MAT Learners for Tree Series – an Abstract Data Type and Two Realizations

Frank Drewes[1], Johanna Högberg[1], and Andreas Maletti[2,⋆]

[1] Department of Computing Science, Umeå University
S–901 87 Umeå, Sweden, {drewes,johanna}@cs.umu.se
[2] Universitat Rovira i Virgili, Departament de Filologies Romàniques
Av. Catalunya 35, 43002 Tarragona, Spain, andreas.maletti@urv.cat

Abstract. We propose abstract observation tables, an abstract data type for learning deterministic weighted tree automata in Angluin's *minimal adequate teacher* model. Besides the "classical" observation table, we show that abstract observation tables can also be implemented by observation trees. The advantage of the latter is that they often require fewer queries to the teacher.

UMINF 09.22 Copyright © 2009 ISSN 0348-0542

1 Introduction

We study the problem of learning deterministic weighted (bottom-up) tree automata (dwta) [5, 7] over a semifield $\mathbb{S}$. A weighted tree automaton computes a recognizable tree series [4], i.e., a function that maps trees to values in $\mathbb{S}$. This is accomplished by assigning a weight in $\mathbb{S}$ to every transition. The weight of a computation (also called a run) is the product of all its transitions, multiplied with an additional final weight that is associated with the final state reached. Finally, the weight of a tree is the sum of the weights of all its runs. In other words, the addition of the semifield is used to handle nondeterminism. Ordinary bottom-up tree automata correspond to the special case obtained by choosing the Boolean semifield as $\mathbb{S}$. We recommend [14] for an introduction to recognizable tree series.

As mentioned above, this article is devoted to the problem of algorithmically learning dwta, also called grammatical inference. The general setting considered in grammatical inference is characterized by a series ψ (or, in the traditional case, a language) for which, a priori, no explicit representation is available. The learning algorithm (henceforth called the learner) only has access to some restricted information. Its goal is to derive, from this information, an automaton computing ψ.

There exist various learning models that make this general picture more precise. Most of them fall into one of the following categories: Gold's *learning from examples with identification in the limit* [16], Valiant's *probably approximately*

⋆ Supported by the *Ministerio de Educación y Ciencia* (MEC) grant JDCI-2007-760.

correct (PAC) learning [23], and Angluin's *query learning* [2]. In this article, we focus on the most prominent type of query learning, proposed in [1], in which the learner has access to an oracle called minimal adequate teacher (MAT). Angluin's original model was designed for learning ordinary deterministic finite automata (dfa). Its first generalization to bottom-up tree automata was proposed by Sakakibara in [21] and later improved in [12].

In the weighted case, if ψ is the target tree series, then a MAT is an oracle able to answer two types of queries. A *coefficient query* passes a tree t to the MAT and receives $\psi(t)$ as the answer. An *equivalence query* passes a dwta A, the hypothesis, to the MAT. If A computes ψ, a special token is returned. Otherwise, the answer is a counterexample, i.e., a tree t such that $A(t) \neq \psi(t)$. One of the first extensions of the MAT learner for dfa to stochastic automata appears in [9]. This algorithm is extended in [20] to certain cancellative semirings [15], which makes the algorithm applicable to string transducers. The first extension to dwta is found in [13]. This learner generalizes the learner in [12] from bottom-up tree automata to 'all-accepting' dwta, and was in turn extended to general dwta in [19]. In both cases, the weight structure is a semifield (i.e., a semiring, in which multiplicative inverses exist). All those algorithms learn deterministic devices. In [17], the first MAT learner for nondeterministic weighted tree automata over fields was presented. An overview of MAT learners for weighted and unweighted tree automata can be found in [10].

Typically, MAT learnability results are based on Myhill-Nerode-like characterizations of recognizability. In essence, the learner learns (representatives of) equivalence classes, which it refines in the light of the counterexamples provided by the teacher. This process terminates when the equivalence has converged to the Myhill-Nerode equivalence. The learners in [12, 13, 19] are all very similar. Following Angluin's original approach, they maintain an observation table, whose rows are indexed by trees representing the states and transitions of an automaton. The columns are indexed by contexts (trees with a "hole") whose purpose is to separate the discovered equivalence classes from each other. From the table, the learner repeatedly constructs a dwta consistent with the information in the table and asks an equivalence query. If the MAT accepts the dwta, then the learning process has converged. Otherwise, the counterexample received is inspected by a technique known as contradiction backtracking [22]. This will either reveal a tree not yet in the table, that represents a new transition, or a context that separates previously equivalent trees in the table, thus making the learner discover a new state.

In [18, Chapter 8], Kearns and Vazirani give a rough description of Angluin's original learner (for regular string languages), in which the observation table has been replaced by a tree-like data structure. As we shall see in Section 5, a generalized version of these observation trees can be used even in the case of dwta, thus yielding an alternative version of the learner in [19]. In Section 6, we shall see that this learner often asks fewer coefficient queries, and is thus supposed to have advantages in practical applications.

Looking at all these different but closely related learners and their correctness proofs, one cannot but realize that the same algorithms and arguments are repeated over and over again, with only slight differences. Thus, rather than cloning [19] to come up with a version using observation trees, we propose an abstract data type called abstract observation table (aot), in Section 3 of this paper. To obtain a concrete realization, one needs to implement an interface consisting of a few abstract routines that have to satisfy certain conditions. The learning algorithm itself is hidden within the aot, and its correctness is established once and for all (depending on the correct implementation of the interface, of course). In this way, the common parts of the correctness proofs of different realizations of the learner are encapsulated. What remains to be considered in each individual case is what is specific to the particular realization at hand.

We provide two such realizations. Firstly, the learner in [19] can easily be seen to be an instance of the aot. Secondly, in Section 5, we develop the one based on observation trees and give a correctness proof (which becomes easy thanks to the results of Section 3). As mentioned above, we expect observation trees to have the advantage of requiring fewer coefficient queries than observation tables, even though the theoretical best and worst cases coincide. To confirm this, we have implemented both learners and have conducted a set of experiments whose results are reported in Section 6. Finally, we show in Section 6 how to optimize the equivalence test derived from [6, Corollary 5.6] to run in linear time in the product of the number of transitions of the two dwta. The straightforward algorithm derived from [6, Corollary 5.6] runs in linear time in the product of the size of the transition tables, which could be exponentionally larger than the number of actual transitions.

In summary, the structure of this paper is the following. In the next section, the necessary preliminaries around trees, tree series, and MAT learning are gathered. In Section 3, we present the aot and prove its correctness. This is the first major contribution of this paper. In Section 4, we show briefly that the learner in [19] is an instance of the aot. Section 5 introduces the instance based on observation trees. Together with Section 6, which confirms that observation trees often use fewer coefficient queries than observation tables, this is the second major contribution of the paper. In addition, Section 6 presents the algorithm for deciding the equivalence of dwta over semifields that we have used in our experiments for the purpose of implementing the teacher. This represents the final major contribution.

2 Preliminaries

The set of natural numbers (including 0) is denoted by $\mathbb{N}$. For every $k \in \mathbb{N}$ the set $\{1, \ldots, k\}$ is denoted by $[k]$. For a set S, the set of all finite sequences (or strings) over S, including the empty sequence ϵ, is denoted by S^*. The cardinality of S and the length of $w \in S^*$ are denoted by $|S|$ and $|w|$, respectively.

The *index* of an equivalence relation $\equiv$ on a set A is the number of equivalence classes induced by $\equiv$, i.e., the cardinality of the quotient set $A/_\equiv$.

Given a function $f\colon A \to B$ and pairs $(a_1, b_1), \dots, (a_n, b_n)$, where the a_i are pairwise distinct, we let $f\langle a_1 := b_1, \dots, a_n := b_n\rangle$ denote the function $g\colon A' \to B'$ with $A' = A \cup \{a_1, \dots, a_n\}$ and $B' = B \cup \{b_1, \dots, b_n\}$ such that $g(a_i) = b_i$ for $i \in [n]$, and $g(a) = f(a)$ for all $a \in A \setminus \{a_1, \dots, a_n\}$. If f is a function with empty domain, then we also write $\langle a_1 := b_1, \dots, a_n := b_n\rangle$ instead of $f\langle a_1 := b_1, \dots, a_n := b_n\rangle$.

A (commutative) semiring $\mathbf{S} = (\mathbb{S}, +, \cdot, 0, 1)$ consists of a set $\mathbb{S}$ together with binary addition and multiplication operations $+$ and $\cdot$, respectively, as well as distinct elements $0, 1 \in \mathbb{S}$ such that (i) $(\mathbb{S}, +, 0)$ and $(\mathbb{S}, \cdot, 1)$ are commutative monoids, (ii) multiplication distributes over addition, and (iii) 0 is absorbing with respect to multiplication. The product $a_1 \cdot \ldots \cdot a_k$ of finitely many elements $a_1, \dots, a_k \in \mathbb{S}$ is denoted by $\prod_{i=1}^{k} a_i$ or equivalently $\prod_{i \in [k]} a_i$ (note that the order is irrelevant since we consider only semirings with commutative multiplication). We say that $\mathbf{S}$ is a semifield if every element $a \in \mathbb{S} \setminus \{0\}$ has a multiplicative inverse, which is denoted by a^{-1} (i.e., $a \cdot a^{-1} = 1$). Throughout the rest of this paper, we let $\mathbf{S}$ be a semifield, which we simply denote by its domain $\mathbb{S}$. In fact, since we will only consider deterministic weighted tree automata, we will not need the addition of $\mathbb{S}$. For sequences $\alpha, \beta \in \mathbb{S}^*$, we write $\alpha \sim \beta$ if β is a multiple of α; i.e., if there exists $a \in \mathbb{S} \setminus \{0\}$ such that $a \cdot \alpha = \beta$ (where multiplication is extended to a function on $\mathbb{S} \times \mathbb{S}^* \to \mathbb{S}^*$ in the obvious way). Note that $\sim$ is an equivalence relation due to the fact that $\mathbb{S}$ has multiplicative inverses.

Alphabets, Trees, and Contexts

A *ranked set* is a set $\Sigma = \bigcup_{k \in \mathbb{N}} \Sigma_k$ of labels (also called symbols) consisting of (not necessarily disjoint) subsets Σ_k. The labels in Σ_k are said to have rank k. We write $f^{(k)}$ to indicate that $f \in \Sigma_k$. The set T_Σ of all trees over Σ consists of all mappings $t\colon N \to \Sigma$ (called trees) with the following properties:

- The set N of *nodes* of t is a finite and non-empty prefix-closed subset of $\mathbb{N}^*$. Thus, for every $vw \in N$ with $v, w \in \mathbb{N}^*$, it holds that $v \in N$.
- For all $v \in N$, there is $k \in \mathbb{N}$ such that $t(v) \in \Sigma_k$ and $\{i \in \mathbb{N} \mid vi \in N\} = [k]$.

For a tree $t\colon N \to \Sigma$, we also write $\mathrm{nodes}(t)$ for N. The *height of* t is denoted by $\mathrm{hg}(t)$, i.e., $\mathrm{hg}(t) = \max\{|v| \mid v \in \mathrm{nodes}(t)\}$. Given a tree t and a node $v \in \mathrm{nodes}(t)$, the *subtree of* t *rooted at* v is denoted by t/v. It is defined by $\mathrm{nodes}(t/v) = \{w \in \mathbb{N}^* \mid vw \in \mathrm{nodes}(t)\}$ and $(t/v)(w) = t(vw)$ for all $w \in \mathrm{nodes}(t/v)$. The set $\mathrm{leaves}(t)$ of all *leaves of* t is the set

$$\mathrm{leaves}(t) = \{v \in \mathrm{nodes}(t) \mid vi \notin N \text{ for all } i \in \mathbb{N}\} \enspace .$$

We shall denote a tree t as $f[t_1, \dots, t_k]$ if $t(\epsilon) = f$, $N \cap \mathbb{N} = [k]$, and $t/i = t_i$ for all $i \in [k]$. In the special case where $k = 0$ (i.e., $\mathrm{nodes}(t) = \{\epsilon\}$), the brackets may be omitted, thus denoting t as f. For a set T of trees, the set of all trees of the form $f[t_1, \dots, t_k]$ such that $f \in \Sigma_k$ and $t_1, \dots, t_k \in T$ is denoted by $\Sigma(T)$.

We will frequently decompose a tree into a context and a subtree. For this purpose, let us reserve the special symbol $\Box$ of rank 0 (and no other rank) that does not occur in Σ. A tree $c \in \mathrm{T}_{\Sigma \cup \{\square\}}$ in which $\square$ occurs exactly once is called a *context* (over Σ). The set of all contexts over Σ is denoted by C_Σ. Given a context $c \in \mathrm{C}_\Sigma$ and a tree t, we denote by ct the concatenation of c and t at $\square$, which is obtained by substituting t for the unique leaf labelled $\square$ in c. More precisely, if $v \in \mathrm{nodes}(c)$ is the unique node such that $c(v) = \square$, then $\mathrm{nodes}(ct) = \mathrm{nodes}(c) \cup \{vw \mid w \in \mathrm{nodes}(t)\}$ with $ct(w) = c(w)$ for all $w \in \mathrm{nodes}(c) \setminus \{v\}$, and $ct/v = t$.

For every $k \in \mathbb{N}$, $t, t_1, \ldots, t_k \in \mathrm{T}_\Sigma$, and $v_1, \ldots, v_k \in \mathrm{nodes}(t)$ such that v_i is not a prefix of v_j for any $i, j \in [k]$, we denote by $t[v_1 \leftarrow t_1, \ldots, v_k \leftarrow t_k]$ the tree obtained by replacing t/v_i by t_i for every $i \in [k]$.

Tree Series and Weighted Tree Automata

A *tree series over* $\mathbb{S}$ is a mapping $\psi\colon \mathrm{T}_\Sigma \to \mathbb{S}$. Its support is

$$\mathrm{supp}(\psi) = \{t \in \mathrm{T}_\Sigma \mid \psi(t) \neq 0\} \enspace .$$

A tree $t \in \mathrm{T}_\Sigma$ is called *live* if there exists $c \in \mathrm{C}_\Sigma$ such that $ct \in \mathrm{supp}(\psi)$. Such a context c is also called a *sign of life for* t.

We now recall the definition of deterministic weighted (bottom-up finite-state) tree automata [4, 5, 7] (dwta, for short) over $\mathbb{S}$. Such a dwta is a tuple $A = (\Sigma, Q, \delta, \lambda)$ where

- Σ is the finite input ranked set,
- Q is the finite set of *states*, considered as symbols of exclusive rank 0,
- $\delta\colon \Sigma(Q) \to Q \times (\mathbb{S} \setminus \{0\})$ is the partial *transition function*, and
- $\lambda\colon Q \to \mathbb{S}$ is the *final weight mapping.*

Intuitively, $\delta(f[q_1, \ldots, q_k]) = (q, a)$ determines the behaviour of the dwta in the situation, in which it processes an occurrence of the symbol f of rank k and the k subtrees $t_1, \ldots, t_k$ of f have already been processed in states $q_1, \ldots, q_k$, respectively. Then the dwta continues in state q and charges the weight a.

The transition function δ extends to trees in a straightforward way, which yields the partial function $\hat{\delta}\colon \mathrm{T}_\Sigma(Q) \to Q \times (\mathbb{S} \setminus \{0\})$ such that $\hat{\delta}(q) = (q, 1)$ for every $q \in Q$ and, for every $f[t_1, \ldots, t_k]$ with $f \in \Sigma_k$ and $t_1, \ldots, t_k \in \mathrm{T}_\Sigma(Q)$:

- If $\hat{\delta}(t_i) = (q_i, a_i)$ for every $i \in [k]$ and $\delta(f[q_1, \ldots, q_k]) = (q, a)$ are all defined, then $\hat{\delta}(f[t_1, \ldots, t_k]) = (q, a \cdot \prod_{i=1}^{k} a_i)$.
- Otherwise, $\hat{\delta}(f[t_1, \ldots, t_k])$ is undefined.

The tree series computed by A is given as follows. For all $t \in \mathrm{T}_\Sigma$, if $\hat{\delta}(t) = (q, a)$, then $A(t) = a \cdot \lambda(q)$. Otherwise, $A(t) = 0$.

Every dwta can be made *total* without affecting the tree series computed, where a dwta A is total if δ is a total function. This is achieved by adding a non-final (i.e., final weight 0) sink state that is the target of all missing transitions (using, for example, weight 1).

Throughout the remainder of this paper, let Σ be a finite ranked set, $\psi\colon \mathrm{T}_\Sigma \to \mathbb{S}$ be a tree series, and $\mathbf{S} = (\mathbb{S}, +, \cdot, 0, 1)$ be a commutative semifield.

Let $s, t \in \mathrm{T}_\Sigma$. For $C \subseteq \mathrm{C}_\Sigma$, we write $s \equiv_C t$ if there exists $a \in \mathbb{S} \setminus \{0\}$ such that $\psi(cs) = a \cdot \psi(ct)$ for every $c \in C$. Note that $\equiv_C$ is an equivalence relation on T_Σ. The relation $\equiv_{\mathrm{C}_\Sigma}$ is simply denoted by $\equiv$.

Theorem 1 (see [5, Theorems 2 and 3]). *The tree series ψ can be computed by some dwta if and only if $\equiv$ has finite index. Moreover, any dwta computing ψ has at least $|L/_{\equiv}|$ states where $L = \{t \in \mathrm{T}_\Sigma \mid t \text{ live}\}$.*

An easy but important observation for the learning algorithms considered in this paper concerns the question how one can establish that $t \not\equiv t'$ for trees $t, t' \in \mathrm{T}_\Sigma$ having a common sign of life c. Let $\psi(ct') = a \cdot \psi(ct)$. By definition, $t \equiv t'$ is equivalent to saying that $\psi(c't') = a \cdot \psi(c't)$ for all $c' \in \mathrm{C}_\Sigma$. In other words, we have the following lemma, that we will usually make use of without explicitly mentioning this fact.

Lemma 2. *Let $t, t' \in \mathrm{T}_\Sigma$ have a common sign of life c. Then $t \equiv t'$ if and only if, for all $c' \in \mathrm{C}_\Sigma$,*

$$\frac{\psi(c't)}{\psi(ct)} = \frac{\psi(c't')}{\psi(ct')} \quad .$$

The Minimal Adequate Teacher

In the following, we will consider grammatical inference of ψ. The aim is to build a dwta computing ψ (if such a dwta exists) using an appropriately extended version of Angluin's *minimal adequate teacher* (MAT) as the source of information about ψ (cf. [1, 2, 3]).

A learning algorithm that infers a dwta computing ψ will henceforth be called a *learner*. Such a learner may ask *coefficient* and *equivalence queries* to the MAT (and the MAT will answer them correctly):

Coefficient: Given a tree $t \in \mathrm{T}_\Sigma$ (provided by the learner), what is $\psi(t)$?
Equivalence: Given a dwta A (provided by the learner), does A compute ψ? If so, the teacher returns the token $\perp$ indicating that $A = \psi$. Otherwise, a counterexample is returned; i.e., a tree $t \in \mathrm{T}_\Sigma$ such that $A(t) \neq \psi(t)$.

The algorithms presented in this paper are all supposed to have access to a MAT. In particular, this implies that these algorithms can obtain $\psi(t)$ for $t \in \mathrm{T}_\Sigma$ by asking a coefficient query. Thus, the reader should bear in mind that every mention of $\psi(t)$ in our algorithms indicates that a coefficient query is asked (unless $\psi(t)$ has already been asked for earlier, in which case a reasonable implementation would have memorized the value).

3 An abstract data type for MAT learners

In this section, we will develop an abstract data type specification, called *abstract observation table*, for MAT learners [1, 2, 3]. The commonly used 'observation table' [1, 11, 12, 13, 19] will be an instance of this specification, but in the next section we will present another data type, called *observation tree*, that avoids (in our experiments) some of the coefficient queries typically asked when the learner fills the observation table.

In essence, our abstract data type manages the two sets, S and T, which are the states and the transitions of the deterministic weighted tree automaton that we construct. Moreover, it maintains two mappings, $\mathrm{sol}\colon S \to \mathrm{C}_\Sigma$ and $\rho\colon T \to S$, which assign a sign of life and a representative. To simplify the notation (and to avoid some parentheses), we write $t\rho$ instead of $\rho(t)$ for every $t \in T$. Let us present a mathematical definition of the universal invariant of an abstract observation table.

Definition 3 (cf. [19, Definition 8]). *Let $S \subseteq T \subseteq \Sigma(S)$ be finite subsets of T_Σ, $\mathrm{sol}\colon S \to \mathrm{C}_\Sigma$, and $\rho\colon T \to S$. Then $(S, T, \mathrm{sol}, \rho)$ is an* abstract observation table *if*

1. $\mathrm{sol}(t\rho) = \square$ *for every* $t \in T \cap \mathrm{supp}(\psi)$ *(trivial sign of life for support)*
2. $\mathrm{sol}(t\rho)t \in \mathrm{supp}(\psi)$ *for every* $t \in T$ *($\mathrm{sol}(t\rho)$ witnesses that t is live)*
3. $t \not\equiv s$ *for every $t \in T$ and $s \in S \setminus \{t\rho\}$* *($t$ distinct from other states)*

In the following, we will abbreviate $\mathrm{sol}(t\rho)$ by $\mathrm{sol}(t)$ (for $t \in \mathrm{T}_\Sigma$) whenever appropriate. Thus, with this convention in mind, the first condition becomes $\mathrm{sol}(t) = \square$ for every $t \in T \cap \mathrm{supp}(\psi)$, and the second becomes $\mathrm{sol}(t)t \in \mathrm{supp}(\psi)$ for every $t \in T$.

Note that the third condition yields $s\rho = s$ for every $s \in S$ because trivially $s \equiv s$. In addition, this implies that $s_1 \not\equiv s_2$ for all distinct $s_1, s_2 \in S$. Finally, for every $t \in T$, $t \in \mathrm{supp}(\psi)$ if and only if $t\rho \in \mathrm{supp}(\psi)$, which can be seen as follows. If $t \in \mathrm{supp}(\psi)$ or $t\rho \in \mathrm{supp}(\psi)$, then $\mathrm{sol}(t) = \square$ by the first condition and thus $t \in \mathrm{supp}(\psi)$ and $t\rho \in \mathrm{supp}(\psi)$ by the second condition.

Conceptually, we note that all known variants of Angluin's original algorithm (at least those we are aware of) maintain a set of contexts for the purpose of distinguishing between the equivalence classes of trees in T. In our abstract version, the only contexts that are explicitly required to be maintained are the signs of life for the trees in T. Thus, it might, in principle, be possible to implement abstract observation tables without managing additional contexts. The difficulty is, of course, to make sure that the third condition is satisfied, because this is usually what separating contexts are used for (cf. the definition of $\equiv$).

In the following, we will use the following interface to manipulate an abstract observation table. To simplify the description of the semantic properties, let $(S, T, \mathrm{sol}, \rho)$ and $(S', T', \mathrm{sol}', \rho')$ be the abstract observation table before and after execution, respectively.

– INITIALIZE *(constructor)*
  - Post-condition: $S' = T' = \emptyset$
– ADDTRANSITION(t, c) with $t \in \Sigma(S)$ and $c \in \mathrm{C}_\Sigma$ *(add new transition)*
  - Pre-conditions: $t \notin T$ and $ct \in \mathrm{supp}(\psi)$
  - Post-conditions: $S \subseteq S'$ and $T \cup \{t\} \subseteq T'$
– ADDSTATE(t, c) with $t \in T$ and $c \in \mathrm{C}_\Sigma$ *(add new state)*
  - Pre-condition: $\dfrac{\psi(ct)}{\psi(\mathrm{sol}(t)t)} \neq \dfrac{\psi(c(t\rho))}{\psi(\mathrm{sol}(t)(t\rho))}$
  - Post-conditions: $S \cup \{t\} \subseteq S'$ and $T \subseteq T'$

In the next definition, it is shown how to construct a dwta A_{aot} from an abstract observation table $(S, T, \mathrm{sol}, \rho)$. The motivation for the pre-conditions of ADDTRANSITION and ADDSTATE is closely related to this definition. As we shall see, the trees in S will be turned into the states of A_{aot}, whereas the trees in T will give rise to its transitions. As a consequence, it will turn out that $\hat{\delta}(t)$ is undefined for trees $t \in \Sigma(S) \setminus T$. In other words, t is not live with respect to A_{aot}, whereas the second part of the precondition states that it, in fact, should be. Thus, t must be added to T. The pre-condition of ADDSTATE is motivated by the fact that A_{aot} maps t and $t\rho$ to the same state (namely $t\rho$), while the pre-condition expresses that $\mathrm{sol}(t)$ and c separate $t\rho$ from t (by Lemma 2). Hence, t should rather be mapped to a new state.

Definition 4 (cf. [5, Definition 4 and p. 9]). *Let* $\mathrm{aot} = (S, T, \mathrm{sol}, \rho)$ *be an abstract observation table. Let* $\psi_{\mathrm{aot}} \colon T \to \mathbb{S}$ *be such that for every* $t \in T$

$$\psi_{\mathrm{aot}}(t) = \frac{\psi(\mathrm{sol}(t)t)}{\psi(\mathrm{sol}(t)(t\rho))} \enspace .$$

We construct the deterministic weighted tree automaton $A_{\mathrm{aot}} = (\Sigma, S, \delta, \lambda)$ *with*

– $\lambda(s) = \psi(s)$ *for every* $s \in S$,
– $\delta(t) = (t\rho, \psi_{\mathrm{aot}}(t))$ *for every* $t \in T$, *and*
– $\delta(t)$ *is undefined for all* $t \in \Sigma(S) \setminus T$.

Let us observe some easy properties of the constructed automaton. First, note that $\psi_{\mathrm{aot}}(s) = 1$ for every $s \in S$ (as $s\rho = s$). Second, A_{aot} computes ψ on all trees of T, which we prove in the next lemma.

Lemma 5 (see [19, Lemma 12]). *Let* $\mathrm{aot} = (S, T, \mathrm{sol}, \rho)$ *be an abstract observation table. Then* $A_{\mathrm{aot}}(t) = \psi(t)$ *for every* $t \in T$.

Proof. Let $A_{\mathrm{aot}} = (\Sigma, S, \delta, \lambda)$. First, we claim that $\hat{\delta}(t) = (t\rho, \psi_{\mathrm{aot}}(t))$ for every $t \in T$. Since $T \subseteq \Sigma(S)$, we have $t = f[s_1, \ldots, s_k]$ for some $f \in \Sigma_k$ and $s_1, \ldots, s_k \in S$. By the induction hypothesis, $\hat{\delta}(s_i) = (s_i, 1)$ because $s_i\rho = s_i$ and $\psi_{\mathrm{aot}}(s_i) = 1$ for every $i \in [k]$. Since $\delta(t) = (t\rho, \psi_{\mathrm{aot}}(t))$, we obtain

$$\hat{\delta}(t) = \Bigl(t\rho, \psi_{\mathrm{aot}}(t) \cdot \prod_{i=1}^{k} 1\Bigr) = (t\rho, \psi_{\mathrm{aot}}(t)) \enspace .$$

Algorithm 1 Learn a minimal deterministic weighted tree automaton for ψ

Post-conditions: returned dwta computes ψ

```
   aot.INITIALIZE                                   // initialize data structure
2: loop
     $t \leftarrow$ EQUAL?($A_{\text{aot}}$)                        // ask equivalence query
4:   if $t = \bot$ then
       return $A_{\text{aot}}$                      // return the approved automaton
6:   else
       aot $\leftarrow$ EXTEND(aot, $t$)              // extend the data structure
```

This proves the claim. Now we can prove the statement as follows.

$$\begin{aligned} A_{\text{aot}}(t) &= \psi_{\text{aot}}(t) \cdot \lambda(t\rho) = \frac{\psi(\text{sol}(t)t)}{\psi(\text{sol}(t)(t\rho))} \cdot \psi(t\rho) \\ &= \begin{cases} \dfrac{\psi(t)}{\psi(t\rho)} \cdot \psi(t\rho) & \text{if } t \in \text{supp}(\psi) \\ 0 & \text{otherwise} \end{cases} \\ &= \psi(t) \enspace , \end{aligned}$$

where the penultimate equality uses the first condition of Definition 3. □

The principal structure of the MAT learner [11, 12, 13, 19] is shown in Algorithm 1. Note that we only adapted it to work with our abstract observation table. We start with the initial empty data structure aot and iteratively query the teacher for counterexamples to our current hypothesis (the current deterministic weighted tree automaton A_{aot}), which is constructed from the current abstract observation table (see Definition 4). We update our abstract data structure with the returned information (using EXTEND), and if the teacher eventually approves our dwta, then we simply return it. We say that an algorithm works *correctly* if whenever the pre-conditions are met at the beginning of the algorithm, then (i) the algorithm terminates and (ii) the post-conditions hold at the point of return. For the next statements, we additionally assume that a correct implementation of our abstract observation table is used.

Theorem 6 (see [19, Theorem 13]). *If* EXTEND *works correctly (see the pre- and post-conditions given in Algorithm 2) and ψ can be computed by some dwta, then Algorithm 1 terminates and returns a minimal dwta computing ψ.*

Proof. Suppose that ψ can be computed by some dwta. Then $\equiv$ has finite index by Theorem 1. Let $n = |\mathrm{T}_\Sigma/_\equiv|$. Clearly, by the third condition in Definition 3, the set S of an abstract observation table (S, T, sol, ρ) contains at most n elements. Trivially, EXTEND is always called with a counterexample, because the counterexample is provided by the teacher. Since $|S|$ and $|T|$ are uniformly bounded and each call to EXTEND increases $|S| + |T|$, there can only be finitely many calls to EXTEND, which yields that Algorithm 1 terminates. Moreover, the returned dwta A_{aot} was approved by the teacher, so A_{aot} trivially computes ψ.

Algorithm 2 Function EXTEND(t) for $\text{aot} = (S, T, \text{sol}, \rho)$

```
Pre-conditions: $t \in \mathrm{T}_\Sigma$ with $A_{\text{aot}}(t) \neq \psi(t)$
Post-conditions: return an abstract observation table $\text{aot}' = (S', T', \text{sol}', \rho')$ such
    that $S \subseteq S'$ and $T \subseteq T'$ and one inclusion is strict

    Decompose $t$ into $t = cu$ where $c \in \mathrm{C}_\Sigma$ and $u \in \Sigma(S) \setminus S$
 2: if $u \notin T$ then
        return  aot.ADDTRANSITION($u, c$)        // $u$ not reachable so far; add transition
 4: else if $\frac{\psi(cu)}{\psi(\text{sol}(u)u)} \neq \frac{\psi(c(u\rho))}{\psi(\text{sol}(u)(u\rho))}$ then
        return  aot.ADDSTATE($u, c$)                                // add new state $u$
 6: else
        return  EXTEND(aot, $c(u\rho)$)                       // normalize and continue
```

By the construction of A_{aot} (see Definition 4), we know that it has at most n states. Since all states (recall that they are trees) of A_{aot} are live, this shows that it is a minimal dwta computing ψ by Theorem 1. □

Finally, let us discuss the function EXTEND, which is displayed in Algorithm 2. Given the counterexample, we search for a minimal subtree of it that is still a counterexample, using a technique called *contradiction backtracking* [22]. Let $\text{aot} = (S, T, \text{sol}, \rho)$ be the abstract observation table and $t \in \mathrm{T}_\Sigma$ be the counterexample; i.e., a tree t such that $A_{\text{aot}}(t) \neq \psi(t)$. We decompose t into a context $c \in \mathrm{C}_\Sigma$ and a tree u that is itself not in S but whose direct subtrees are all in S. In some sense, this is a minimal subtree that could possibly be offending, because A_{aot} computes the correct coefficient on all trees in T by Lemma 5. Moreover, such a subtree must exist, because $t \notin S$ (since t is a counterexample).

Now, we distinguish two cases. If u was already seen (i.e., $u \in T$), then by Lemma 5, A_{aot} returns $\psi(u)$ if applied to u. Thus an error is made when processing the context c. To this end, we test whether the context c separates u and $u\rho$; the latter is the state that represents u. Provided that c does not distinguish between u and $u\rho$, then we continue our search for an error with the simplified counterexample $c(u\rho)$. In the other cases, either u and $u\rho$ could be separated or u was not seen before. Thus, we either add u as a new state (in the former case) or as a new transition (in the latter case). Thus the post-condition of the algorithm is trivially met.

It is clear that the pre-conditions of aot.ADDSTATE and aot.ADDTRANSITION are met as well. It remains to prove that the recursive call of EXTEND meets the pre-conditions of EXTEND. To this end, we need to prove that $c(u\rho)$ is also a counterexample in line 7. This is shown in the next lemma.

Lemma 7 (cf. [19, Lemma 16]). *Let* $\text{aot} = (S, T, \text{sol}, \rho)$ *be an abstract observation table,* $t \in T$*, and* $c \in \mathrm{C}_\Sigma$ *such that*

$$\frac{\psi(ct)}{\psi(\text{sol}(t)t)} = \frac{\psi(c(t\rho))}{\psi(\text{sol}(t)(t\rho))} \quad . \tag{1}$$

If $A_{\text{aot}}(ct) \neq \psi(ct)$*, then also* $A_{\text{aot}}(c(t\rho)) \neq \psi(c(t\rho))$*.*

Proof. Let $A_{\text{aot}} = (\Sigma, S, \delta, \lambda)$. By the claim in the proof of Lemma 5 it follows that $\hat{\delta}(t) = (t\rho, \psi_{\text{aot}}(t))$ and $\hat{\delta}(t\rho) = (t\rho, 1)$ because $t \in T$. Trivially, the former yields that $\hat{\delta}(ct)$ is defined if and only if $\hat{\delta}(c(t\rho))$ is defined. Moreover, if they are defined, then there exist $s \in S$ and $a \in \mathbb{S} \setminus \{0\}$ such that $\hat{\delta}(ct) = (s, a \cdot \psi_{\text{aot}}(t))$ and $\hat{\delta}(c(t\rho)) = (s, a)$. Now we distinguish three cases:

- First, let $ct \notin \text{supp}(A_{\text{aot}})$. Then clearly also $c(t\rho) \notin \text{supp}(A_{\text{aot}})$, because $A_{\text{aot}}(t) = \psi(t) \neq 0$. Since ct is a counterexample, we have $ct \in \text{supp}(\psi)$ and thus also $c(t\rho) \in \text{supp}(\psi)$ by (1), which proves that $A_{\text{aot}}(c(t\rho)) \neq \psi(c(t\rho))$.
- Second, let $ct \notin \text{supp}(\psi)$. Since ct is a counterexample by assumption, we obtain that $ct \in \text{supp}(A_{\text{aot}})$ and thus also $c(t\rho) \in \text{supp}(A_{\text{aot}})$. Moreover, equation (1) yields that $c(t\rho) \notin \text{supp}(\psi)$, which again proves that $A_{\text{aot}}(c(t\rho)) \neq \psi(c(t\rho))$.
- Third, let $ct \in \text{supp}(A_{\text{aot}}) \cap \text{supp}(\psi)$. By the same reasoning as in the previous cases, this yields that $c(t\rho) \in \text{supp}(A_{\text{aot}}) \cap \text{supp}(\psi)$. By the observation above,

$$\begin{aligned} A_{\text{aot}}(ct) = a \cdot \psi_{\text{aot}}(t) \cdot \lambda(s) &= \frac{\psi(\text{sol}(t)t)}{\psi(\text{sol}(t)(t\rho))} \cdot A_{\text{aot}}(c(t\rho)) \\ &= \frac{\psi(ct)}{\psi(c(t\rho))} \cdot A_{\text{aot}}(c(t\rho)) \end{aligned}$$

  by (1). Since all factors are nonzero, we obtain

$$\frac{A_{\text{aot}}(ct)}{\psi(ct)} = \frac{A_{\text{aot}}(c(t\rho))}{\psi(c(t\rho))} \enspace .$$

  By assumption, the left-hand side is different from 1, which proves that $A_{\text{aot}}(c(t\rho)) \neq \psi(c(t\rho))$. □

Consequently, the recursive call of EXTEND is correct. An easy size argument (counting the subtrees of t that are not in T) can be used to show that the recursion terminates (see [13, Lemma 5.3]). Thus we obtain the main statement of this section.

Corollary 8 (of Theorem 6). *If ψ can be computed by some dwta, then Algorithm 1 terminates and returns a minimal dwta computing ψ.*

4 Observation Tables

Let us briefly present the "classical" implementation of our abstract observation table: the 'observation table'. Several similar implementations exist; the one presented here corresponds to the one in [19].

Definition 9 (see [19, Definition 8]). *Let $T \subseteq \Sigma(T)$ and $C \supseteq \{\Box\}$ be finite subsets of T_Σ and C_Σ, respectively. An* observation table *is a $(T \times C)$-matrix $\boldsymbol{P}$ with $\boldsymbol{P}(t, c) = \psi(ct)$ for every $t \in T$ and $c \in C$ such that, for every $t \in T$, there exists $c \in C$ with $\boldsymbol{P}(t, c) \neq 0$.*

*Given a set $S \subseteq T$, the pair $(S, \boldsymbol{P})$ is an S-*observation table*, if*

– *$S \subseteq \Sigma(S)$,*
– *$s_1 \not\equiv_C s_2$ for all $s_1, s_2 \in S$, and* (no $\equiv_C$-equivalent rows in S)
– *for every $t \in T$ there exists $s \in S$ such that $t \equiv_C s$.* (no new rows in T)

Note that $t \equiv_C t'$ for trees $t, t' \in T$ means that the row indexed by t is a multiple of the one indexed by t' (by a nonzero factor). It has essentially been shown in [12, 13, 19] that every observation table $\boldsymbol{P}$ can be turned into an S-observation table by choosing an appropriate set $S \subseteq T$, and that the operations of our abstract observation table can be implemented with the help of observation tables. Let us quickly show how this works.

Lemma 10. *Observation tables implement abstract observation tables.*

Proof. Let $(S, \boldsymbol{P}\colon T \times C \to \mathbb{S})$ be an S-observation table. The *abstract observation table $(S, T', \mathrm{sol}, \rho)$ represented by* $(S, \boldsymbol{P})$ is given follows:

– $T' = T \cap \Sigma(S)$,
– for every $s \in S$,

$$\mathrm{sol}(s) = \begin{cases} \Box & \text{if } s \in \mathrm{supp}(\psi) \\ c & \text{otherwise, for some } c \in C \text{ such that } cs \in \mathrm{supp}(\psi) \end{cases}$$

(by Definition 9, such an element exists for every $s \in S$), and
– $t\rho = s$ where $s \in S$ is such that $t \equiv_C s$ (by the second and third condition for S-observation trees s exists and is unique).

To verify the conditions of our abstract observation table, let $t \in T'$.

1. If $t \in \mathrm{supp}(\psi)$ then $t\rho \in \mathrm{supp}(\psi)$ because $t \equiv_C t\rho$. Consequently, $\mathrm{sol}(t) = \Box$.
2. By definition, $\mathrm{sol}(t)(t\rho) \in \mathrm{supp}(\psi)$. Again $t \equiv_C t\rho$ and since $\mathrm{sol}(t) \in C$, we obtain $\mathrm{sol}(t)t \in \mathrm{supp}(\psi)$.
3. Let $s \in S \setminus \{t\rho\}$. By the definition of ρ and the first condition of Definition 9, $t \equiv_C t\rho$ and $t\rho \not\equiv_C s$. Since $\equiv_C$ is an equivalence relation and $\equiv \subseteq \equiv_C$ (see remarks below [19, Definition 7]), this yields $t \not\equiv_C s$ and $t \not\equiv s$.

It remains to define INITIALIZE, ADDTRANSITION, and ADDSTATE. Of course, INITIALIZE returns $(\emptyset, \boldsymbol{P}_{\boldsymbol{\epsilon}})$, where $\boldsymbol{P}_{\boldsymbol{\epsilon}}$ is the empty matrix (i.e., $T = C = \emptyset$). The function ADDTRANSITION simply adds t to T and c to C (and extends $\boldsymbol{P}$ by means of coefficient queries).[3] If necessary, it *completes* S by adding elements of $T \cup \{t\}$ to it until the third condition of Definition 9 is fulfilled. Similarly, ADDSTATE adds t to S and c to C, updates $\boldsymbol{P}$, and completes S. For both ADDTRANSITION and ADDSTATE, it is straightforward to check that the resulting pair $(S', \boldsymbol{P}')$ is an S'-observation table, and that the abstract observation table it represents fulfills the post-condition of the respective function. $\Box$

[3] In fact, C can be left unchanged if it already contains a sign of life for t.

5 Observation Trees

We are now going to show that the abstract data type proposed in the previous section can alternatively be implemented by an *observation tree.* For MAT learning of regular string languages, this idea has roughly been described earlier by Kearns and Vazirani in [18, Chapter 8]. The expected advantage of observation trees over observation tables is that they require a smaller number of coefficient queries to be asked. This is important if we want to make practical use of MAT learners for recognizable tree series, because such uses normally require an (exact or approximate) simulation of the teacher, which means that the complexity of answering coefficient and equivalence queries cannot be neglected.

To understand the idea behind observation trees, it is useful to have a look at Definition 9 and the proof of Lemma 10. Intuitively, the major purpose of the matrix $\boldsymbol{P}$ is to be able to guarantee that condition (iii) of Definition 3 holds. In other words, the collected contexts provide explict evidence that trees t, t' with $t\rho \neq t'\rho$ belong to distinct congruence classes. Suppose that, at some stage of the algorithm, there are trees $t, t' \in T$ such that $t\rho = s = t'\rho$, but the teacher provides the learner with a counterexample that, via EXTEND, reveals a separating context c. The addition of c to the table divides the set $T' = \rho^{-1}(s)$ into subsets $T'_1, \ldots, T'_k$ with $k \geq 2$. The addition of c may also subdivide some of the other sets $\rho^{-1}(s')$ with $s' \in S \setminus \{s\}$ as a side effect. Although this side effect is welcomed (because it speeds up convergence), it has the disadvantage of forcing us to query the teacher for all the coefficients $\psi(ct')$ with $t' \in T$. To avoid the latter, we may organize our data in a tree, where the internal nodes are contexts and the leaves are the sets in $T/_{\ker(\rho)}$.[4] In the situation considered above, when the new context c has been discovered, the leaf T' would be replaced with $c[T'_1, \ldots, T'_k]$. Then, only the coefficients $\psi(ct')$ for all $t' \in T'$ need to be asked for.

Formally, let Ω be the infinite ranked set such that $\Omega_0 = \text{fin}(\mathrm{T}_\Sigma)$ and $\Omega_k = \mathrm{C}_\Sigma$ for every $k \geq 1$. For a tree $\tau \in \mathrm{T}_\Omega$ and $v \in \text{nodes}(\tau) \setminus \text{leaves}(\tau)$, we let $\mathrm{C}_\tau(v)$ denote the set of contexts on the path from the root of τ to v, including the latter. In other words, if $v_1 = \epsilon, \ldots, v_n = v$ are the prefixes of v, then $\mathrm{C}_\tau(v) = \{\tau(v_1), \ldots, \tau(v_n)\}$. Below, we also use the notation $T(\tau)$ to designate the set $\bigcup_{v \in \text{leaves}(\tau)} \tau(v)$.

Definition 11. *A tree $\tau \in \mathrm{T}_\Omega$ is an* observation tree *if*

1. $\tau(\epsilon) \in \{\square, \emptyset\}$,
2. *for all $v \in \text{nodes}(\tau) \setminus \text{leaves}(\tau)$, if $\tau/v = c[\tau_1, \ldots, \tau_k]$, then*

$$T(\tau/v)/_{\mathrm{C}_\tau(v)} = \{T(\tau_1), \ldots, T(\tau_k)\} \ , \text{ and}$$

3. *for all $v \in \text{leaves}(\tau)$ and $t \in \tau(v)$, $\mathrm{C}_\tau(v)$ contains a sign of life for t.*

*Given a set S such that $S \subseteq T(\tau) \subseteq \Sigma(S)$, the pair (S, τ) is an S-*observation tree *if $\tau = \emptyset$ or $|\tau(v) \cap S| = 1$ for all $v \in \text{leaves}(\tau)$.*

[4] $T/_{\ker(\rho)}$ denotes the quotient of T under the equivalence $\{(t, t') \in T^2 \mid t\rho = t'\rho\}$. For technical convenience, we let $T/_{\ker(\rho)} = \{\emptyset\}$ in the special case where $T = \emptyset$.

Algorithm 3 Function $\textsc{addTransition}(t, c)$ for an abstract observation table $(S, T, \mathrm{sol}, \rho)$ represented by (S, τ)

Pre-conditions: $t \in \Sigma(S) \setminus T$ and $c \in \mathrm{C}_\Sigma$ with $ct \in \mathrm{supp}(\psi)$
Post-conditions: return an S'-observation tree (S', τ') such that $S \subseteq S'$ and $T \cup \{t\} \subseteq T(\tau')$, representing an abstract observation table $\mathrm{aot}' = (S', T', \mathrm{sol}', \rho')$

```
    $v \leftarrow \mathrm{nod}_\tau(t)$
 2: **if** $\tau = \emptyset$ **then**
        **return** $(\{t\}, \Box[c[\{t\}]])$          // $\mathrm{aot}' = (\{t\}, \{t\}, \langle t := c' \rangle, \langle t := t \rangle)$, $c' \in \{c, \Box\}$
 4: **else if** $v \in \mathrm{leaves}(\tau)$ **then**
        **return** $(S, \tau[v \leftarrow \tau(v) \cup \{t\}])$          // $\mathrm{aot}' = (S, T \cup \{t\}, \mathrm{sol}, \rho\langle t := s \rangle)$
                                                  //          where $S \cap \tau(v) = \{s\}$
 6: **else**
        let $\tau/v = c'[\tau_1, \ldots, \tau_k]$
 8:     **if** $\{\psi(dt) \mid d \in \mathrm{C}_\tau(v)\} \neq \{0\}$ **then**
            **return** $(S \cup \{t\}, \tau[v \leftarrow c'[\tau_1, \ldots, \tau_k, \{t\}]])$     // sign of life $c$ not needed
                                                  // $\mathrm{aot}' = (S \cup \{t\}, T \cup \{t\},$
                                                  //      $\mathrm{sol}\langle t := d \rangle, \rho\langle t := t \rangle)$
10:     **else**
            **return** $(S \cup \{t\}, \tau[v \leftarrow c'[\tau_1, \ldots, \tau_k, c[\{t\}]]])$   // $\mathrm{aot}' = (S \cup \{t\}, T \cup \{t\},$
                                                  //      $\mathrm{sol}\langle t := c \rangle, \rho\langle t := t \rangle)$
```

Let us now see how observation trees can implement abstract observation tables. For this, let (S, τ) be an S-observation tree. We define the *abstract observation table* $(S, T, \mathrm{sol}, \rho)$ *represented by* (S, τ), as follows. The set T is given by $T(\tau)$. By the second condition, for every tree $t \in T$, there is a unique leaf u of τ such that $t \in \tau(u)$. Henceforth, we denote u by $\mathrm{nod}_\tau(t)$. Now, for every $s \in S$, define $\mathrm{sol}(s) = \tau(v)$ where v is the shortest prefix of $\mathrm{nod}_\tau(s)$ such that $\tau(v)$ is a sign of life for s. By the third condition, v exists, and by the first condition it is equal to ϵ (yielding $\mathrm{sol}(s) = \Box$) if $s \in \mathrm{supp}(\psi)$. Finally, the definition of ρ is straightforward: $t\rho$ is the unique element of $\tau(\mathrm{nod}_\tau(t)) \cap S$ for every $t \in T$.

It should be clear that the tuple $(S, T, \mathrm{sol}, \rho)$ constructed in this way fulfils the conditions of Definition 3. It remains to give implementations of $\textsc{initialize}$, $\textsc{addTransition}$, and $\textsc{addState}$. Unsurprisingly, $\textsc{initialize}$ returns (S, τ) with $S = \emptyset$ and $\tau = \emptyset$. The functions $\textsc{addTransition}$ and $\textsc{addState}$ are given in Algorithms 3 and 4, respectively. In their definitions, we use the following extension of nod_τ. For a tree $t \in \mathrm{T}_\Sigma$, let $\mathrm{nod}_\tau(t)$ be the maximal node $v \in \mathrm{nodes}(\tau)$ (with respect to $|v|$) such that $t \equiv_{\mathrm{C}_\tau(u)} t'$ for all $t' \in T(\tau/v)$ and all proper prefixes u of v. Note that v is uniquely determined, and that the requirement "$t \equiv_{\mathrm{C}_\tau(u)} t'$ for all $t' \in T(\tau/v)$" is equivalent to "$t \equiv_{\mathrm{C}_\tau(u)} t'$ for a $t' \in T(\tau/v)$" (both by the second condition). The latter makes it possible to find $\mathrm{nod}_\tau(t)$ efficiently. The reader should also notice that the extension of nod_τ is consistent with the earlier definition of $\mathrm{nod}_\tau(t)$ for $t \in T$.

The last case distinction in $\textsc{addTransition}$ is needed only for efficiency reasons; i.e., to keep the observation tree small. If efficiency is not a concern, then

Algorithm 4 Function $\textsc{addState}(t, c)$ for an abstract observation table $(S, T, \mathrm{sol}, \rho)$ represented by (S, τ)

Pre-conditions: $t \in T$ and $c \in \mathrm{C}_\Sigma$ with $\dfrac{\psi(ct)}{\psi(\mathrm{sol}(t)t)} \neq \dfrac{\psi(c(t\rho))}{\psi(\mathrm{sol}(t)(t\rho))}$

Post-conditions: return an S'-observation tree (S', τ') such that $S \cup \{t\} \subseteq S'$ and $T \subseteq T(\tau')$, representing an abstract observation table $\mathrm{aot}' = (S', T', \mathrm{sol}', \rho')$

```
   let $\mathrm{nod}_\tau(t) = v = ui$ with $i \in \mathbb{N}$ and $\{T_1, \ldots, T_k\} = \tau(v)/_{\equiv_{\mathrm{C}_\tau(u) \cup \{c\}}}$
2: choose $s_1 \in T_1, \ldots, s_k \in T_k$, such that $\{t, t\rho\} \subseteq \{s_1, \ldots, s_k\}$
   **return** $(S \cup \{s_1, \ldots, s_k\}, \tau[v \leftarrow c[T_1, \ldots, T_k]])$
                    // $\mathrm{aot}' = (S \cup \{s_1, \ldots, s_k\}, T, \mathrm{sol}', \rho')$, where
                    //     $\mathrm{sol}' = \mathrm{sol}\langle s_1 := \mathrm{sol}(t), \ldots, s_k := \mathrm{sol}(t)\rangle$,
                    //     $\rho'(t_i) = s_i$ for $i \in [k]$ and $t_i \in T_i$, and
                    //     $\rho'(u) = u\rho$ for $u \in T \setminus \tau(v)$
```

$(S, \tau[v \leftarrow c'[\tau_1, \ldots, \tau_k, c[\{t\}]]])$ can be returned in either case. In fact, a similar case distinction could be made in line 3, because c is not needed if $t \in \mathrm{supp}(\psi)$.

Lemma 12. *Observation trees implement abstract observation tables.*

Proof. It sufffices to show that $\textsc{addTransition}$ and $\textsc{addState}$ are correct, i.e., that they return S'-observation trees (S', τ') with $S \subseteq S'$ and $T \cup \{t\} \subseteq T(\tau')$ (in case of $\textsc{addTransition}$), resp. $S \cup \{t\} \subseteq S'$ and $T \subseteq T(\tau')$ (in case of $\textsc{addState}$).

Correctness of $\textsc{addTransition}$. The return statement in line 3 is obviously correct, because c is a sign of life for t.

If the condition in line 4 holds, and u is the parent node of v, then $t \equiv_{\mathrm{C}_\tau(u)} t'$ for all $t' \in \tau(v)$. Hence, it follows that the addition of t to $\tau(v)$ does not violate any of the requirements imposed on S-observation trees.

Finally, consider the third case. By the definition of $\mathrm{nod}_\tau(t)$, for all $t' \in \tau(v)$, it holds that $t \not\equiv_{\mathrm{C}_\tau(v)} t'$ but $t \not\equiv_{\mathrm{C}_\tau(u)} t'$ for all proper prefixes u of v. Consequently, $(S \cup \{t\}, \tau[v \leftarrow c'[\tau_1, \ldots, \tau_k, \{t\}]])$ satisfies all conditions imposed on S-observation trees, with the possible exception of condition 3. If condition 3 is violated, then $(S \cup \{t\}, \tau[v \leftarrow c'[\tau_1, \ldots, \tau_k, c[\{t\}]]])$ satisfies it, since c is a sign of life for t. (Clearly, the remaining conditions are not affected by the insertion of c.) Hence, the two return statements in lines 9 and 11 are correct.

Correctness of $\textsc{addState}$. Let (S', τ') be the pair returned by the algorithm. Concerning line 1, notice first that $v \neq \epsilon$, because $T \neq \emptyset$ and, thus, $\tau(\epsilon) = \square$. The pre-condition ensures that $t \not\equiv_{\mathrm{C}_\tau(u) \cup \{c\}} t\rho$, i.e., $t \in T_i$ and $t\rho \in T_j$ for distinct $i, j \in [k]$ in line 1. Hence, $s_1, \ldots, s_k$ can be chosen as required in line 2, which means that (S', τ') with $S' = S \cup \{s_1, \ldots, s_k\}$ and $\tau' = \tau[v \leftarrow c[T_1, \ldots, T_k]]$ satisfies condition 2 of Definition 11. Further, condition 3 is satisfied since τ satisfies it, $T(\tau') = T = T(\tau)$, and $\mathrm{C}_{\mathrm{nod}_\tau(t')} \subseteq \mathrm{C}_{\mathrm{nod}_{\tau'}(t')}$ for all $t' \in T$. Hence, (S', τ') is an S'-observation tree. □

Let us roughly compare the size of an observation table $\boldsymbol{P}$, and the number of coefficient queries required to build it, with the corresponding numbers for an observation tree τ. Clearly, the number of rows of $\boldsymbol{P}$ is equal to the cardinality of $T(\tau)$, because both are equal to $|T|$. For non-trivial cases, the number K of columns of $\boldsymbol{P}$ lies between 2 and $|S|$. These bounds are sharp. On the one hand, two contexts may separate any number of equivalence classes from each other. On the other hand, $|S|$ contexts may be needed to separate the $|S|$ equivalence classes from each other. Thus, $\boldsymbol{P}$ has between $2|T|$ and $|S||T|$ cells, requiring as many coefficient queries.

When using an observation tree τ, the number of coefficient queries required to build it is determined by the depth d at which the trees in T reside in τ. More precisely, let $d(t) = |\mathrm{nod}_\tau(t)|$ for $t \in T$. Then, since a coefficient query has to be asked for each node v such that v is a proper prefix of $\mathrm{nod}_\tau(t)$, the overall number of coefficient queries used to build τ is $D(\tau) = \sum_{t \in T} d(t)$. From the observation that $d(t) \leq |S| + 1$ for all $t \in T$, we obtain the worst-case estimation $D(\tau) \leq (|S| + 1)|T|$, which is essentially the same as above. In the best case, $d(t) = 2$ for all $t \in T$, again yielding the same estimation as above.

So, why should τ have an advantage over $\boldsymbol{P}$? The reason is that, in most cases, one may expect the average of the $d(t)$ to be considerably smaller than the number K of columns of $\boldsymbol{P}$. This is because the contexts indexing the columns of $\boldsymbol{P}$ must simultaneously separate *all* trees in S from each other, whereas the contexts in $\mathrm{C}_\tau(u)$, for a leaf $v = ui$ of τ, only need to separate one tree in S (the one in $\mathrm{nod}_\tau(v)$) from the remaining ones. In other words, we expect $d_{\mathrm{avg}} = \frac{1}{|T|} \sum_{t \in T} d(t)$ to be considerably smaller than K, and thus, $D(\tau) = d_{\mathrm{avg}}|T|$ to be considerably smaller than $K|T|$.

Of course, in concrete cases, there are many factors that can affect d_{avg}. A thorough study of d_{avg} and its relation with K is beyond the scope of this article. Such a study should take into account the properties of the tree series ψ to be learned and suitable probabilistic assumptions regarding the behaviour of the MAT.

6 Experiments

As established in Section 5, observation trees and observation tables are both proper realizations of the abstract observation table (aot) of Section 3. The termination and correctness of the learner are thus guaranteed when instantiated with either data structure. As argued in Section 5, observation trees are expected to have an advantage over observation tables as the former should usually require fewer coefficient queries, but only slightly more equivalence queries, than the latter. To confirm this expectation, we implemented the relevant data structures and algorithms in Java and conducted a series of experiments.[5] In particular, the aot is implemented as an abstract class, the learner as an algorithm instantiated with an aot, and the observation tree and observation table as data structures realizing the aot. The teacher is implemented for various restricted and unrestricted weighted tree automata over semifields. From here on, we refer to the

[5] The source files can be downloaded from `http://www.cs.umu.se/~johanna/adt/`.

learner as $\mathrm{L}_*^{\mathsf{table}}$ when instantiated with an observation table, and as $\mathrm{L}_*^{\mathsf{tree}}$ when instantiated with an observation tree.

In our experiments, we record both the number of coefficient and equivalence queries – the latter because one may suspect that $\mathrm{L}_*^{\mathsf{table}}$, if it by chance receives contexts separating many trees from each other, may use fewer equivalence queries than $\mathrm{L}_*^{\mathsf{tree}}$.

6.1 Results and discussion

We investigate the performance of $\mathrm{L}_*^{\mathsf{table}}$ and $\mathrm{L}_*^{\mathsf{tree}}$ with respect to five families of tree series, each parameterized by a natural number, namely Tower, Size, Numbers, DisMod and PowSet. Their definitions read as follows.

Tower The tree series $\textsc{Tower}_n$, $n \in \mathbb{N}$, is defined over the field $\mathbb{R}$ and the ranked alphabet $\Sigma = \Sigma_0 \cup \Sigma_1$, where $\Sigma_0 = \{\bot\}$ and $\Sigma_1 = \{f_1, \ldots, f_n\}$. For every $t \in \mathrm{T}_\Sigma$,

$$\textsc{Tower}_n(t) = \begin{cases} 1 & \text{if } t = f_n^{i_n} \cdots f_1^{i_1} \bot \text{ with } i_1, \ldots, i_n \geq 1, \text{ and} \\ 0 & \text{otherwise} \ . \end{cases}$$

Size The tree series $\textsc{Size}_n$, $n \in \mathbb{N}$, is defined over the max-plus semiring $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$ and the ranked alphabet $\Sigma = \{a^{(0)}, f^{(2)}\}$. For every $t \in \mathrm{T}_\Sigma$,

$$\textsc{Size}_n(t) = \begin{cases} n & \text{if } n = |\mathrm{nodes}(t)|, \text{ and} \\ -\infty & \text{otherwise} \ . \end{cases}$$

Numbers The tree series $\textsc{Numbers}_n$, $n \in \mathbb{N}$, over $\mathbb{R}$ and $\Sigma = \Sigma_0 \cup \Sigma_2$, where $\Sigma_0 = [n]$ and $\Sigma_2 = \{f\}$, is such that for every $t \in \mathrm{T}_\Sigma$,

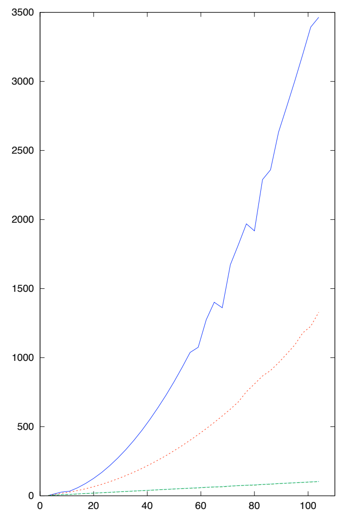
$$\textsc{Numbers}_n(t) = \begin{cases} \displaystyle\prod_{v \in \mathrm{leaves}(t)} t(v) & \text{if } \mathrm{hg}(t) \geq 1, \text{ and} \\ 0 & \text{otherwise} \ . \end{cases}$$

DisMod The tree series $\textsc{DisMod}_n$, $n \in \mathbb{N}$, over $\mathbb{R}$ and $\Sigma = \Sigma_0 \cup \Sigma_2$, where $\Sigma_0 = [\lceil n/4 \rceil]$ and $\Sigma_2 = \{f\}$, is such that for every $t \in \mathrm{T}_\Sigma$,

$$\textsc{DisMod}_n(t) = \begin{cases} 1 & \text{if } t \in \mathrm{T}_{\{f,i\}}, i \in \Sigma_0, \text{ and } \mathrm{hg}(t) \equiv 1 \pmod{n}, \text{ and} \\ 0 & \text{otherwise} \ . \end{cases}$$

PowSet The tree series $\textsc{PowSet}_n$, $n \in \mathbb{N}$, over $\mathbb{R}$ and $\Sigma = \Sigma_0 \cup \Sigma_2$, where $\Sigma_0 = 2^{[n]}$ and $\Sigma_2 = \{f\}$, is such that for every $t \in \mathrm{T}_\Sigma$,

$$\textsc{PowSet}_n(t) = \begin{cases} 1 & \text{if } \displaystyle\bigcup_{v \in \mathrm{leaves}(t)} t(v) = [n], \text{ and} \\ 0 & \text{otherwise} \ . \end{cases}$$



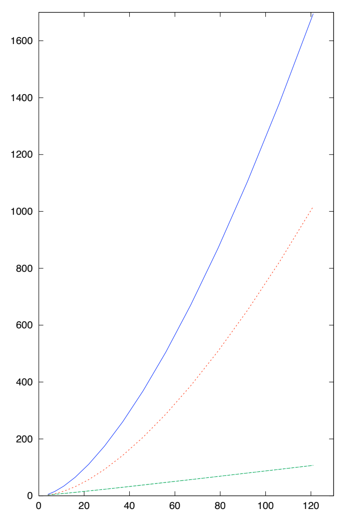
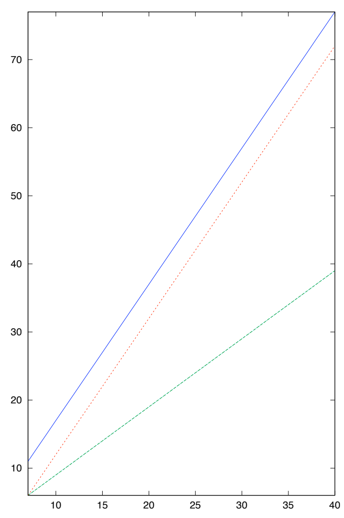


Fig. 1. The outcome of applying L_*^{table} and L_*^{tree} to tree series Tower (left plot) and Size (right plot). The x-axis is labeled with the size of the target dwta; the y-axis with the number of queries posed. The curves are in turn (from above to below): the number of coefficient queries posed by L_*^{table}; the number of coefficient queries posed by L_*^{tree}; and the number of equivalence queries posed by L_*^{table} and L_*^{tree} (which coincide).



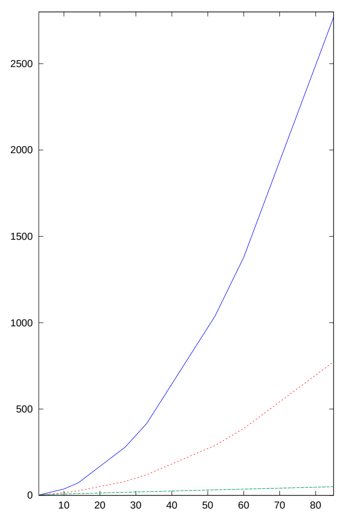


Fig. 2. The outcome of applying L_*^{table} and L_*^{tree} to tree series Numbers (left plot) and DisMod (right plot). Curves and labels are as in Figure 1.

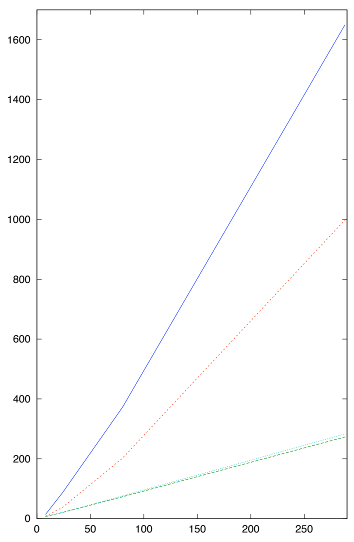


Fig. 3. The outcome of applying $\mathrm{L}_*^{\mathsf{table}}$ and $\mathrm{L}_*^{\mathsf{tree}}$ to the tree series PowSet. Curves and labels are as in Figure 1.

The number of coefficient and equivalence queries posed by $\mathrm{L}_*^{\mathsf{table}}$ and $\mathrm{L}_*^{\mathsf{tree}}$ when inferring the tree series are plotted in Figures 1, 2, and 3.

In all experiments, $\mathrm{L}_*^{\mathsf{tree}}$ requires fewer coefficient queries than $\mathrm{L}_*^{\mathsf{table}}$. The only example in which it makes more equivalence queries than $\mathrm{L}_*^{\mathsf{table}}$ is given by the tree series PowSet. To learn PowSet, $\mathrm{L}_*^{\mathsf{tree}}$ needed 3–4% more equivalence queries. However, the overall computation time (including the time consumed by the teacher!) was still faster than $\mathrm{L}_*^{\mathsf{table}}$. In our implementation, the overhead of using observation trees is larger than for observation tables, so although $\mathrm{L}_*^{\mathsf{tree}}$ is typically faster than $\mathrm{L}_*^{\mathsf{table}}$, this was not the case for the tree series Numbers. Here, the savings in terms of coefficient queries was too small to make up for the additional overhead, so $\mathrm{L}_*^{\mathsf{tree}}$ ran slightly slower than $\mathrm{L}_*^{\mathsf{table}}$.

Let us, finally, discuss how we implemented the teacher. In our implementation, it is initialized with a dwta A computing the target tree series. To answer a coefficient query for a tree t, the teacher simply runs A on t. To answer an equivalence query; i.e., deciding whether A is equivalent to some dwta B, the teacher searches for a tree on which A and B disagree. It is easy to see that if A and B are dwta over the Boolean semiring, or are all-accepting dwta (meaning that every final weight is non-zero) over a semifield, then such a tree can, if it exists, be found in time $\mathcal{O}(n^r m^r)$ where r is the maximum rank of the ranked set, and n and m are the number of states of A and B, respectively. It was shown in [8] that the equivalence problem for probabilistic string automata over fields is in time $\mathcal{O}(|\Sigma|(n+m)^3)$. When the weights are taken from a semifield, the problem can, as we shall see in the subsequent section, be solved in time $\mathcal{O}(r^{nm})$ by an algorithm based on the pumping lemma of [6, Corollary 5.6].

6.2 Deciding the equivalence problem for dwta over semifields

In the following, for a function $f\colon A \to B_1 \times B_2$, we let f_i ($i \in [2]$) denote the composition of f with the projection onto B_i. In particular, for a total dwta $(\Sigma, P, \delta, \lambda)$, $\delta_1\colon \Sigma(P) \to P$ and $\delta_2\colon \Sigma(P) \to \mathbb{S} \setminus \{0\}$ are the (total) functions such that $\delta(u) = (\delta_1(u), \delta_2(u))$ for every $u \in \Sigma(P)$.

Let $A = (\Sigma, P, \delta, \lambda)$ and $B = (\Sigma, Q, \eta, \nu)$ be total dwta. By [7, Theorem 6.1.6] we can suppose, without loss of generality, that A and B have Boolean final weights (i.e., $\lambda\colon P \to \{0,1\}$ and $\nu\colon Q \to \{0,1\}$). We first construct the direct product total dwta $A \cdot B^{-1} = (\Sigma, P \times Q, \delta', \lambda')$ by

$$\begin{aligned}\delta'(f[\langle p_1, q_1\rangle, \dots, \langle p_k, q_k\rangle]) = (\langle \delta_1(f[p_1, \dots, p_k]), \eta_1(f[q_1, \dots, q_k])\rangle,\\ \delta_2(f[p_1, \dots, p_k]) \cdot \eta_2(f[q_1, \dots, q_k])^{-1})\end{aligned}$$

and

$$\lambda'(\langle p, q\rangle) = \begin{cases} 1 & \text{if } \lambda(p) = 1 = \nu(q) \\ 0 & \text{otherwise.} \end{cases}$$

This construction (without the inverses) is taken from [6, Definition 3.7]. Next, we observe that if any state $\langle p, q\rangle$ with $\lambda(p) \neq \nu(q)$ is reachable (i.e., $\hat{\delta}'_1(t) = \langle p, q\rangle$ for some $t \in \mathrm{T}_\Sigma$), then clearly A and B are not equivalent because $A(t) \neq B(t)$ for any tree t that reaches this state. If such a pair does not exist, we can eliminate all states $\langle p, q\rangle$ with $\lambda(p) \neq \nu(q)$ from $A \cdot B^{-1}$ without changing the tree series computed. Moreover, in this case, for every $t \in \mathrm{T}_\Sigma$

$$(A \cdot B^{-1})(t) = \begin{cases} 0 & \text{if } A(t) = 0 = B(t) \\ A(t) \cdot B^{-1}(t) & \text{otherwise} \end{cases}$$

(essentially by [6, Lemma 3.8]). Thus, to decide whether A and B are equivalent, it is sufficient to decide whether this reduced total dwta computes a Boolean tree series (i.e., all coefficients are either 0 or 1). This can (effectively) be decided by [6, Corollary 6.9].

In practice, we often find ourselves working with partial automata, and in these cases it is, of course, sensible to use a decision algorithm that avoids processing dead states in the product automaton (and their associated transitions), because the transition table of a partial dwta may be exponentially smaller than the corresponding total dwta. We therefore conclude with an algorithm that operates on the same principle, but iterates over transitions rather than trees to take advantage of sparsity in the transition table.

In the following, $A = (\Sigma, P, \delta, \lambda)$ and $B = (\Sigma, Q, \eta, \nu)$ are *partial* dwta with Boolean final weights. It is computationally easy to decide if A and B have the same support. Since we operate in a semifield, the support is regular, so it suffices to test the equality of two regular languages. Moreover, the involved automata already yield deterministic unweighted tree automata for the support by setting every nonzero weight to one, so the size of the deterministic unweighted automata for the support is the same as the size of the input automata. Thus, the equality of the supports can be decided by the classical equivalence test for deterministic tree automata, running in time $\mathcal{O}(|A||B|)$. Thus, we may henceforth assume that A and B have the same support, and that A and B have been purged of useless (i.e., unreachable or dead) states.

Lemma 13. *Let A and B be partial dwta that contain no useless states, and are such that* $\mathrm{supp}(A) = \mathrm{supp}(B)$. *The automata A and B are equivalent iff for every $p \in P$ and $q \in Q$ there is a constant $a_{p,q} \in \mathbb{S}$, such that, for all $t \in \delta_1^{-1}(q) \cap \eta_1^{-1}(p)$,*

(i) $a_{p,q} = \frac{\delta_2(t)}{\eta_2(t)}$, *and*
(ii) if p (and thus q) is final, then $a_{p,q} = 1$.

Proof. For the "if" direction, we combine conditions (i) and (ii), and thus obtain that if p and q are final states, then $\delta_2(t) = \eta_2(t)$ for all $t \in \delta_1^{-1}(p) \cap \eta_1^{-1}(q)$.

For the opposite direction, let A and B be equivalent. Consider $p \in P$, $q \in Q$, and $t, u \in \delta_1^{-1}(p) \cap \eta_1^{-1}(q)$. To establish (i), we have to show that $\frac{\delta_2(t)}{\eta_2(t)} = \frac{\delta_2(u)}{\eta_2(u)}$. Since A contains no dead states and $\mathrm{supp}(A) = \mathrm{supp}(B)$, there is a sign of life $c \in C_\Sigma$ for t and u with respect to both A and B. Moreover,

$$\frac{\delta_2(ct)}{\delta_2(t)} = \delta_2(cp) = \frac{\delta_2(cu)}{\delta_2(u)} \quad \text{and} \quad \frac{\eta_2(ct)}{\eta_2(t)} = \eta_2(cq) = \frac{\eta_2(cu)}{\eta_2(u)} ,$$

because $\delta_1(t) = p = \delta_1(u)$ and $\eta_1(t) = q = \eta_1(u)$. We can now compute as follows:

$$\begin{aligned}
\frac{\delta_2(t)}{\eta_2(t)} &= \frac{\delta_2(ct)}{\delta_2(cp)} \cdot \frac{\eta_2(cq)}{\eta_2(ct)} \\
&= \frac{\delta_2(ct) \cdot \delta_2(u)}{\delta_2(cu)} \cdot \frac{\eta_2(cu)}{\eta_2(ct) \cdot \eta_2(u)} \\
&= \frac{\delta_2(ct) \cdot \delta_2(u) \cdot \eta_2(cu)}{\eta_2(ct) \cdot \eta_2(u) \cdot \delta_2(cu)} \\
&= \frac{\delta_2(u)}{\eta_2(u)}
\end{aligned}$$

because $\frac{\delta_2(ct)}{\eta_2(ct)} = 1 = \frac{\eta_2(cu)}{\delta_2(cu)}$ since A and B are equivalent and have Boolean final weights. This last observation also shows that the second condition holds. □

Algorithm 5 traverses the transitions of the product automaton $A \cdot B^{-1}$ to compute a constant $a_{p,q} \in \mathbb{S}$ for each pair of states $\langle p, q \rangle \in P \times Q$ that is reachable, i.e., each pair with $\delta_1^{-1}(p) \cap \eta_1^{-1}(q) \neq \emptyset$. (Note that the constants $a_{p,q}$ for unreachable pairs of states are irrelevant, as they can be chosen arbitrarily.)

For this purpose, the algorithm maintains a partial mapping τ that assigns to each pair of states $\langle p, q\rangle \in P \times Q$ found reachable a constant in $\mathbb{S}$ that is the current candidate for $a_{p,q}$. In the algorithm, the domain of τ is denoted by $\mathrm{dom}(\tau)$. The algorithm starts with the totally undefined mapping $\perp$ and terminates when it discovers a violation of Lemma 13 or has reached a stable state.

Algorithm 5 Decide if A and B are equivalent.

```
Pre-conditions: $A = (\Sigma, P, \delta, \lambda)$ and $B = (\Sigma, Q, \eta, \nu)$ are partial dwta with Boolean
    final weights, contain no useless states, and are such that $\mathrm{supp}(A) = \mathrm{supp}(B)$.
    $\tau \leftarrow \perp$
 2: repeat
      unchanged $\leftarrow$ true
 4:   for all $\langle p, q\rangle \in P \times Q$ such that
          $\exists f \in \Sigma_k$ and $\langle p_1, q_1\rangle, \dots, \langle p_k, q_k\rangle \in \mathrm{dom}(\tau)$ with
          $\delta_1(f[p_1, \dots, p_k]) = p$ and $\eta_1(f[q_1, \dots, q_k]) = q$
       do
        $w_A = \delta_2(f[p_1, \dots, p_k])$
 6:     $w_B = \eta_2(f[q_1, \dots, q_k])$
        $a_{p,q} \leftarrow w_A \cdot w_B^{-1} \cdot \tau(\langle p_1, q_1\rangle) \cdot \ldots \cdot \tau(\langle p_k, q_k\rangle)$
 8:     if $p$ is final and $a_{p,q} \neq 1$ then
          return false
10:     if $\langle p, q\rangle \notin \mathrm{dom}(\tau)$ then
          $\tau(\langle p, q\rangle) \leftarrow a_{p,q}$
12:       unchanged $\leftarrow$ false
        else if $\tau(\langle p, q\rangle) \neq a_{p,q}$. then
14:       return false
    until unchanged
16: return true
```

Lemma 14. *Algorithm 5 decides if A and B are equivalent.*

Proof. Termination is obvious, because every execution of the main loop except the last enlarges $\mathrm{dom}(\tau)$. We show that Algorithm 5 returns *true* if and only if there is, for every $p \in P$ and $q \in Q$, a constant $a_{p,q} \in \mathbb{S}$ that fulfills Condition (i) and (ii) of Lemma 13.

Suppose that Algorithm 5 returns *false*. This can happen in two cases.

- In the first case (lines 13 and 14), the algorithm has reached the same pair of states $\langle p, q\rangle$ on distinct trees s and t, such that $\delta_2(t) \cdot \eta_2(t)^{-1} \neq \delta_2(s) \cdot \eta_2(s)^{-1}$. This violates Condition (i) of Lemma 13.
- In the second case (lines 8 and 9), the algorithm has discovered that a pair of final states $\langle p, q\rangle$ are reachable on a tree t, such that $\delta_2(t) \cdot \eta_2(t)^{-1} \neq 1$. This violates Condition (ii) of Lemma 13.

For the other direction, suppose that the algorithm returns *true* after some iterations of the main loop. For a contradiction, assume that there is a minimal tree $t = f[t_1, \ldots, t_k] \in \delta_1^{-1}(p) \cap \eta_1^{-1}(q)$, such that $\frac{\delta_2(t)}{\eta_2(t)} \neq \tau(\langle p, q\rangle)$ (including the possibility that $\tau(\langle p, q\rangle)$ is undefined). Let $p_i = \delta_1(t_i)$ and $q_i = \eta_1(t_i)$, for all $i \in [k]$. By the minimality assumption, $\frac{\delta_2(t_i)}{\eta_2(t_i)} = \tau(\langle p_i, q_i\rangle)$.

Now, consider the last execution of the body of the main loop. At some point during that execution, the body of the *for all* loop will be executed with the given choice of $\langle p, q\rangle$, f, and $\langle p_1, q_1\rangle, \ldots, \langle p_k, q_k\rangle$. Since neither the condition in line 8 nor the one in line 10 is fulfilled (the latter because $\mathrm{dom}(\tau)$ does not change, according to the termination condition of the main loop), line 13 is reached. However, since line 14 is not reached, this means that

$$\begin{aligned}
&= a_{p,q} \\
&= w_A \cdot w_B^{-1} \cdot \tau(\langle p_1, q_1\rangle) \cdot \ldots \cdot \tau(\langle p_k, q_k\rangle) \\
&= (w_A \cdot \delta_2(t_1) \cdot \ldots \cdot \delta_2(t_k)) \cdot (w_B \cdot \eta_2(t_1) \cdot \ldots \cdot \eta_2(t_k))^{-1} \\
&= \frac{\delta_2(t)}{\eta_2(t)} \,,
\end{aligned}$$

contradicting the assumption that $\frac{\delta_2(t)}{\eta_2(t)} = \tau(\langle p, q\rangle)$. □

Lemma 15. *Algorithm 5 executes in time $\mathcal{O}(|A||B|)$.*

Proof. For an efficient implementation, we begin by calculating a sign of life for every state in the smaller of the two automata. This is done by computing representative trees for each state, at a cost of $\mathcal{O}(\min(|A|, |B|))$ operations, and then searching for the shortest path from each state to an accepting state, consuming another $\mathcal{O}(\min(|P|^2, |Q|^2)) \leq \mathcal{O}(\min(|A|^2, |B|^2))$ operations.

In the worst case, the main loop must traverse every transition in $A \cdot B^{-1}$, but since this is sufficient, the complexity of this loop, and of the entire algorithm, is $\mathcal{O}(|A \cdot B^{-1}|) = \mathcal{O}(|A||B|)$. □

We finally note that Algorithm 5 can easily be extended to return a counterexample whenever A and B are found to be different. For this, it suffices to store, along with each of the values $\tau(\langle p, q\rangle)$, a corresponding tree $t_{p,q}$, such that $\frac{\delta_2(t_{p,q})}{\eta_2(t_{p,q})} = \tau(\langle p, q\rangle)$. Then the two *return* statements in lines 9 and 14 can be adapted in the obvious way to return a counterexample.

References

[1] Dana Angluin. Learning regular sets from queries and counterexamples. *Inform. and Comput.*, 75(2):87–106, 1987.

[2] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987.

[3] Dana Angluin. Queries revisited. In *Proc. 12th Int. Conf. Algorithmic Learning Theory*, volume 2225 of LNCS, pages 12–31. Springer, 2001.

[4] Jean Berstel and Christophe Reutenauer. Recognizable formal power series on trees. *Theoret. Comput. Sci.*, 18(2):115–148, 1982.

[5] Björn Borchardt. The Myhill-Nerode theorem for recognizable tree series. In *Proc. 7th Int. Conf. Developments in Language Theory*, volume 2710 of LNCS, pages 146–158. Springer, 2003.

[6] Björn Borchardt. A pumping lemma and decidability problems for recognizable tree series. *Acta Cybernet.*, 16(4):509–544, 2004.

[7] Björn Borchardt. *The Theory of Recognizable Tree Series.* PhD thesis, Technische Universität Dresden, 2005.

[8] Corinna Cortes, Mehryar Mohri, and Ashish Rastogi. Lp distance and equivalence of probabilistic automata. *J. Comput. System Sci.*, 18(4):761–779, 2007.

[9] Colin de la Higuera and José Oncina. Learning stochastic finite automata. In *Proc. 7th Int. Coll. Grammatical Inference*, volume 3264 of LNCS, pages 175–186. Springer, 2004.

[10] Frank Drewes. MAT learners for recognizable tree languages and tree series. *Acta Cybernet.*, 2009. To appear.

[11] Frank Drewes and Johanna Högberg. Learning a regular tree language from a teacher. In *Proc. 7th Int. Conf. Developments in Language Theory*, volume 2710 of LNCS, pages 279–291. Springer, 2003.

[12] Frank Drewes and Johanna Högberg. Query learning of regular tree languages: How to avoid dead states. *Theory of Comput. Syst.*, 40(2):163–185, 2007.

[13] Frank Drewes and Heiko Vogler. Learning deterministically recognizable tree series. *J. Automata, Languages and Combinatorics*, 12(3):332–354, 2007.

[14] Zoltán Fülöp and Heiko Vogler. Weighted tree automata and tree transducers. In Werner Kuich, Manfred Droste, and Heiko Vogler, editors, *Handbook of Weighted Automata*, chapter 9, pages 313–403. Springer, 2009.

[15] Jonathan S. Golan. *Semirings and their Applications.* Kluwer Academic, Dordrecht, 1999.

[16] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[17] Amaury Habrard and Jose Oncina. Learning multiplicity tree automata. In *Proc. 8th Int. Colloquium Grammatical Inference*, volume 4201 of LNAI, pages 268–280. Springer, 2006.

[18] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, 1994.

[19] Andreas Maletti. Learning deterministically recognizable tree series — revisited. In *Proc. 2nd Int. Conf. Algebraic Informatics*, volume 4728 of LNCS, pages 218–235. Springer, 2007.

[20] José Oncina. Using multiplicity automata to identify transducer relations from membership and equivalence queries. In *Proc. 9th Int. Coll. Grammatical Inference*, volume 5278 of LNCS, pages 154–162. Springer, 2008.

[21] Yasubumi Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoret. Comput. Sci.*, 76(2–3):223–242, 1990.

[22] Ehud Y. Shapiro. *Algorithmic Program Debugging.* ACM Distinguished Dissertation. MIT Press, 1983.

[23] Leslie G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, 1984.