R}_{\mathsf{reg}}} u$ if and only if $\mathsf{leaf}(t) = \mathsf{leaf}(u)$.* □

Trees in $\mathcal{R}_{\mathsf{reg}}$-normal form, if $E = \varnothing$, are *right* associative as derivation trees of regular word grammar are right associative. For instance, when the grammar has the production rules $\alpha \to \mathsf{a}\beta$, $\beta \to \mathsf{b}\gamma$, $\gamma \to \mathsf{c}$, we have $\alpha(\mathsf{a} \circ \beta(\mathsf{b} \circ \gamma(\mathsf{c})))$ for the derivation starting from $\alpha$. The word $\mathsf{abc}$ is the leaves of this derivation tree. Meanwhile, in tree automata setting, the word $\mathsf{abc}$ can be represented to be $\mathsf{a} \cdot (\mathsf{b} \cdot \mathsf{c})$. Tree automaton accepting $\mathsf{a} \cdot (\mathsf{b} \cdot \mathsf{c})$ is equipped with the transition rules $\mathsf{a} \to p_{\mathsf{a}}(\mathsf{a})$, $p_{\mathsf{a}}(x) \cdot \beta(y) \to \alpha(x \cdot y)$, $\mathsf{b} \to p_{\mathsf{b}}(\mathsf{b})$, $p_{\mathsf{b}}(x) \cdot \gamma(y) \to \beta(x \cdot y)$, $\mathsf{c} \to \gamma(\mathsf{c})$. Here we use the infix operators $\circ$ and $\cdot$ for denoting the branching node of derivations and the concatenation of alphabet.

Accordingly, trees which have such *regular* leaves should contain constants that appear on the left in a tree whenever non-constant trees appear at the same level in the tree. If $\mathcal{A}$ is a tree automaton, the leaf language $\{\, \mathsf{leaf}(t) \mid (t)\!\downarrow^{!}_{\mathcal{R}_{\mathsf{reg}}}$ accepted by $\mathcal{A} \,\}$ is regular, while $\{\, \mathsf{leaf}(t) \mid t$ accepted by $\mathcal{A} \,\}$ is context-free. Since trees accepted by tree automata are derivation trees generated by context-free grammar, this observation can be easily obtained. But what about a tree automaton accepting $\{\, t \mid \mathsf{leaf}(t)$ generated by $\mathcal{G} \,\}$?

**Lemma 6.** *Given a regular grammar $\mathcal{G} = (\Sigma, \mathcal{S}, \mathsf{s}_0, \Lambda)$, we can construct a regular tree automaton $\mathcal{A}$ which accepts all trees whose leaves are generated by a regular grammar $\mathcal{G}$.* □

**Corollary 6.** *The above lemma can be extended to an arbitrary signature $F$ if $F$ contains constants from $\Sigma$ and a binary function symbol.* □

Of particular interest about Lemma 6 is that this property does not hold for context-free grammar. The proof of this observation is found in Appendix A.

**Lemma 7.** *For every regular tree automaton $\mathcal{A}$, $\{\, t \mid (t)\!\downarrow^{!}_{\mathcal{R}_{\mathsf{reg}}}$ accepted by $\mathcal{A} \,\}$ is a regular tree language.* □

Hence, the class of tree automata, each of which accepts all trees whose leaves is generated by a regular grammar, is a proper sub-class of regular tree automata. In the framework of TAN, this statement can be represented as follows.

**Theorem 6.** *The class of TAN with $\mathcal{R}_{\mathsf{reg}}$ is a sub-class of regular tree automata. Moreover, for every confluent sub-system $\mathcal{R}$ of $\mathcal{R}_{\mathsf{reg}}$, the class of TAN with $\mathcal{R}$ is also a sub-class of regular tree automata, and is*

- *closed under $=_{\mathcal{R}}$,*
- *closed under Boolean operations,*
- *decidable in the membership, emptiness, universality and inclusion problems.*

□

An associativity rewrite system $\mathcal{R}_{\mathsf{A}}$, whose rewrite rules are $f(f(x, y), z) \to f(x, f(y, z))$ for some of the binary function symbols, is such an example of confluent sub-systems. This class of rewrite systems is useful in translating *unranked* trees in ranked signature.

In case of *unranked* signature, trees with regular leaves are manipulated by *hedge automata*. Recently, the hedge automata theory is accepted in XML community [20], and in which XML documents are considered as tree-structured objects, called *hedges*. Hence, as if an XML schema matches documents, a hedge automaton accepts the set of hedges. Hedges in $\textsc{Hdg}(\Sigma, C)$ over a finite set $\Sigma$ of *label* symbols and a finite set $C$ of *constants* are recursively defined as follows:

$$
\begin{array}{lll}
t \in \textsc{Hdg}(\Sigma, C) & \text{if } t = c & \text{for some } c \in C \\
& \text{if } t = a\langle\, w \,\rangle & \text{for some } a \in \Sigma,\ w \in \textsc{Hdgs}(\Sigma, C) \\
w \in \textsc{Hdgs}(\Sigma, C) & \text{if } w = \epsilon & \text{(null hedge)} \\
& \text{if } w = t & \text{for some } t \in \textsc{Hdg}(\Sigma, C) \\
& \text{if } w = u \circ v & \text{for some } u, v \in \textsc{Hdgs}(\Sigma, C)
\end{array}
$$

The infix operator $\circ$ over hedges is associative over hedges with the identity axiom for $\epsilon$. In the unranked setting, $t_1 \circ \cdots \circ t_n$ can be denoted as $\circ(t_1, \ldots, t_n)$, and it satisfies that $\circ(t_1, \ldots, t_i, \epsilon, t_{i+1}, \ldots, t_n) = \circ(t_1, \ldots, t_i, t_{i+1}, \ldots, t_n)$.

One should remark on hedges defined above. In the literature, e.g. [19], the null hedge $\epsilon$ and the sequence of hedges are defined as well-formed hedges. However, this leads us to confusion in distinguishing *elements* from *contents* in the context of XML. According to XML schema recommended by W3C [30], an *empty-element tag*, that corresponds to some $c$ in $C$, is an element but is not the empty sequence of elements. On the other hand, an element (tagged $a$) with no content means the element having the empty sequence in between *start-tag* and *end-tag*, that corresponds $a\langle\ \rangle$.

A hedge automaton is a tuple $\mathcal{H} = (\Sigma, C, N, N_{\text{fin}}, \Delta)$, in which $N$ is a finite set of unary state symbols disjoint from $\Sigma$ and $C$, $N_{\text{fin}}$ is a set of final states with $N_{\text{fin}} \subseteq N$, and $\Delta$ is a set of transition rules for $\Sigma$ and $C$. Transition rules in $\Delta$ are in the two shapes of

$$
\begin{array}{c}
c \to \beta(c) \\
a\langle\, \alpha_1(x_1) \circ \cdots \circ \alpha_n(x_n) \,\rangle \to \beta(a\langle\, x_1 \circ \cdots \circ x_n \,\rangle) \ \Leftarrow\ \alpha_1 \cdots \alpha_n \in \mathcal{L}(\mathcal{G})
\end{array}
$$

with $a \in \Sigma$, $x_1, \ldots, x_n \in V$ and $\alpha_1, \ldots, \alpha_n, \beta \in N$.

Formally, we should introduce *sequence variables* ([13]) for the second type of rules, so the left-hand side can match an arbitrary sequence $\alpha_1(x_1) \circ \cdots \circ \alpha_n(x_n)$. In the sense of tree automata, we thus allow hedge automata to contain in $\Delta$ infinite number of regular transition rules in a certain type.

The grammar $\mathcal{G}$ over $N$ that appears in the second type of rules must be regular, but that can be different for each transition rule. The left-arrow ($\Leftarrow$) together with the membership condition represents the guard, meaning that, if the sequence $\alpha_1 \cdots \alpha_n$ of states is generated by $\mathcal{G}$, the transition fires. Variables $x_1, \ldots, x_n$ in a rule are instantiated to certain hedges when the rule is applied.

In tree automata framework, hedges can be modeled over the signature

$$
\begin{array}{lllll}
F_{\Sigma,C}\colon & \{\, a \mid a \in \Sigma \,\} & \mathsf{arity}(a) = 1 \text{ for each } a, & \{\, \circ \,\} & \mathsf{arity}(\circ) = 2, \\
& \{\, c \mid c \in C \,\} & \mathsf{arity}(c) = 0 \text{ for each } c, & \{\, \epsilon \,\} & \mathsf{arity}(\epsilon) = 0.
\end{array}
$$

$$
\begin{aligned}
\mathcal{H}_{\mathrm{in}}: \qquad \mathsf{f}\langle\, \gamma(x) \circ \beta(y) \,\rangle &\;\to\; \gamma(\mathsf{f}\langle\, x \circ y \,\rangle) \\
\mathsf{g}\langle\, \alpha(z_1) \circ \cdots \circ \alpha(z_m) \,\rangle &\;\to\; \gamma(\mathsf{g}\langle\, z_1 \circ \cdots \circ z_m \,\rangle) \\
\mathsf{g}\langle\, \beta(w_1) \circ \cdots \circ \beta(w_n) \,\rangle &\;\to\; \delta(\mathsf{g}\langle\, w_1 \circ \cdots \circ w_n \,\rangle) \\
\mathsf{a} &\;\to\; \alpha(\mathsf{a}) \\
\mathsf{b} &\;\to\; \beta(\mathsf{b})
\end{aligned}
$$

**Fig. 1.** Example of XML Schema

Hereafter, to discuss the translation from unranked to ranked signatures, we define the *unflattening* operation:

$$
\mathsf{unflat}(t) = \begin{cases} a(\mathsf{unflat}(t_1) \circ \cdots (\mathsf{unflat}(t_n) \circ \mathsf{unflat}(t_{n+1})) \cdots) \\ \qquad\qquad \text{if } t = a\langle\, t_1 \cdots t_n\, t_{n+1} \,\rangle \text{ with } n \geqslant 0, \\ a(\epsilon) \qquad\qquad \text{if } t = a\langle\, \rangle, \\ t \qquad\qquad \text{if } t \in C. \end{cases}
$$

Moreover, we use $\mathsf{flat}$ for the reverse translation, e.g. $\mathsf{flat}(\,\mathsf{f}((\mathsf{a} \circ \mathsf{b}) \circ \mathsf{g}(\mathsf{c}))\,) = \mathsf{f}\langle\, \mathsf{a} \circ \mathsf{b} \circ \mathsf{g}\langle\, \mathsf{c} \,\rangle \,\rangle$.

**Proposition 1.** *For every hedge automaton $\mathcal{H}$ over $\Sigma$ and $C$, we can construct a tree automaton $\mathcal{A}_{\mathcal{H}}$ over $F_{\Sigma,C}$ such that $t$ is accepted by $\mathcal{H}$ if and only if* $\mathsf{unflat}(t)$ *is accepted by $\mathcal{A}_{\mathcal{H}}$.* □

In the above setting, $\mathsf{f}(\mathsf{a} \circ (\mathsf{b} \circ \mathsf{c}))$ and $\mathsf{f}((\mathsf{a} \circ \mathsf{b}) \circ \mathsf{c})$ are distinguished, because we *do not* assume that $(x \circ y) \circ z = x \circ (y \circ z)$ is given. However, due to Lemma 7, we can generalize Proposition 1 as follows.

**Corollary 7.** *For every hedge automaton $\mathcal{H}$ over $\Sigma$ and $C$, we can construct a tree automaton $\mathcal{B}_{\mathcal{H}}$ over $F_{\Sigma,C}$ such that* $\mathsf{flat}(t)$ *is accepted by $\mathcal{H}$ if and only if $t$ is accepted by $\mathcal{B}_{\mathcal{H}}$.* □

The sizes $|\mathcal{A}_{\mathcal{H}}|$ and $|\mathcal{B}_{\mathcal{H}}|$ are $O(|\mathcal{H}|)$ and $O(|\mathcal{H}|^3)$, respectively. The time complexities are linear for the construction of $\mathcal{A}_{\mathcal{H}}$, and cubic for $\mathcal{B}_{\mathcal{H}}$ to the size of $\mathcal{H}$. In Corollary 7, we can take the TAN $\mathcal{A}_{\mathcal{H}}$ with $\mathcal{R}_{\mathsf{A}}$, instead of $\mathcal{B}_{\mathcal{H}}$. In this case, Proposition 1 is generalized without losing the advantages of the size and time complexities. Detailed complexity analysis is found in Appendix B.

As an application for XML manipulation, we consider the simple example (Fig. 1) that contains the hedge automaton $\mathcal{H}_{\mathrm{in}}$, whose final state is $\gamma$. The language accepted by $\mathcal{H}_{\mathrm{in}}$ consists of the hedges $\mathsf{f}\langle\, \mathsf{g}\langle\, \mathsf{a}^* \,\rangle \mathsf{g}\langle\, \mathsf{b}^* \,\rangle \,\rangle$. Here $\circ$ is omitted, and $\mathsf{a}^*$ ($\mathsf{b}^*$) denotes an arbitrary sequence of $\mathsf{a}$ (resp. $\mathsf{b}$). In the example, we suppose that $\mathsf{a}$ and $\mathsf{b}$ are the elements with different types, and $\mathsf{f}$ and $\mathsf{g}$ are the tags, such that $\mathsf{g}$ contains $\mathsf{a}$ or $\mathsf{b}$ only. According to Lemma 7, we can construct tree automata $\mathcal{A}_{\mathcal{H}_{\mathrm{in}}}$ over $F_{\Sigma,C}$, which corresponds to $\mathcal{H}_{\mathrm{in}}$.

Now we consider the constraint $\Psi_{\leqslant}$ for hedges, such that $\Psi_{\leqslant}(t)$ holds if $t = \mathsf{f}\langle\, \mathsf{g}\langle \mathsf{a}^* \rangle \mathsf{g}\langle \mathsf{b}^* \rangle \,\rangle$ and the number of occurrences of $\mathsf{a}$ is less than or equals to the number of $\mathsf{b}$. For instance, $\Psi_{\leqslant}(\mathsf{f}\langle\, \mathsf{g}\langle \mathsf{aa} \rangle \mathsf{g}\langle \mathsf{bbb} \rangle\rangle)$ is true.

In tree automata framework, the above predicate $\Psi_{\leqslant}$ can be translated to the TAN $\mathcal{A}_{\leqslant} = (\mathcal{R}, Q, Q_{\mathrm{fin}}, \Delta_{\leqslant})$ with $\mathcal{R}_{\leqslant} = (F_{\Sigma,C}, \varnothing, R_{\leqslant})$ as follows.

$$
\begin{aligned}
\Delta_{\leqslant}\colon\ & \mathsf{f}(\eta(x)) \to \theta(\mathsf{f}(x)) \qquad \gamma(x) \circ \delta(y) \to \eta(x \circ y) \\
& \mathsf{g}(\alpha(x)) \to \gamma(\mathsf{g}(x)) \qquad \epsilon \to \alpha(\epsilon) \\
& \mathsf{g}(\beta(x)) \to \delta(\mathsf{g}(x)) \qquad \epsilon \to \beta(\epsilon) \quad \mathsf{b} \to \beta(\mathsf{b}) \quad \beta(x) \circ \beta(y) \to \beta(x \circ y) \\
Q\colon\ & \alpha \quad \beta \quad \gamma \quad \delta \quad \eta \quad \theta \\
Q_{\mathrm{fin}}\colon\ & \theta \\
R_{\leqslant}\colon\ & \mathsf{f}(\mathsf{g}(\mathsf{a} \circ x) \circ \mathsf{g}(\mathsf{b} \circ y)) \to \mathsf{f}(\mathsf{g}(x) \circ \mathsf{g}(y)) \qquad \mathsf{f}(\mathsf{g}(\mathsf{a}) \circ \mathsf{g}(\mathsf{b})) \to \mathsf{f}(\mathsf{g}(\epsilon) \circ \mathsf{g}(\epsilon)) \\
& \mathsf{f}(\mathsf{g}(\mathsf{a} \circ x) \circ \mathsf{g}(\mathsf{b})) \to \mathsf{f}(\mathsf{g}(x) \circ \mathsf{g}(\epsilon)) \qquad \mathsf{f}(\mathsf{g}(\mathsf{a}) \circ \mathsf{g}(\mathsf{b} \circ y)) \to \mathsf{f}(\mathsf{g}(\epsilon) \circ \mathsf{g}(y)) \\
& (x \circ y) \circ z \to x \circ (y \circ z) \qquad \epsilon \circ x \to x \qquad x \circ \epsilon \to x
\end{aligned}
$$

**Lemma 8.** *$\mathcal{A}_{\leqslant}$ with $\mathcal{R}_{\leqslant}$ accepts $t$ if and only if $\mathsf{flat}(t)$ satisfies $\Psi_{\leqslant}$.* □

Using a similar construction, we can define the TAN $\mathcal{A}_{<}$, $\mathcal{A}_{\geqslant}$, $\mathcal{A}_{>}$ for the constraints $\Psi_{<}$ (less than), $\Psi_{\geqslant}$ (greater than or equals to) and $\Psi_{>}$ (greater than).

*Remark 3.* The languages $\mathcal{L}(\mathcal{A}_{\leqslant})$, $\mathcal{L}(\mathcal{A}_{<})$, $\mathcal{L}(\mathcal{A}_{\geqslant})$, $\mathcal{L}(\mathcal{A}_{>})$ are not regular.

Furthermore, $\mathcal{L}(\mathcal{A}_{\leqslant})$ is closed under Boolean operations and the membership problem is decidable, because the above $\mathcal{R}_{\leqslant}$ is strongly normalizing and confluent as there are 9 critical pairs and they are all joinable. Besides, the emptiness, universality and inclusion problems are decidable, because $R_{\leqslant}$ is left-linear and $E = \varnothing$. Similar properties hold for $\mathcal{L}(\mathcal{A}_{<})$, $\mathcal{L}(\mathcal{A}_{\geqslant})$ and $\mathcal{L}(\mathcal{A}_{>})$.

## 6 Related Work and Conclusions

Recently, Dal Zilio and Lugiez [7] and thereafter Seidl *et al.* [25] proposed the extension of XML schema language in which a commutative infix operator over hedges is equipped. We denote below this operator by $\otimes$. The concatenation $t_1 \otimes \cdots \otimes t_n$ of hedges is represented by the multiset $\{\!\{t_1, \ldots, t_n\}\!\}$ in [25]. By this extension, their frameworks require the additional sort of transition rules

$$a\langle\, \alpha_1(x_1) \otimes \cdots \otimes \alpha_n(x_n) \,\rangle \to \beta(a\langle\, x_1 \otimes \cdots \otimes x_n \,\rangle) \quad \Leftarrow \quad \alpha_1 \cdots \alpha_n \in \mathcal{L}(\mathcal{G}/\mathsf{C})$$

A *commutative regular grammar* $\mathcal{G}/\mathsf{C}$ is a regular grammar modulo the commutativity. The language generated by $\mathcal{G}/\mathsf{C}$ is the same as the commutative closure of $\mathcal{L}(\mathcal{G})$, i.e. $u \in \mathcal{L}(\mathcal{G}/\mathsf{C})$ if and only if $u =_{\mathsf{C}} v$ and $v \in \mathcal{L}(\mathcal{G})$. The class of the commutative closure of regular languages coincides with the class of the commutative closure of context-free languages. Moreover, the Parikh image [24] of regular languages (also of the commutative closure of regular languages) is the semi-linear sets, as well as Presburger definable languages. Using this fact,

Dal Zilio and Lugiez define a query language for XML, called *sheaves logic*, in which the formulas may contain Presburger constraints so the number of occurrences of a content with a certain type in a sequence of contents can be inquired.

In tree automata with ranked signature, this extension in hedge automata can be simulated by assuming that $\otimes$ has the associativity $(x \otimes y) \otimes z = x \otimes (y \otimes z)$ and commutativity $x \otimes y = y \otimes x$ axioms. Hedge automata with this $\otimes$ are thus modeled by regular tree automata modulo the AC axioms. However, the example of XML manipulation discussed in Section 5, e.g. Lemma 8, is beyond this class of ETA, and moreover,

*Remark 4.* the constraints $\Psi_{\leqslant}$, $\Psi_{<}$, $\Psi_{\geqslant}$, $\Psi_{>}$ are not expressed by sheaves logic.

Regarding the translation of hedge automata, the special class of tree automata, called *stepwise tree automata*, is studied [3]. The signature of this class consists of constant symbols from $\Sigma$ and the binary symbol @. Their transition rules are in the two forms:

$$a \to \alpha(a) \qquad\qquad \alpha(x) \mathbin{@} \beta(y) \to \gamma(x \mathbin{@} y)$$

for some $a \in \Sigma$ and $\alpha, \beta, \gamma$ are state symbols. Epsilon transition rules $\alpha(x) \to \beta(x)$ can be permitted though they do not appear formally in the original definition. This class of tree automata has the bidirectional correspondence to hedge automata in terms of Proposition 1. Moreover, we can find in [17] the comprehensive study on the minimization of stepwise automata.

In the research of XML document processing, one of the prominent topics is the *type checking problem*. This question asks us if, for XML document types $\tau_{\text{in}}$ and $\tau_{\text{out}}$ and an XML transformer $\Phi$, every document transformed by $\Phi$ from of type $\tau_{\text{in}}$ is of type $\tau_{\text{out}}$. In practice, these document types $\tau_{\text{in}}$ and $\tau_{\text{out}}$ are described in an XML schema language, and this problem is represented as the inclusion problem $\{\, t \mid \exists s \in \mathcal{L}(\tau_{\text{in}})\colon t = \Phi(s) \,\} \subseteq \mathcal{L}(\tau_{\text{out}})$. For reasoning about this problem, Milo *et al.* [18] studied a special class of transformer on ranked trees, called *k-pebble tree transducer* ($k$-PTT), and they showed that the complexity of the type checking problem for $k$-PTT is $k$-tower of exponential. Engelfriet *et al.* showed in [8] that $k$-PTT is simulated by $(k+1)$-fold compositions of another kind of transformer, called *stay macro tree transducers* (SMTT). Recently, Maneth *et al.* showed in [15] that the type checking can be solved in polynomial time if SMTT is *n-bounded copying*, that is, if nodes in an input tree are copied by transformer at most $n$-times in the output tree.

In rewriting, the type checking problem can be reformulated as the problem $\{\, t \mid \exists s \in \mathcal{L}(\tau_{\text{in}})\colon s \to^{!}_{\mathcal{R}_\Phi} t \,\} \subseteq \mathcal{L}(\tau_{\text{out}})$. In [26], Takai *et al.* proposed a class of rewrite systems that effectively preserves the regularity. This result enables us to handle the type checking problem whenever $\mathcal{R}_\Phi$ is in this class. For automated reasoning, Ohsaki and Takai have developed the verification tool *ACTAS* [21], and Genet *et al.* independently developed another tool *Timbuk* [9], that computes the exact and under-approximated $\mathcal{R}$-descendant closure of regular tree languages.

In rewriting community, equational rewriting has been studied over the years, and this theory underlies several practical challenges. In fact, there are executable specification languages, verification tools, and program optimization

tools that support rewriting modulo equational axioms. Among elaborated studies on equational rewriting, tree automata with normalization (TAN) may have the roots in *normalized rewriting* [16]. The theory of normalized rewriting brought the idea in dealing with completion modulo axioms, and in practice that developed the software C*i*ME [6].

In the paper, we showed the decidability (Theorems 1–4 and Corollaries 1,2) and the closure properties (Theorem 5 and Corollaries 3,4) of our tree automata framework. Corollary 5 can be improved if the intersection-emptiness of AC-regular language and $R_{\mathsf{l}}$-normal forms is a decidable question. In Section 5 we discussed advantages of TAN through the observation of transformations from XML schema to tree automata. In the example of XML manipulation, we showed by demonstrating that XML schema and a constraint can be translated to TAN.

For further exploration, we have an interest in the extension of TAN, in the direction of PTA [10] and of monotone ETA [23]. The tool development based on our framework is useful in practice, and it can be easily implemented in the rule-based software, in particular for the tools supporting AC tree automata [21], due to Lemma 1 and Properties (i)–(iii).

**Acknowledgments.** The authors would like to thank Ralf Treinen and Florent Jacquemard for their comments on the early work. We have done most of the preliminary work while Ohsaki visited in ENS de Cachan in August and September, 2006. We also thank Stéphanie Delaune for her suggestion that improves Lemma 3.

## References

1. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò and L. Vigneron: *The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications*, Proc. of 17th CAV, Edinburgh (UK), LNCS 3576, pp. 281–285, Springer, 2005.
2. F. Baader and T. Nipkow: *Term Rewriting and All That*, Cambridge University Press, 1998.
3. J. Carme, J. Niehren and M. Tommasi: *Querying Unranked Trees with Stepwise Tree Automata*, Proc. of 15th RTA, Aachen (Germany), LNCS 3091, pp. 105–118, Springer, 2004.
4. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison and M. Tommasi: *Tree Automata Techniques and Applications*, draft, 2005. Available at URL `http://www.grappa.univ-lille3.fr/tata`
5. H. Comon and F. Jacquemard: *Ground Reducibility is EXPTIME-Complete*, Information and Computation 187(1), pp. 123–153, Elsevier, 2003.
6. E. Contejean, C. Marché, B. Monate and X. Urbain: *The CiME System: Version 2.02*, 2004. Software and the document available at URL `http://cime.lri.fr/`
7. S. Dal Zilio and D. Lugiez: *XML Schema, Tree Logic and Sheaves Automata*, Applicable Algebra in Engineering, Communication and Computing 17(5), pp. 337–377, Springer, 2006.
8. J. Engelfriet and S. Maneth: *A Comparison of Pebble Tree Transducers with Macro Tree Transducers*, Acta Informatica 39, pp. 613–698, Springer, 2003.
9. G. Feuillade, T. Genet and V. Viet Triem Tong: *Reachability Analysis over Term Rewriting Systems*, Journal of Automated Reasoning 33, pp. 341-383, Springer, 2004.

10. J. Hendrix, H. Ohsaki and M. Viswanathan: *Propositional Tree Automata*, Proc. of 17th RTA, Seattle (Washington), LNCS 4098, pp. 50–65, Springer, 2006.
11. H. Hosoya, J. Vouillon and B.C. Pierce: *Regular Expression Types for XML*, Proc. of 5th ICFP, Montreal (Canada), SIGPLAN Notices 35(9), pp. 11–22, ACM Press, 2000.
12. N. Klarlund and A. Møller: *MONA Version 1.4 User Manual*, BRICS Notes Series NS-01-1, Department of Computer Science, University of Aarhus, 2001.
13. T. Kutsia: *Solving Equations Involving Sequence Variables and Sequence Functions*, Proc. of 7th AISC, Linz (Austria), LNCS 3249, pp. 157–170, Springer, 2004.
14. D. Lugiez: *Multitree Automata That Count*, Theoretical Computer Science 333(1–2), pp. 225–263, Elsevier, 2005.
15. S. Maneth, T. Perst and H. Seidl: *XML Type Checking in Polynomial Time*, Proc. of 11th ICDT, Barcelona (Spain), LNCS 4353, pp. 254–268, Springer, 2007.
16. C. Marché: *Normalized Rewriting: An Alternative to Rewriting Modulo a Set of Equations*, Journal of Symbolic Computation 21(3), pp. 253–288, Elsevier, 1996.
17. W. Martens and J. Niehren: *Minimizing Tree Automata for Unranked Trees*, Proc. of 10th DBPL, Trondheim (Norway), LNCS 3774, pp. 232–246, Springer, 2005.
18. T. Milo, D. Suciu and V. Vianu: *Typechecking for XML Transformers*, Proc. of 19th PODS, Dallas (Texas), pp. 11–22, ACM Press, 2000.
19. M. Murata: *Extended Path Expression for XML*, Proc. of 20th PODS, Santa Barbara (California), pp. 61–72, ACM Press, 2001.
20. M. Murata and P. Prescod: *SGML/XML and Forest/Hedge Automata Theory*, Online Resource for Markup Language Technologies, *Cover Pages*, OASIS, May 2001. Document available at URL `http://xml.coverpages.org/hedgeAutomata.html`
21. H. Ohsaki and T. Takai: *ACTAS : A System Design for Associative and Commutative Tree Automata Theory*, Proc. of 5th RULE, Aachen (Germany), ENTCS 124, pp. 97–111, Elsevier, 2005.
22. H. Ohsaki and T. Takai: *Decidability and Closure Properties of Equational Tree Languages*, Proc. of 13th RTA, Copenhagen (Denmark), LNCS 2378, pp. 114–128, Springer, 2002.
23. H. Ohsaki: *Beyond Regularity: Equational Tree Automata for Associative and Commutative Theories*, Proc. of 15th CSL, Paris (France), LNCS 2142, pp. 539–553, Springer, 2001.
24. R. Parikh: *On Context-Free Languages*, JACM 13(4), pp. 570–581, ACM Press, 1966.
25. H. Seidl, T. Schwentick and A. Muscholl: *Numerical Document Queries*, Proc. of 22nd PODS, San Diego (California), pp. 155–166, ACM Press, 2003.
26. T. Takai, Y. Kaji and H. Seki: *Right-Linear Finite Path Overlapping Term Rewriting Systems Effectively Preserve Recognizability*, Proc. of 11th RTA, Norwich (UK), LNCS 1833, pp. 246–260, Springer, 2000.
27. J.W. Thatcher: *Characterizing Derivation Trees of Context-Free Grammars Through a Generalization of Automata Theory*, Journal of Computer and System Sciences 1(4), pp. 317–322, Elsevier, 1967.
28. K.N. Verma: *Two-Way Equational Tree Automata for AC-Like Theories: Decidability and Closure Properties*, Proc. of 14th RTA, Valencia (Spain), LNCS 2706, pp. 180–197, Springer, 2003.
29. K.N. Verma: *On Closure under Complementation of Equational Tree Automata for Theories Extending AC*, Proc. of 10th LPAR, Almaty (Kazakhstan), LNCS 2850, pp. 183–197, Springer, 2003.
30. *Extensible Markup Language (XML) Schema 1.0*, 3rd edition, W3C, 2004. Document available at URL `http://www.w3.org/TR/2004/REC-xml-20040204/`

# A Proofs

**Proof of Remark 1.** For the class of equational tree automata, Lemma 9 hold.

**Lemma 9.** *If $R = \varnothing$, $\mathcal{L}(\mathcal{A}) = \mathcal{E}(\mathcal{L}(\mathcal{A}'))$ where $\mathcal{A}'$ is obtained by replacing $E$ in $\mathcal{A}$ with $\varnothing$.*

One should note that this does not hold if we take the following syntax

$$f(\alpha_1, \ldots, \alpha_n) \to \beta \qquad \alpha \to \beta$$

as the definition of transitions rules.

We recall the proof by Verma [29]. Suppose for leading to the contradiction that there exists an automaton $\mathcal{B}$ with the above $R$ and $E$ in Remark 1, such that $\mathcal{B}$ accepts $\mathcal{L}(\mathcal{A}_2)$. Let $\mathcal{B}'$ be the tree automaton obtained by replacing $E$ in $\mathcal{B}$ with $\varnothing$, then by the above lemma, $\mathcal{L}(\mathcal{B}) = \mathcal{E}(\mathcal{L}(\mathcal{B}'))$. This implies that for every $i, j$ with $i \neq j$, there exists a term $t$ which is accepted by $\mathcal{B}'$ and $t$ is equivalent to

$$\mathsf{f}[\,\mathsf{h}(\mathsf{g}^i(\mathsf{a})), \ldots, \mathsf{h}(\mathsf{g}^i(\mathsf{a})), \mathsf{h}(\mathsf{g}^j(\mathsf{a})), \ldots, \mathsf{h}(\mathsf{g}^j(\mathsf{a}))\,]$$

modulo $\mathsf{AC}(\mathsf{f})$. Here $\mathsf{f}[\,]$ represents a context consisting only of $\mathsf{f}$. From Pigeonhole Principle, there exist $i$ and $j$ such that $i \neq j$ and $\{\, \alpha \in Q \mid \mathsf{g}^i(\mathsf{a}) \to^*_{\mathcal{B}'} \alpha(\mathsf{g}^i(\mathsf{a}))\,\} = \{\, \alpha \in Q \mid \mathsf{g}^j(\mathsf{a}) \to^*_{\mathcal{B}'} \alpha(\mathsf{g}^i(\mathsf{a}))\,\}$. Hence, the above term is accepted by $\mathcal{B}'$ if and only if $\mathsf{f}[\,\mathsf{h}(\mathsf{g}^i(\mathsf{a})), \ldots, \mathsf{h}(\mathsf{g}^i(\mathsf{a})), \mathsf{h}(\mathsf{g}^i(\mathsf{a})), \ldots, \mathsf{h}(\mathsf{g}^i(\mathsf{a}))\,]$ is accepted. □

**Proof of Lemma 1.** First we define the mapping $\mathsf{rem}$ for terms in $T(F \cup Q)$ such that

$$\mathsf{rem}(t) = \begin{cases} f(\mathsf{rem}(t_1), \ldots, \mathsf{rem}(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \text{ for some } f \in F, \\ \mathsf{rem}(t') & \text{if } t = \alpha(t') \text{ for some } \alpha \in Q. \end{cases}$$

This mapping strips off all state symbols from the input. By structural induction, we can prove that $\mathsf{rem}(t) \in \mathrm{NF}_F(\mathcal{R})$ implies $t \in \mathrm{NF}_F(\mathcal{R})$. Next we prove this lemma and additionally the property that $\mathsf{rem}(t') =_{\mathcal{E}} (t)\!\downarrow^!_{\mathcal{R}}$. We use induction on the length of $\to^+_{\mathcal{A}}$. The base case is easy, because $\to_{\mathcal{A}}$ and $\to^!_{\mathcal{R}} \cdot \to_{\mathcal{A}'}$ coincide by definition. Moreover, in this case, $(t)\!\downarrow^!_{\mathcal{R}}$ and $t'$ can be represented as $(t)\!\downarrow^!_{\mathcal{R}} =_{\mathcal{E}} C[c]$ and $t' =_{\mathcal{E}} C[\alpha(c)]$ for some context $C$ over $F$, constant $c$ from $F$ and state $\alpha$ in $Q$. If $t' \notin \mathrm{NF}_F(\mathcal{R})$, then $C[\alpha(c)] \notin \mathrm{NF}_F(\mathcal{R})$, that implies $C[c] \notin \mathrm{NF}_F(\mathcal{R})$, but it contradicts to the assumption. For the induction step, we assume $t \to^+_{\mathcal{A}} t'' \to_{\mathcal{A}} t'$. By induction hypothesis, $(t)\!\downarrow^!_{\mathcal{R}} \to^*_{\mathcal{A}'} t''$ and $\mathsf{rem}(t'') =_{\mathcal{E}} (t)\!\downarrow^!_{\mathcal{R}}$, and then $t'' \in \mathrm{NF}_F(\mathcal{R})$. Thus, $(t)\!\downarrow^!_{\mathcal{R}} \to^*_{\mathcal{A}'} t'' \to_{\mathcal{A}'} t'$. Regarding the transition $t'' \to_{\mathcal{A}'} t'$, we have the two cases:

(1) $t'' =_{\mathcal{E}} C'[f(\alpha_1(u_1), \ldots, \alpha_n(u_n))]$ and $t' =_{\mathcal{E}} C'[\beta(f(u_1, \ldots, u_n))]$ for some $f(\alpha_1(x1), \ldots, \alpha_n(x_n)) \to \beta(f(x_1, \ldots, x_n)) \in \Delta$,

(2) $t'' =_{\mathcal{E}} C'[\alpha(u)]$ and $t' =_{\mathcal{E}} C'[\beta(u)]$ for some $\alpha(x) \to \beta(x) \in \Delta$.

Here $C'$ is a context over $F \cup Q$, $u_1, \ldots, u_n, u$ are terms over $F \cup Q$, and $\alpha_1, \ldots, \alpha_n, \alpha, \beta$ are states in $Q$. In any case, by induction hypothesis, since $\mathsf{rem}(t'') =_{\mathcal{E}} (t)\!\downarrow^!_{\mathcal{R}}$, we have $\mathsf{rem}(t') =_{\mathcal{E}} (t)\!\downarrow^!_{\mathcal{R}}$, because $\mathsf{rem}(t'') =_{\mathcal{E}} \mathsf{rem}(t')$. □

**Proof of Theorem 3.** Given tree automata $\mathcal{A}_1$ and $\mathcal{A}_2$ with $\mathcal{R}$-normalization. From assumption, we have an equational tree automaton $\mathcal{B}$ with the same equational theory which accepts $\mathrm{NF}_F(\mathcal{R})$. From monotonicity of closure operators, we obtain

$$\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2) \quad \Leftrightarrow \quad \mathcal{L}(\mathcal{A}'_1) \cap \mathcal{L}(\mathcal{B}) \subseteq \mathcal{L}(\mathcal{A}'_2) \cap \mathcal{L}(\mathcal{B})$$

where $\mathcal{A}'_i$ for $i \in \{1, 2\}$ is an equational tree automaton define by replacing $R$ in $\mathcal{A}_i$ with $\varnothing$ □

**Proof of Lemma 2.** Let $\mathcal{A} = (\mathcal{R}, Q, Q_{\mathrm{fin}}, \Delta)$. From assumption, $(t)\downarrow^!_{\mathcal{R}}$ is unique up to modulo $\mathcal{E}$. We take $\mathcal{A}' = ((\mathcal{E}, \varnothing), Q, Q_{\mathrm{fin}}, \Delta)$. From Lemma 1, we have

$$t \in \mathcal{L}(\mathcal{A}) \;\Leftrightarrow\; (t)\downarrow^!_{\mathcal{R}} \in \mathcal{L}(\mathcal{A}') \;\Leftrightarrow\; (t)\downarrow^!_{\mathcal{R}} \notin (\mathcal{L}(\mathcal{A}'))^{\mathsf{c}}.$$

The second equivalence is obtained from the confluence of $\mathcal{R}$. From assumption, there effectively exists an equational tree automaton $\mathcal{B} = ((\mathcal{E}, \varnothing), Q', Q'_{\mathrm{fin}}, \Delta')$ such that $\mathcal{L}(\mathcal{B}) = (\mathcal{L}(\mathcal{A}'))^{\mathsf{c}}$. Therefore, $((\mathcal{E}, R), Q', Q'_{\mathrm{fin}}, \Delta')$ accepts $t$ if and only if $\mathcal{A}$ does not accept $t$. □

**Proof: Why strong normalization for Theorem 1?** We use the typical reduction from PCP (Post's correspondence problem). Given an instance of the PCP $P = \{ (u_i, v_i) \mid 1 \leqslant i \leqslant n \colon u_i, v_i \in \{ \mathsf{a}, \mathsf{b} \}^+ \}$, the set $R_P$ consists of the rewrite rules

$$\mathsf{f}(x, y) \to \mathsf{f}(\overline{u_i}(x), \overline{v_i}(y)) \qquad \text{for each } (u_i, v_i) \in P$$

and

$$\mathsf{f}(\mathsf{a}(x), \mathsf{a}(y)) \to \mathsf{g}(\mathsf{a}(x), \mathsf{a}(y)) \qquad \mathsf{f}(\mathsf{b}(x), \mathsf{b}(y)) \to \mathsf{g}(\mathsf{b}(x), \mathsf{b}(y))$$

$$\mathsf{g}(\mathsf{a}(x), \mathsf{a}(y)) \to \mathsf{g}(x, y) \quad \mathsf{g}(\mathsf{b}(x), \mathsf{b}(y)) \to \mathsf{g}(x, y)$$

$$\mathsf{g}(\mathsf{c}, \mathsf{c}) \to \mathsf{d} \qquad \mathsf{f}(x, y) \to \mathsf{e}$$

where $\overline{u}(t) = \overline{u'}(\mathsf{a}(t))$ if $u = \mathsf{a}\,u'$ and $\overline{u}(t) = \overline{u'}(\mathsf{b}(t))$ if $u = \mathsf{b}\,u'$; otherwise, $\overline{u}(t) = t$. Let $\mathcal{R} = ((\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{f}, \mathsf{g}\}, \varnothing), R_P)$ and $\mathcal{A}_P = (\mathcal{R}, \{ \alpha \}, \{ \alpha \}, \{ \mathsf{d} \to \alpha(\mathsf{d}) \})$. Observe that $\mathcal{R}$ is weakly normalizing, because every term of the form $\mathsf{f}(t_1, t_2)$ can be rewritten to $\mathsf{e}$, and $\mathcal{R}$ is strongly normalizing over $\mathcal{T}(F - \{ \mathsf{f} \})$. From Lemma 1, we can see that $\mathsf{f}(\mathsf{c}, \mathsf{c}) \in \mathcal{L}(\mathcal{A}_P)$ if and only if $(\mathsf{f}(\mathsf{c}, \mathsf{c}))\downarrow^!_{\mathcal{R}} = \mathsf{d}$. Trivially, the latter statement holds if and only if $P$ has a solution. □

**Proof of Theorem 5.** It is obvious that this class of tree languages is closed under union. For the intersection, we take an equational rewriting system $\mathcal{R} = (\mathcal{E}, R)$ with $\mathcal{E} = (F, E)$, and tree automata $\mathcal{A}$ and $\mathcal{B}$ modulo $\mathcal{R}$-normalization. Define $\mathcal{A}'$ and $\mathcal{B}'$ to be equational tree automata, each of which is obtained by replacing $R$ with $\varnothing$, respectively for $\mathcal{A}$ and $\mathcal{B}$. Then, from assumption, there exists $\mathcal{C} = ((\mathcal{E}, \varnothing), Q, Q_{\mathrm{fin}}, \Delta)$ such that $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{A}') \cap \mathcal{L}(\mathcal{B}')$. For every $t \in \mathcal{T}(F)$, $(t)\downarrow^!_{\mathcal{R}}$ is unique, so $(t)\downarrow^!_{\mathcal{R}} \in \mathcal{L}(\mathcal{A}')$ and $(t)\downarrow^!_{\mathcal{R}} \in \mathcal{L}(\mathcal{B}')$ if and only if $t \in \mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$. Thus, $((\mathcal{E}, R), Q, Q_{\mathrm{fin}}, \Delta)$ accepts $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$. □

**Proof of Lemma 6.** Without loss of generality we assume that every production rule in $\Lambda$ is in the shape of $\alpha \to \mathsf{a}\beta$ or $\alpha \to \mathsf{a}$. Hereafter in the proof, we show

the tree automata construction over the signature $F = \Sigma \cup \{ \mathsf{f} \}$ with $\mathsf{arity}(\mathsf{f}) = 2$. We define the tree automaton $\mathcal{A} = (F, Q, Q_{\mathrm{fin}}, \Delta)$ associated to $\mathcal{G}$ as follows:

$$
\begin{aligned}
F &: \Sigma \cup \{ \mathsf{f} \} \text{ such that } \mathsf{arity}(a) = 0 \text{ for all } a \in \Sigma \\
Q &: \{ \mathsf{q}_{\alpha \to \beta} \mid \alpha, \beta \in \mathcal{S} \} \cup \{ \mathsf{q}_\alpha \mid \alpha \in \mathcal{S} \} \\
Q_{\mathrm{fin}} &: \{ \mathsf{q}_{\mathsf{s}_0} \} \\
\Delta &: \{ \mathsf{f}(\mathsf{q}_{\alpha \to \beta}(x), \mathsf{q}_\beta(y)) \to \mathsf{q}_\alpha(\mathsf{f}(x, y)) \mid \alpha, \beta \in \mathcal{S} \} \cup \\
&\quad \{ \mathsf{f}(\mathsf{q}_{\alpha \to \beta}(x), \mathsf{q}_{\beta \to \gamma}(y)) \to \mathsf{q}_{\alpha \to \gamma}(\mathsf{f}(x, y)) \mid \alpha, \beta, \gamma \in \mathcal{S} \} \cup \\
&\quad \{ a \to \mathsf{q}_{\alpha \to \beta}(a) \mid \alpha \to a\beta \in \Lambda \text{ for some } \alpha, \beta \in \mathcal{S} \text{ and } a \in \Sigma \} \cup \\
&\quad \{ a \to \mathsf{q}_\alpha(a) \mid \alpha \to a \in \Lambda \text{ for some } \alpha \in \mathcal{S} \text{ and } a \in \Sigma \}
\end{aligned}
$$

Then we can show by structural induction that $t$ is accepted by $\mathcal{A}$ if and only if $\mathsf{leaf}(t)$ is generated by $\mathcal{G}$. By construction of $\Delta$, we observe that for all $w \in \Sigma^*$, $a \in \Sigma$ and $\alpha, \beta \in \mathcal{S}$, (1) $\alpha \to_{\mathcal{G}} a$ if and only if $a \to_{\mathcal{A}} \mathsf{q}_\alpha(a)$, and (2) $\alpha \to_{\mathcal{G}}^* w\beta$ if and only if $u \to_{\mathcal{A}}^* \mathsf{q}_{\alpha \to \beta}(u)$ and $\mathsf{leaf}(u) = w$. □

**Proof: Why Lemma 6 does not hold if $\mathcal{G}$ is context-free grammar?** We consider the language $L = \{ \mathsf{a}^n \mathsf{b}^n \mid n \geqslant 1 \}$ which is generated by a context-free grammar. We suppose for leading the contradiction that there is a tree automaton $\mathcal{A} = (F, Q, Q_{\mathrm{fin}}, \Delta)$ whose leaf language is $L$. The signature $F$ must contain a function symbol whose arity is greater than 1. But since $F$ is finite, trees accepted by $\mathcal{A}$ do not have an upper-bound in the depth. In fact, when we take a natural number $k$ which is *large* enough to apply Pumping Lemma, there exists a tree $C[t]$ whose subtree $t$ satisfies that $\mathsf{depth}(t) > |Q|$, $\mathsf{leaf}(t) = \mathsf{b}^+$ and $\mathsf{length}(\mathsf{leaf}(t)) > |Q|$. Then $\mathcal{A}$ accepts $C[t']$ whose subtree $t'$ has less leaves than that of $t$. Because $\mathsf{leaf}(C[t'])$ is no longer a word in $L$, it leads to the contradiction. □

**Proof of Lemma 7.** We define the grammar $\mathcal{G} = (\Sigma, \mathsf{s}_0, \mathcal{S}, \Lambda)$ associated to $\mathcal{A}$ as follows:

$$
\begin{aligned}
\Sigma &: \{ f \in F \mid \mathsf{arity}(f) = 0 \} \\
\mathcal{S} &: Q \cup \{ \mathsf{s}_0 \} \\
\Lambda &: \left\{ \beta \to c_1 \dots c_n \alpha \;\middle|\; \begin{array}{l} f(c_1, \dots, c_n, \alpha(x)) \to_{\mathcal{A}}^* \beta(f(c_1, \dots, c_n, x)) \\ \quad \text{for some } f, c_1, \dots, c_n \in F \text{ and } \alpha, \beta \in Q \end{array} \right\} \cup \\
&\quad \left\{ \beta \to c \;\middle|\; c \to_{\mathcal{A}}^* \beta(c) \text{ for some } c \in F \text{ and } \beta \in Q \right\} \cup \\
&\quad \left\{ \mathsf{s}_0 \to \alpha \;\middle|\; \alpha \in Q_{\mathrm{fin}} \right\}
\end{aligned}
$$

Using the induction on $t$ in $\mathrm{NF}_F(\mathcal{R}_{\mathsf{reg}})$, we can prove that $t$ is accepted by $\mathcal{A}$ if and only if $\mathsf{leaf}(t)$ is generated by $\mathcal{G}$. Since $\mathcal{G}$ generates a regular language, due to Corollary 6, we can construct a tree automaton $\mathcal{B}$ that accept trees whose leaves are generated by $\mathcal{G}$. Hence $\mathcal{B}$ accepts $\{ t \mid (t)\downarrow^!_{\mathcal{R}_{\mathsf{reg}}} \text{ accepted by } \mathcal{A} \}$. □

## B Complexity

For an equational rewrite system $\mathcal{R} = (F, E, R)$, if $E = \varnothing$ and $R = \varnothing$, we write $\mathcal{A} = (F, Q, Q_{\mathrm{fin}}, \Delta)$ for $\mathcal{A} = ((F, \varnothing, \varnothing), Q, Q_{\mathrm{fin}}, \Delta)$, and we say $\mathcal{A}$ is a tree

automaton. Let us estimate the size of the tree automata and word grammar constructed in the proofs, and evaluate the computational complexity needed for the constructions.

In the proof of Lemma 7, an input tree automaton $\mathcal{A}$ is translated into a regular word grammar $\mathcal{G}$, and then a tree automaton $\mathcal{B}$ is obtained due to Corollary 6. In Proposition 1, we explain the property that a hedge automaton $\mathcal{H}$ can be translated into a tree automaton $\mathcal{A}$, and this statement is generalized in Corollary 7, as a hedge automaton $\mathcal{H}$ is translated into a tree automaton $\mathcal{B}$ via $\mathcal{A}$. Hereafter in the following, we let $\mathcal{H} = (\Sigma, C, N, N_{\mathrm{fin}}, \Delta_{\mathcal{H}})$, $\mathcal{A} = (F, Q, Q_{\mathrm{fin}}, \Delta)$, $\mathcal{B} = (F', Q', Q'_{\mathrm{fin}}, \Delta')$, and $\mathcal{G} = (\Sigma, \mathcal{S}, \mathsf{s}_0, \Lambda)$.

**Complexity in Corollary 6.** In the proof of Lemma 6, the number of constructed rules is $O(|\Lambda| + |\mathcal{S}|^3)$ where the factor $|\mathcal{S}|^3$ is due to the second line of the constructed rules $\Delta$ in the proof. When Lemma 6 is extended to Corollary 6, the number of the rules grows up to $O(|\Lambda| + |\mathcal{S}|^{N+1})$ where $N$ is the maximum arity of function symbols in $F$. Precisely, $|F'| \in O(|\Sigma|)$, $|Q'| \in O(|\mathcal{S}|^2)$, $|Q'_{\mathrm{fin}}| \in O(1)$ and $|\Delta'| \in O(|\Lambda| + |\mathcal{S}|^{N+1})$.

**Complexity in Lemma 7.** Let us look at the constructed rules $\Lambda$ in the proof of Lemma 7. The number of rules in the first line of $\Lambda$ is $O(|F|^{N-1} \times |Q|^2)$. This exponential blowing-up can be avoided by constructing a rule of the shape $\beta \to c\gamma$ ($\beta, \gamma \in Q$, $c \in F$) one by one instead of constructing a rule of the shape $\beta \to c_1 \dots c_n \alpha$. Thus, $|\Lambda| \in O(N|F||\Delta| + |Q_{\mathrm{fin}}|)$. Combining the size estimation in Corollary 6 and this observation, the size of $\mathcal{B}$ can be represented by $|F'| \in O(|\Sigma|) = O(|F|)$, $|Q'| \in O(|\mathcal{S}|^2) = O(|Q|^2)$, $|Q'_{\mathrm{fin}}| \in O(1)$ and $|\Delta'| \in O(|\Lambda| + |\mathcal{S}|^{N+1}) = O(N \times |F| \times |\Delta| + |Q_{\mathrm{fin}}| + |Q|^{N+1})$. The time complexity for the construction of $\mathcal{B}$ from $\mathcal{A}$ is also $O(N \times |F| \times |\Delta| + |Q_{\mathrm{fin}}| + |Q|^{N+1})$.

**Complexity in Proposition 1.** The construction of $|\mathcal{A}|$ in Proposition 1 is rather straightforward and $|F| \in O(|\Sigma| + |C|)$, $|Q| \in O(|N| + \sum_{\mathcal{G} \in \Delta_{\mathcal{H}}} |N_{\mathcal{G}}|)$, $|Q_{\mathrm{fin}}| \in O(|N_{\mathrm{fin}}|)$ and $|\Delta| \in O(|\Delta_{\mathcal{H}}| + \sum_{\mathcal{G} \in \Delta_{\mathcal{H}}} |\Lambda_{\mathcal{G}}|)$ where $N_{\mathcal{G}}$ (resp. $\Lambda_{\mathcal{G}}$) denotes the set of non-terminals (resp. rules) of a regular word grammar $\mathcal{G}$ appearing in a rule in $\Delta_{\mathcal{H}}$. Thus, $|\mathcal{A}| \in O(\mathcal{H})$ and the time needed for the construction of $\mathcal{A}$ is also linear in the size of $\mathcal{H}$.

**Complexity in Corollary 7.** The size of each component $\mathcal{B}$ can be estimated by those for Proposition 1 and Lemma 7 above. Note that since the signature $F_{\Sigma,C}$ of $\mathcal{A}$ constructed in the proof of Proposition 1 contains only a binary function symbol and constants, we can let $N = 2$ when we apply Lemma 7 to $\mathcal{A}$ to obtain a desirable tree automaton $\mathcal{B}$. Therefore, $|F'| \in O(\,|\Sigma| + |C|\,)$, $|Q'| \in O(\,(|N| + \sum_{\mathcal{G} \in \Delta_{\mathcal{H}}} |N_{\mathcal{G}}|)^2\,)$, $|Q'_{\mathrm{fin}}| \in O(1)$, and $|\Delta'| \in O(\,(|\Sigma| + |C|) \times (|\Delta_{\mathcal{H}}| + \sum_{\mathcal{G} \in \Delta_{\mathcal{H}}} |\Lambda_{\mathcal{G}}|) + |N_{\mathrm{fin}}| + (|N| + \sum_{\mathcal{G} \in \Delta_{\mathcal{H}}} |N_{\mathcal{G}}|)^3\,)$. Summarizing, $|\mathcal{B}| \in O(|\mathcal{H}|^3)$ and the time complexity needed for the construction of $\mathcal{B}$ is also $O(|\mathcal{H}|^3)$.