Learning Deterministically Recognizable Tree Series

Frank Drewes
Department of Computing Science
Umeå University
S-90187 Umeå, Sweden
drewes@cs.umu.se

Heiko Vogler
Department of Computer Science
Dresden University of Technology
D-01062 Dresden, Germany
vogler@tcs.inf.tu-dresden.de

Abstract. We devise a learning algorithm for deterministically recognizable tree series where the weights are taken from a commutative group. For this, we use an adaptation of the minimal adequate teacher model that was originally introduced by Angluin. The algorithm runs in polynomial time and constructs the unique minimal deterministic bottom-up finite state weighted tree automaton that recognizes the tree series in question.

UMINF 06.27 ISSN 0348-0542

1 Introduction

In this paper, we propose a learning algorithm for deterministically recognizable tree series. In traditional language learning, the aim is to derive an explicit formal description of an unknown language $U \subseteq T^*$ from examples or similar information. An algorithm accomplishing this task for a given class C of languages is called a learner for C. The theory of language learning was initiated by the seminal paper of Gold [Gol67] on language identification in the limit. Gold considered two sources of information, leading to the notions of learning from text and learning from an informant. In the first case, the learner receives an exhaustive stream of positive examples; in the second, both positive and negative examples are provided. The learner responds with a guess (an automaton, say) each time it receives a new piece of information. The language U is learned in the limit if all but a finite number of the guessed automata are equal and recognize U. Gold showed that, in this model, all recursively enumerable languages can be learned from an informant, but not even all regular languages can be learned from text.

Later, an alternative type of learning called query learning was proposed by Angluin [Ang87, Ang88, Ang04]. In this setting, the learner can actively query an oracle, called the teacher. One type of query learning is MAT learning [Ang87]. It assumes the existence of a *minimally adequate teacher* (MAT) who can answer two types of queries: membership and equivalence queries. A membership query asks whether a certain string is an element of U, and the teacher faithfully answers this question.

An equivalence query asks whether a proposed automaton correctly describes U. The teacher either accepts the automaton or returns a counterexample, i.e., a string which is in the symmetric difference between U and the language accepted by the proposed automaton.

An advantage of MAT learning over identification in the limit is that there exists a natural termination criterion. For the case where U is regular, it is shown in [Ang87] that the minimal deterministic finite automaton recognizing U can be learned in polynomial time in the MAT model. Several other researchers adopted the model; see, e.g., [BR87, Ish90, SY93, Yok94, FR95, Fer02].

The learner proposed in [Ang87] was extended to (skeletal) regular tree languages by Sakakibara [Sak90], which was recently improved by Drewes and Högberg [DH06b]. The advantage of the learner studied in [DH06b] is that it never examines or creates dead states, thus learning the minimal partial deterministic automaton recognizing U (see also [DH06a]). In the present paper, we show that a similar approach can be used to learn recognizable tree series. To our knowledge, this is the first time the questions of learning recognizable tree series is addressed.

The theory of recognizable tree series extends the theory of recognizable formal power series [Eil74, SS78, Wec78, BR88, KS86, Kui97] from strings to trees; a (formal) tree series is a mapping $U \colon T_\Sigma \to S$ where T_Σ is the set of trees over the ranked alphabet Σ and S is the carrier set of a semiring $\underline{S} = (S, \oplus, \odot, 0, 1)$. For the recognition of tree series, a suitable type of automaton is obtained from an ordinary tree automaton [GS84, GS97] by providing every transition with a weight being an element of S. Then, roughly speaking, given an input tree $t \in T_\Sigma$, an automaton A may have several runs on t; the weight of a run is the product (taking $\odot$) of the weights of the transitions which are applied in this run. Finally, the weight of accepting t by S is the sum (taking $\oplus$) of the weights of all the runs on t. The resulting mapping $\underline{A} \colon T_\Sigma \to S$ is the tree series recognized by A.

Recognizable tree series have been introduced in [BR82] (over fields as weight domain) and generalized in [Boz99, Kui98] (by replacing fields by semirings). Since then the theory of recognizable tree series has been developed along the lines of the boolean, i.e., unweighted case. For instance, it has been proved that recognizable tree series over semirings (sometimes with additional restrictions) are equivalent to: least solutions of finite polynomial systems [Kui98, ÉK03], rational tree series [Boz99, Kui98, Pec03b, Pec03a, DPV05], and tree series definable by weighted (restricted) monadic second-order logic [DV05]. Some more results about recognizable tree series can be found in: [BR82, Bor04a] on pumping lemmata and decidability results; [Sei90, Boz91] on deciding the equivalence of two effectively given recognizable tree series over a field; [Sei94, BFGM05] on deciding the boundedness of values which may occur for a weighted tree automaton over partially ordered semirings; [Boz99] (also cf. Lemma 4.1.13 von [Bor04a]) on the equivalence of the run semantics and the inital algebra semantics. (The present paper will be based on the latter.) For a survey article on recognizable tree series we refer the reader to [ÉK03].

In order to be able to learn a recognizable tree series U, we use an appropriate adaptation of the MAT model in which membership queries are replaced by coefficient queries: given a tree $t \in T_\Sigma$, the teacher returns the coefficient of t, i.e., the value (U, t).[1] Naturally, an equivalence query now asks whether a proposed automaton A recognizes U, and a counterexample is a tree $t \in T_\Sigma$ such that $(\underline{A}, t) \neq (U, t)$.

The learners given in [Ang87, Sak90, DH06b, DH06a] are all based on the Myhill-Nerode theorems for regular string languages (cf. e.g. [Yu97]) and for recognizable tree languages [FV89, Koz92]. The basic idea of the learning algorithms is to extract, from the counterexamples, trees that represent the equivalence classes of T_Σ induced by U, and to use these representatives as the states of the automata proposed by the learner. Here, we exploit the Myhill-Nerode theorem for recognizable tree series in a similar manner; this theorem has been proved in [Bor03, Bor04b] for recognizable tree series over commutative semifields. Interestingly, the fact that the learner in [DH06b, DH06a] avoids dead states turns out to be an essential property for this approach to work. Intuitively, a dead state corresponds to subtrees having the weight zero; without avoiding them, the learner would attempt to divide by zero when constructing a proposed automaton. As a result, we obtain a learner that learns a recognizable tree series in polynomial time. To be precise, we consider only the so-called deterministically aa-recognizable tree series, i.e., tree series which are accepted by deterministic automata in which every state is final (*a*ll *a*ccepting). Since the considered automata are deterministic, we do not need the semiring addition; hence, instead of commutative semifields (needed for the Myhill-Nerode theorem in the nondeterministic case) we use commutative groups as domains of the weights. Finally, we note that it is an open problem whether the restriction 'aa' can be dropped.

The paper is structured as follows. The next section recalls some basic notions and notation. Section 3 contains the definition of deterministic bottom-up finite state weighted tree automata and discusses their basic properties to the extent necessary for the remainder of the paper. Section 4 presents the learner and discusses an example. Finally, Section 5 proves the correctness of the algorithm and discusses its complexity.

2 Preliminaries

Throughout the paper, $\mathbb{Z}$ denotes the set of all integers; $\mathbb{N}$ and $\mathbb{N}_+$ denote the sets $\{0, 1, 2, \ldots\}$ and $\{1, 2, \ldots\}$, respectively.

If $\sim \subseteq T \times T$ is an equivalence relation over a set T (i.e., $\sim$ is reflexive, symmetric, and transitive), and t is an element of T, then the equivalence class of t is $[t]_\sim = \{s \in T \mid s \sim t\}$.

[1]As usual when dealing with formal power series, we denote the value of U at t by (U, t) rather than by $U(t)$.

2.1 Partial functions

Let A, B be two sets and $f \subseteq A \times B$. If for every $x \in A$ there is at most one $y \in B$ such that $(x, y) \in f$, then f is a *partial function* and it is denoted by $f \colon A \multimapdotbothA B$; rather than $(x, y) \in f$ we write as usual $f(x) = y$. We let

$$\begin{aligned} \mathit{df}(f) &= \{x \in A \mid \text{ there is a } y \in B \text{ such that } f(x) = y\} \text{ and} \\ \mathit{rg}(f) &= \{y \in B \mid \text{ there is an } x \in A \text{ such that } f(x) = y\}. \end{aligned}$$

We say that $f(x)$ is *defined* if $x \in \mathit{df}(f)$; otherwise, $f(x)$ is *undefined.* Note that writing $f(x) = y$ implies in particular that $f(x)$ is defined. By convention, every expression containing an undefined subexpression yields itself an undefined value. If e, e' are two expressions involving partial functions and we want to express that they are either both defined and equal or are both undefined, then we use the notation $e \doteq e'$.

2.2 Trees

A *ranked alphabet* is a finite set of symbols $\Sigma = \bigcup_{k \in \mathbb{N}} \Sigma^{(k)}$ which is partitioned into pairwise disjoint subsets $\Sigma^{(k)}$. The symbols in $\Sigma^{(k)}$ are said to have rank k; this we will also be indicated by writing $f^{(k)}$ for some $f \in \Sigma^{(k)}$. If $\Sigma = \bigcup_{k \in \mathbb{N}} \Sigma^{(k)}$ and $\Delta = \bigcup_{k \in \mathbb{N}} \Delta^{(k)}$ are two ranked alphabets, then $\Sigma \cup \Delta$ is the ranked alphabet $\bigcup_{k \in \mathbb{N}} \big(\Sigma^{(k)} \cup \Delta^{(k)}\big)$. We write $\Sigma \cap \Delta = \emptyset$, if for every $k \in \mathbb{N}$ we have $\Sigma^{(k)} \cap \Delta^{(k)} = \emptyset$.

The set T_Σ of *trees over* Σ is the smallest set $T \subseteq \Sigma^*$ such that $f\, t_1 \cdots t_k \in T$ whenever $f \in \Sigma^{(k)}$ and $t_1, \ldots, t_k \in T$. For the sake of better readability we write $f[t_1, \ldots, t_k]$ instead of $f\, t_1 \cdots t_k$, unless $k = 0$. For a set T of trees (over an arbitrary ranked alphabet), $\Sigma(T)$ denotes the set $\{f[t_1, \ldots, t_k] \mid k \geq 0, f \in \Sigma^{(k)}, t_1, \ldots, t_k \in T\}$.

We will frequently decompose a tree into a so-called context and a subtree. For this purpose, we reserve the symbol $\square$ of rank zero that does not occur in Σ. A tree $c \in T_{\Sigma \cup \{\square\}}$ in which $\square$ occurs exactly once (as a leaf) is called a *context (over* Σ*)*. The set of all contexts over Σ is denoted by C_Σ. Given a context c and a tree s, we denote by $c[\![s]\!]$ the tree which is obtained from c by replacing $\square$ by s.

In the rest of this paper Σ denotes an arbitrary ranked alphabet.

2.3 Monoids

A *(multiplicative) monoid* is an algebraic structure $(G, \odot, 1)$ with an operation $\odot$ and a constant 1 such that $\odot$ is associative and 1 is the unit element of $\odot$. A monoid has an *absorbing element* $0 \in G$ if for every $x \in G$ the equalities $x \odot 0 = 0 \odot x = 0$ hold. Such a monoid is denoted by $(G, \odot, 0, 1)$. A monoid with absorbing 0 is a *group* if for every $x \in G \setminus \{0\}$ there is a unique $y \in G$ such that $x \odot y = y \odot x = 1$; this element is called the *inverse* of x and it is denoted by x^{-1}. A monoid (resp. group) is *commutative* if for every $x, y \in G$ the equality $x \odot y = y \odot x$ holds.

We note that every group G is *zero-divisor free*, i.e., for every $x, y \in G$: if $x \odot y = 0$, then $x = 0$ or $y = 0$.

Throughout the rest of this paper G denotes an arbitrary commutative group $(G, \odot, 0, 1)$. Our algorithms will frequently have to operate on elements of G. Therefore, we assume that the elements of G can be represented effectively and that, with the input $x, y \in G$, both $x \odot y$ and x^{-1} are computable.

3 Formal tree series and weighted tree automata

We now define the types of formal tree series and weighted tree automata that will be considered in the remainder of this paper. Since we restrict ourselves to the deterministic case, it suffices to consider tree series with coefficients taken from the commutative group G rather than from a semiring.

3.1 Formal tree series

Let Σ be a ranked alphabet. A *(formal) tree series (over T_Σ and the commutative group G)* is a mapping $U\colon T_\Sigma \to G$. For every $t \in T_\Sigma$, the element $U(t) \in G$ is called the *coefficient* of t and it is also denoted by (U, t). The *support of U* is defined as the set $supp(U) = \{t \in T_\Sigma \mid (U, t) \neq 0\}$. The tree series U is *subtree closed* if for every $t \in T_\Sigma$ the fact that $t \in supp(U)$ implies that $s \in supp(U)$ for every subtree s of t. The set of all tree series over T_Σ and G is denoted by $G\langle\langle T_\Sigma\rangle\rangle$.

3.2 Finite state weighted tree automata

A *deterministic zero-free bottom-up finite state weighted tree automaton (for short: wta) over Σ and G* is a tuple $A = (Q, \delta, \lambda, F)$ such that

- Q is a ranked alphabet (of *states*) with $Q = Q^{(0)}$ and $Q \cap \Sigma = \emptyset$,
- $\delta = (\delta_k \mid k \geq 0)$ is a family of partial mappings $\delta_k\colon \Sigma^{(k)} \times Q^k \multimap Q$ (*transition mappings*),
- $\lambda = (\lambda_k \mid k \geq 0)$ is a family of partial mappings $\lambda_k\colon \Sigma^{(k)} \times Q^k \multimap G \setminus \{0\}$ (*coefficient mappings*) that satisfy $df(\delta_k) = df(\lambda_k)$, and
- $F \subseteq Q$ (*final states*).

An *all accepting wta* (for short: aa-wta) is a wta such that every state is also a final state. In this case we drop the F from the specification of the automaton and just write $A = (Q, \delta, \lambda)$.

Next we define the partial mappings $\widetilde{\delta}\colon T_{\Sigma\cup Q} \multimap Q$ and $\widetilde{\lambda}\colon T_{\Sigma\cup Q} \multimap G$ simultaneously by induction, as follows:

- for every $q \in Q$, $\widetilde{\delta}(q) = q$ and $\widetilde{\lambda}(q) = 1$, and
- for every tree $t = f[t_1, \ldots, t_k] \in \Sigma(T_{\Sigma\cup Q})$,

$$\widetilde{\delta}(t) \doteq \delta_k(f, \widetilde{\delta}(t_1), \ldots, \widetilde{\delta}(t_k))$$

and
$$\widetilde{\lambda}(t) \doteq \lambda_k(f, \widetilde{\delta}(t_1), \ldots, \widetilde{\delta}(t_k)) \odot \bigodot_{i=1}^{k} \widetilde{\lambda}(t_i).$$

Note that, for $t = f[t_1, \ldots, t_k] \in \Sigma(T_{\Sigma \cup Q})$, the values $\widetilde{\delta}(t)$ and $\widetilde{\lambda}(t)$ are undefined if at least one of the $\widetilde{\delta}(t_i)$ is undefined or if $\widetilde{\delta}(t_1) = q_1, \ldots, \widetilde{\delta}(t_k) = q_k$ but $(f, q_1, \ldots, q_k) \notin df(\delta_k)$. Clearly, $df(\widetilde{\delta}) = df(\widetilde{\lambda})$ and $0 \notin rg(\widetilde{\lambda})$. In order to reduce notational complexity, we will drop the tilde from $\widetilde{\delta}$ and $\widetilde{\lambda}$ in the sequel and just write δ and λ, respectively.

The *tree series recognized by A* is the tree series $\underline{A}\colon T_\Sigma \to G$ such that for every $t \in T_\Sigma$:

$$(\underline{A}, t) = \begin{cases} \lambda(t) & \text{if } t \in df(\delta) \text{ and } \delta(t) \in F, \\ 0 & \text{otherwise.} \end{cases}$$

A tree series $U \in G\langle\langle T_\Sigma \rangle\rangle$ is *deterministically recognizable* (resp. *deterministically aa-recognizable*) if there is a wta A (resp. aa-wta A) such that $U = \underline{A}$.

Observation 3.1 Let A be an aa-wta. Then $\underline{A}$ is subtree closed.

Example 3.2 Let $\Sigma = \{f^{(1)}, a^{(0)}\}$. Moreover, consider the group $(\mathbb{Q}, \cdot, 0, 1)$. Now consider the formal tree series $U \in G\langle\langle T_\Sigma \rangle\rangle$ such that $(U, a) = 0$ and $(U, f^n[a]) = 1$ for every $n \geq 1$ (where $f^n[a]$ abbreviates $f[\cdots f[a] \cdots]$ with n occurrences of f).

This formal tree series can be accepted by the wta $A = (Q, \delta, \lambda, F)$ with $Q = \{q, p\}$, $F = \{q\}$, and $\delta_0(a) = p$, $\lambda_0(a) = 1$, $\delta_1(f, p) = \delta_1(f, q) = q$, and $\lambda_1(f, p) = \lambda_1(f, q) = 1$. Thus, for every $n \geq 0$, we have that $\lambda(f^n[a]) = 1$. However, $(\underline{A}, a) = 0$, because $p \notin F$. Moreover, $(\underline{A}, f^n[a]) = 1$ for every $n \geq 1$. Thus, $\underline{A} = U$.

Due to Observation 3.1, U cannot be recognized by an aa-wta.

Example 3.3 Let $\Sigma = \{f^{(2)}, a^{(0)}, b^{(0)}\}$ and consider the group $(\mathbb{Z}_\infty, +, \infty, 0)$, where $\mathbb{Z}_\infty = \mathbb{Z} \cup \{\infty\}$ and $+$ is extended to $\mathbb{Z}_\infty$ as usual. Clearly, ∞ is absorbing with respect to $+$, and 0 is the unit element of $+$.

Now we consider the following formal tree series $U \in G\langle\langle T_\Sigma \rangle\rangle$. A subtree s of a tree $t \in T_\Sigma$ is called an a-subtree if every leaf of s is labeled by a. Let $\max_a(t)$ be the set of all maximal a-subtrees of t. Then, for every $t \in T_\Sigma$, we define
$$(U, t) = \sum_{s \in \max_a(t)} size(s)$$
where $size(s)$ is the number of positions (or: nodes) of s. Thus, for instance, $(U, f[f[a, a], f[a, b]]) = 4$.

This formal tree series U is deterministically aa-recognizable. To show this, let $A = (Q, \delta, \lambda)$ be the wta such that

- $Q = \{c, p\}$ where c and p stand for 'count' and 'propagate', respectively,

- $\delta_0(a) = c$, $\lambda_0(a) = 1$, and $\delta_0(b) = p$, $\lambda_0(b) = 0$, and

$$\delta_2(f, q_1, q_2) = \begin{cases} c & \text{if } q_1 = q_2 = c \\ p & \text{otherwise} \end{cases}$$

$$\lambda_2(f, q_1, q_2) = \begin{cases} 1 & \text{if } q_1 = q_2 = c \\ 0 & \text{otherwise.} \end{cases}$$

It should be rather obvious that $U = \underline{A}$. □

Lemma 3.4 For every wta $A = (Q, \delta, \lambda, F)$, every context $c \in C_\Sigma$, and every tree $t \in T_\Sigma$,

$$\delta(c[\![t]\!]) \doteq \delta(c[\![\delta(t)]\!]) \quad \text{and} \quad \lambda(c[\![t]\!]) \doteq \lambda(c[\![\delta(t)]\!]) \odot \lambda(t).$$

Proof. The equality $\delta(c[\![t]\!]) \doteq \delta(c[\![\delta(t)]\!])$ is well known from the theory of ordinary deterministic bottom-up finite tree automata, and it follows by a straightforward structural induction on c. The proof of the equality $\lambda(c[\![t]\!]) \doteq \lambda(c[\![\delta(t)]\!]) \odot \lambda(t)$ is similar, using the fact that the group G is assumed to be commutative, and it is therefore left out. □

We note that our notion of wta is closely related to the deterministic bottom-up weighted tree automata with final states (fs-wta, for short) defined in [Bor04b, Definition 4.1.9]; see also [BV03, Bor03]. In particular, these models are equally powerful. (Readers who are unfamiliar with fs-wta should skip the rest of this paragraph.) Let $M = (Q, \Sigma, F, G, \mu)$ be an fs-wta with final state set F (where $\mu = (\mu_k \mid k \geq 0)$ and $\mu_k \colon \Sigma^{(k)} \to G^{Q^k \times Q}$) and $A = (Q, \delta, \lambda, F)$ be a wta over Σ and G. Then we say that M and A are *related* if for every $k \geq 0$, $f \in \Sigma^{(k)}$, $q, q_1, \ldots, q_k \in Q$, the following properties hold:

- $\delta_k(f, q_1, \ldots, q_k)$ is undefined if and only if $\mu_k(f)_{q_1,\ldots,q_k,q} = 0$ for all $q \in Q$,
- if $\mu_k(f)_{(q_1,\ldots,q_k),q} \neq 0$, then $\delta_k(f, q_1, \ldots, q_k) = q$ and $\lambda_k(f, q_1, \ldots, q_k) = \mu_k(f)_{(q_1,\ldots,q_k),q}$, and
- if $(f, q_1, \ldots, q_k) \in df(\delta_k)$, then $\mu_k(f)_{(q_1,\ldots,q_k),q} = \lambda_k(f, q_1, \ldots, q_k)$ where $q = \delta_k(f, q_1, \ldots, q_k)$.

It can easily be proved that M and A compute the same tree series if they are related. Moreover, for every wta A there is an fs-wta M with final states such that M and A are related, and vice versa, for every fs-wta M with final states there is a wta A such that M and A are related. Thus, as claimed above, wta and fs-wta are equally powerful models.

In this paper we will only deal with aa-wta, i.e., with deterministic all accepting bottom-up finite state weighted tree automata. Note that this is a relatively weak restriction since every (deterministic) wta A can be split into an aa-wta A_{aa} and an ordinary bottom-up finite tree automaton B such that, for all $t \in T_\Sigma$,

$$(\underline{A}, t) = \begin{cases} (\underline{A_{\mathrm{aa}}}, t) & \text{if } t \in L(B), \\ 0 & \text{otherwise,} \end{cases}$$

where $L(B)$ denotes the tree language recognized by B.

3.3 Minimal deterministic wta

In this section we will construct, for a given deterministically aa-recognizable tree series $U \in G\langle\langle T_\Sigma \rangle\rangle$, an aa-wta A_U such that $U = \underline{A_U}$ and A_U is minimal with respect to the number of states of all the aa-wta which recognize U. More or less, we follow the lines of Section 5 of [Bor03] (and Section 7.1 of [Bor04b]), where deterministic bottom-up weighted tree automata with final weights over commutative semifields were considered. (Such automata do not only distinguish states to be final, but additionally produce a weight or cost in such final states.) However, due to determinism, the addition of the semifield is not needed; thus we choose a commutative (multiplicative) group with an absorbing 0 for our development. Moreover, our approach differs from the one of [Bor03] in the sense that we deal with *all accepting* tree automata, whereas in [Bor03] final states can be distinguished. If final states can be distinguished, then one can get rid of final weights (cf. Lemma 6.1.4 of [Bor04b]); however, this construction does not preserve the property of being all accepting. Thus, in order to have a clear and self-contained presentation, we show all the adapted lemmata and their proofs (but always refer to related lemmata of [Bor03] and [Bor04b]).

Let $U \in G\langle\langle T_\Sigma \rangle\rangle$ be a deterministically aa-recognizable tree series. Recall from Observation 3.1 that U is subtree closed, i.e., if $t \in \mathit{supp}(U)$, then this membership also holds for every subtree of t. We define the *equivalence relation of U* as the binary relation $\sim_U \subseteq T_\Sigma \times T_\Sigma$ such that for every $t, t' \in T_\Sigma$, $t \sim_U t'$ if and only if there is an $x \in G \setminus \{0\}$ such that for every $c \in C_\Sigma$ the equation $(U, c[\![t]\!]) = (U, c[\![t']\!]) \odot x$ holds (cf. Section 5 of [Bor03] and Section 7.1 of [Bor04b]). In this section, we will abbreviate a class $[t]_{\sim_U}$ by $[t]$.

Lemma 3.5 (cf. Lemma 7.1.2(ii) of [Bor04b]) The relation $\sim_U$ is a congruence relation with respect to the intial Σ-term algebra.
Proof. It is clear that $\sim_U$ is an equivalence relation. Now let $k \geq 1$, $f \in \Sigma^{(k)}$, and for every $1 \leq i \leq k$, let $t_i, t'_i \in T_\Sigma$ such that $t_i \sim_U t'_i$. Then, for every $1 \leq i \leq k$, there is a $y_i \in G \setminus \{0\}$ such that for every context $c \in C_\Sigma$, we have $(U, c[\![t_i]\!]) = (U, c[\![t'_i]\!]) \odot y_i$. Since G is zero-divisor free, it follows that $\bigodot_{i=1}^k y_i \neq 0$. Now let $d \in C_\Sigma$ be an arbitrary context over Σ. Then we can calculate as follows:

$$\begin{aligned}
(U, d[\![f[t_1, \ldots, t_k]]\!]) &= (U, d[\![f[\Box, t_2, \ldots, t_k]]\!][\![t_1]\!]) \\
&= (U, d[\![f[\Box, t_2, \ldots, t_k]]\!][\![t'_1]\!]) \odot y_1 \\
&= (U, d[\![f[t'_1, t_2, \ldots, t_k]]\!]) \odot y_1 \\
&\vdots \\
&= (U, d[\![f[t'_1, \ldots, t'_k]]\!]) \odot \textstyle\bigodot_{i=1}^k y_i.
\end{aligned}$$

Hence $f[t_1, \ldots, t_k] \sim_U f[t'_1, \ldots, t'_k]$. $\Box$

Lemma 3.6 (cf. Lemma 7.1.3 of [Bor04b]) Let $A = (Q, \delta, \lambda)$ be an arbitrary aa-wta over Σ and G such that $\underline{A} = U$. Then, for all trees $t, t' \in T_\Sigma$, if $\delta(t) = \delta(t')$, then $t \sim_U t'$.

Proof. Let $t, t' \in T_\Sigma$ such that $\delta(t) = \delta(t')$. Thus, in particular, $t, t' \in \mathit{df}(\lambda)$ and $t, t' \in \mathit{supp}(U)$. Then, since G is a group, there is an $x \in G \setminus \{0\}$ such that $\lambda(t) = \lambda(t') \odot x$. Now let $c \in C_\Sigma$ be an arbitrary context over Σ. We distinguish two cases.

$\underline{c[\![t]\!] \in \mathit{df}(\delta)}$: Then, due to Lemma 3.4 and the assumption that $\delta(t) = \delta(t')$, also $c[\![t']\!] \in \mathit{df}(\delta)$. Since A is all accepting we can compute as follows (by using Lemma 3.4 twice):

$$\begin{aligned}
(\underline{A}, c[\![t]\!]) &= \lambda(c[\![t]\!]) \\
&= \lambda(c[\![\delta(t)]\!]) \odot \lambda(t) \\
&= \lambda(c[\![\delta(t')]\!]) \odot \lambda(t') \odot x \\
&= \lambda(c[\![t']\!]) \odot x \\
&= (\underline{A}, c[\![t']\!]) \odot x.
\end{aligned}$$

$\underline{c[\![t]\!] \notin \mathit{df}(\delta)}$: Then again it follows easily that also $c[\![t']\!] \notin \mathit{df}(\delta)$. Hence $(\underline{A}, c[\![t]\!]) = 0 = (\underline{A}, c[\![t']\!]) = (\underline{A}, c[\![t']\!]) \odot x$. $\square$

Now we construct the aa-wta $A_U = (Q_U, \delta_U, \lambda_U)$ as follows (cf. Definition 7.2.1 of [Bor04b]):

- $Q_U = \{[t] \mid t \in \mathit{supp}(U)\}$ and
- the families $\delta_U = (\delta^U_k \mid k \geq 0)$ and $\lambda_U = (\lambda^U_k \mid k \geq 0)$ are defined for every $k \geq 0$, $f \in \Sigma^{(k)}$, and $t_1, \ldots, t_k \in T_\Sigma$ by

$$\delta^U_k(f, [t_1], \ldots, [t_k]) = \begin{cases} [f[t_1, \ldots, t_k]] & \text{if } (U, f[t_1, \ldots, t_k]) \neq 0, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

and

$$\lambda^U_k(f, [t_1], \ldots, [t_k]) = \begin{cases} (U, f[t_1, \ldots, t_k]) \odot \bigodot_{i=1}^{k} (U, t_i)^{-1} & \\ \qquad\qquad \text{if } (U, f[t_1, \ldots, t_k]) \neq 0, \\ \text{undefined} \quad \text{otherwise.} \end{cases}$$

Note that Q_U is finite: since U is deterministically aa-recognizable and by Lemma 3.6 every aa-wta A with $\underline{A} = U$ has at least $|Q_U|$ states. Consequently, the finiteness of the state set of A implies that Q_U is finite as well. Moreover, the definitions of δ_k and λ_k do not depend on the choice of $t_1, \ldots, t_k$, because $\sim_U$ is a congruence, cf. Lemma 3.5. Finally, note that $\lambda^U_k(f, [t_1], \ldots, [t_k]) \neq 0$ unless $\lambda^U_k(f, [t_1], \ldots, [t_k])$ is undefined. This is because $[t_i] \in Q_U$ implies that $(U, t_i) \neq 0$ and thus $(U, t_i)^{-1} \neq 0$; since every multiplicative group is zero-divisor free and G is a group, this yields $\bigodot_{i=1}^{k} (U, t_i)^{-1} \neq 0$. Thus, if $(U, f[t_1, \ldots, t_k]) \neq 0$, then $\lambda^U_k(f, [t_1], \ldots, [t_k]) = (U, f[t_1, \ldots, t_k]) \odot \bigodot_{i=1}^{k} (U, t_i)^{-1} \neq 0$, and if $(U, f[t_1, \ldots, t_k]) = 0$, then $\lambda^U_k(f, [t_1], \ldots, [t_k])$ is undefined.

Lemma 3.7 (cf. Theorem 4 of [Bor03] and Theorem 7.4.1(a) of [Bor04b]) *A_U is a state-minimal aa-wta which recognizes U.*

Proof. Let us show that $U = \underline{A_U}$. For this we prove the following two statements, for every $t \in T_\Sigma$:

(1) If $t \in supp(U)$, then $\delta(t) = [t]$ and $\lambda_U(t) = (U, t)$.

(2) If $t \notin supp(U)$, then $\delta(t)$ is undefined.

Clearly, from these two statements the equality $U = \underline{A_U}$ follows directly. Let us prove statement (1) by induction on t. Let $t = f[t_1, \ldots, t_k] \in supp(U)$. By the induction hypothesis we can assume that, for every $1 \le i \le k$, the implication of statement (1) holds for t_i. Since U is deterministically aa-recognizable and $t \in supp(U)$, it follows from Observation 3.1 that also $t_i \in supp(U)$, and thus we know that $\delta(t_i) = [t_i]$ and $\lambda_U(t_i) = (U, t_i)$. Then, using the definitions of δ_U and λ_U, we obtain that $\delta_U(t) = [t]$ and

$$\begin{aligned}
\lambda_U(t) &= \lambda_k^U(f, \delta_U(t_1), \ldots, \delta_U(t_k)) \odot \textstyle\bigodot_{i=1}^k \lambda_U(t_i) \\
&= \lambda_k^U(f, [t_1], \ldots, [t_k]) \odot \textstyle\bigodot_{i=1}^k (U, t_i) \\
&= (U, f[t_1, \ldots, t_k]) \odot \textstyle\bigodot_{i=1}^k (U, t_i)^{-1} \odot \textstyle\bigodot_{i=1}^k (U, t_i) \\
&= (U, f[t_1, \ldots, t_k]).
\end{aligned}$$

Now we prove statement (2) by induction on t. Let $t \notin supp(U)$, i.e., $(U, t) = 0$. Then, in particular, $\delta_k^U(f, [t_1], \ldots, [t_k])$ is undefined (note that the equivalence classes $[t_i]$ are defined, although they might not be states in Q_U). Now we make a case analysis:

$\underline{\forall 1 \le i \le k \colon t_i \in supp(U)}$: Then, using statement (1), it holds that $\delta(t_i) = [t_i]$. Consequently, $\delta(t) = \delta_k^U(f, \delta(t_1), \ldots, \delta(t_k)) = \delta_k^U(f, [t_1], \ldots, [t_k])$ is undefined.

$\underline{\exists 1 \le i \le k \colon t_i \notin supp(U)}$: Then, by the induction hypothesis, $\delta(t_i)$ is undefined, and hence, by definition of δ, also $\delta(t)$ is undefined.

This proves statement (2), and in total we obtain that $U = \underline{A_U}$. Minimality of A_U follows from Lemma 3.6. $\square$

Lemma 3.8 Up to bijective renaming of states, every minimal aa-wta A recognizing U is equal to A_U.

Proof. Let $A = (Q, \delta, \lambda)$ be a minimal aa-wta such that $\underline{A} = U$. By Lemma 3.6 and the minimality of A it holds for all $t, t' \in supp(U)$ that $t \sim_U t'$ if and only if $\delta(t) = \delta(t')$. Furthermore, minimality implies that, for every $q \in Q$, there exists a tree $t \in supp(U)$ such that $\delta(t) = q$. Hence the mapping $\pi \colon Q \to \{[t] \mid t \in supp(U)\}$ defined by $\pi(q) = [t]$, where $t \in T_\Sigma$ is any tree such that $\delta(t) = q$, is a bijection. For simplicity we will assume that π is the identity.

Next, we show that zero-freeness implies $\delta = \delta_U$. Let $f \in \Sigma^{(k)}$ and $t_1, \ldots, t_k \in supp(U)$, and consider $t = f[t_1, \ldots, t_k]$. Clearly, $t \in supp(U)$ implies $\delta_k(f, [t_1], \ldots, [t_k]) = \delta(t) = [t] = \delta_k^U(f, [t_1], \ldots, [t_k])$. If $t \notin supp(U)$ (i.e., $(\underline{A}, t) = 0$), then $\lambda_k(f, [t_1], \ldots, [t_k])$ must be undefined, because A is zero-free and $\lambda(t_i) = (U, t_i) \neq 0$ for $1 \le i \le k$ (because of the assumption that $t_1, \ldots, t_k \in supp(U)$). Hence, $\delta_k(f, [t_1], \ldots, [t_k])$ is undefined as well, and thus equal to $\delta_k^U(f, [t_1], \ldots, [t_k])$.

Now, let us show that $\lambda_k(f, [t_1], \ldots, [t_k]) \doteq \lambda_k^U(f, [t_1], \ldots, [t_k])$ (where, again, $f \in \Sigma^{(k)}$, $t_1, \ldots, t_k \in supp(U)$, and $t = f[t_1, \ldots, t_k]$). If one side is defined while the other is not, by zero-freeness exactly one of $(\underline{A}, t)$ and $(\underline{A_U}, t)$ would be equal to zero, contradicting the assumption that $\underline{A} = U = \underline{A_U}$. Thus, let us assume that both sides are defined. Then, since $0 \notin rg(\lambda) = rg(\lambda_U)$, we have

$$\begin{aligned}\lambda_k(f, [t_1], \ldots, [t_k]) &= \lambda(t) \odot \textstyle\bigodot_{i=1}^{k} \lambda(t_i)^{-1} \\ &= (U, t) \odot \textstyle\bigodot_{i=1}^{k} (U, t_i)^{-1} \\ &= \lambda_U(t) \odot \textstyle\bigodot_{i=1}^{k} \lambda_U(t_i)^{-1} \\ &= \lambda_k^U(f, [t_1], \ldots, [t_k]),\end{aligned}$$

as claimed. $\square$

4 Learning

We now turn to the main topic of this paper, namely learning a tree series. First, we define the learning model to be used.

4.1 The learning model

In the rest of this paper let $U \in G\langle\langle T_\Sigma \rangle\rangle$ be a deterministically aa-recognizable tree series. The aim is to learn U algorithmically, where the only source information available is a “teacher” who knows U and can answer the following two types of queries:

(a) **Coefficient queries**: Given a tree $t \in T_\Sigma$, the teacher returns the coefficient $(U, t) \in G$.

(b) **Equivalence queries**: Given an aa-wta A. The teacher checks whether $\underline{A} = U$. If not, then she returns a counterexample, i.e., a tree $t \in T_\Sigma$ such that $(\underline{A}, t) \neq (U, t)$.

This model is a straightforward adaptation of the MAT model (where MAT stands for *minimal adequate teacher*) introduced by Angluin [Ang87]. Since the goal in the original model is to learn a language rather than a series, it uses membership queries rather than coefficient queries.

4.2 Observation tables

The idea behind our learning algorithm stems from [Ang87], where it was used to devise a learning algorithm for regular string languages. Intuitively, the algorithm tries to build A_U without knowing the equivalence relation $\sim_U$ in its entirety. This construction is based on the following lemma.

Lemma 4.1 Let $s, s' \in supp(U)$. Then $s \sim_U s'$ if and only if $(U, c[\![s]\!]) \odot (U, s)^{-1} = (U, c[\![s']\!]) \odot (U, s')^{-1}$ for all $c \in C_\Sigma$.

Proof. By the definition of $\sim_U$, we have $s \sim_U s'$ if and only if there exists an $x \in G \setminus \{0\}$ such that $(U, c[\![s]\!]) = (U, c[\![s']\!]) \odot x$ for all $c \in C_\Sigma$. Clearly, $x = (U, s')^{-1} \odot (U, s)$ is such an x if $(U, c[\![s]\!]) \odot (U, s)^{-1} = (U, c[\![s']\!]) \odot (U, s')^{-1}$ (since G is zero-divisor free). Conversely, if $(U, c[\![s]\!]) = (U, c[\![s']\!]) \odot x$ for all $c \in C_\Sigma$, we obtain $x = (U, c[\![s]\!]) \odot (U, c[\![s']\!])^{-1}$ by choosing $c = \square$ (where the inverse is defined because $s' \in supp(U)$). Consequently, $(U, c[\![s]\!]) \odot (U, s)^{-1} = (U, c[\![s']\!]) \odot (U, s')^{-1}$ for all $c \in C_\Sigma$. $\square$

Lemma 4.1 motivates the following definition.

Definition 4.2 For $C \subseteq C_\Sigma$, we let $\sim_C$ denote the equivalence relation on $supp(U)$ such that, for all $s, s' \in supp(U)$, $s \sim_C s'$ holds if and only if $(U, c[\![s]\!]) \odot (U, s)^{-1} = (U, c[\![s']\!]) \odot (U, s')^{-1}$ for every $c \in C$.

Obviously, $\sim_C$ is indeed an equivalence relation. In order to enhance readability, we drop the subscript $\sim_C$ from the notation $[s]_{\sim_C}$ ($s \in R$) if C is clear from the context.

By Lemma 4.1, $\sim_{C_\Sigma}$ equals $\sim_U$. However, since U is assumed to be deterministically aa-recognizable, the number of equivalence classes of $\sim_U$ is finite. Hence, there is in fact a finite set $C \subseteq C_\Sigma$ such that $\sim_C = \sim_U$. This is of particular interest because, with C being finite, $s \sim_C s'$ can be decided using coefficient queries. The idea behind the learning algorithm is to determine such a set C, together with representatives of the equivalence classes. At each stage of the algorithm, the information gathered so far is recorded in a so-called observation table. This notion is defined next.

Definition 4.3 An *observation table* is a triple $T = (S, R, C)$, where

(1) S and R are finite subsets of T_Σ such that $S \subseteq R \subseteq \Sigma(S)$,

(2) $C \subseteq C_\Sigma$ with $|C| \leq |S|$,

(3) $R \subseteq supp(U)$, and

(4) $s \not\sim_C s'$ for every two distinct $s, s' \in S$.

The observation table T is *complete* if, for every $s \in R$, there is some $s' \in S$ such that $s \sim_C s'$. Owing to item (4), this s' is uniquely determined. We denote it by $select_T(s)$.

The requirement in item (3) corresponds to the avoidance of so-called dead states in [DH06b]: throughout the whole learning process, the learner solely considers trees whose coefficients are nonzero. This is important because it means that the inverse of a tree in R is always defined.

We mention here that, from the implementation point of view, there are two natural ways in which observation tables can be implemented. These may be called the implicit and the explicit one. The implicit implementation follows Definition 4.3 and records only S, R, and C, but not the value $(U, c[\![s]\!]) \odot (U, s)^{-1}$ for $c \in C$ and $s \in R$. Each time the algorithm needs to know this value, the teacher is asked two coefficient queries. While this behaviour is theoretically sound and saves memory space, it may obviously lead to an unnecessarily large number of coefficient queries. The explicit approach would therefore turn the observation table into a two-dimensional array indexed by the elements of R and C. The cell given by s and c contains the value $(U, c[\![s]\!]) \odot (U, s)^{-1}$. Thus, only one pair of coefficient queries is needed for each pair (s, c).

Lemma 4.4 Let $T = (S, R, C)$ be a complete observation table. For every $s \in S$, it holds that $\mathit{select}_T(s) = s$. Moreover, if $f[s_1, \ldots, s_k] \in R$, then $\mathit{select}_T(s_i) = s_i$ for every $i = 1, \ldots, k$.
Proof. The first statement is due to item (4) of Definition 4.3. Furthermore, for $f[s_1, \ldots, s_k] \in R$ item (1) of Definition 4.3 yields $s_1, \ldots, s_k \in S \subseteq R$. Therefore, the second statement follows from the first one. □

4.3 Synthesizing an aa-wta

From a complete observation table $T = (S, R, C)$, we may synthesize an aa-wta by following the construction of the minimal deterministic aa-wta described in Section 3.3, but replacing $\sim_U$ with $\sim_T$. The *aa-wta synthesized from* T is the aa-wta $A_T = (Q_T, \delta_T, \lambda_T)$ such that

- $Q_T = \{\langle s \rangle \mid s \in S\}$ and
- $\delta_T = (\delta_k \mid k \geq 0)$ and $\lambda_T = (\lambda_k \mid k \geq 0)$, where we define for every $k \geq 0$, $f \in \Sigma^{(k)}$, and $s_1, \ldots, s_k \in S$

$$\delta_k(f, \langle s_1 \rangle, \ldots, \langle s_k \rangle) = \begin{cases} \langle \mathit{select}_T(f[s_1, \ldots, s_k]) \rangle & \text{if } f[s_1, \ldots, s_k] \in R, \\ \text{undefined} & \text{otherwise} \end{cases}$$

and

$$\lambda_k(f, \langle s_1 \rangle, \ldots, \langle s_k \rangle) = \begin{cases} (U, f[s_1, \ldots, s_k]) \odot \bigodot_{i=1}^{k} (U, s_i)^{-1} \\ \qquad\qquad \text{if } f[s_1, \ldots, s_k] \in R, \\ \text{undefined} \quad \text{otherwise.} \end{cases}$$

To see that A_T is well defined, consider the definition of $\lambda_k(f, \langle s_1 \rangle, \ldots, \langle s_k \rangle)$ for some $f[s_1, \ldots, s_k] \in R$. Since $R \subseteq \Sigma(S)$ (cf. item (1) of Definition 4.3), we have that $s_i \in S \subseteq R$ for every $i = 1, \ldots, k$. By item (3) of Definition 4.3, it follows that $(U, s_i) \neq 0$ and hence, the inverses exist and λ_k is well defined. Note also that the finiteness of observation tables implies that A_T can be constructed effectively (using coefficient queries if an implicit implementation of T is used). In particular, the number of states is finite.

Lemma 4.5 Let $T = (S, R, C)$ be a complete observation table, and let $s \in R$. Then $\delta_T(s) = \langle \mathit{select}_T(s) \rangle$ and $(\underline{A_T}, s) = \lambda_T(s) = (U, s)$.
Proof. This follows by structural induction on $s = f[s_1, \ldots, s_k] \in R$. By the definition of observation tables, $R \subseteq \Sigma(S)$ and thus $s_1, \ldots, s_k \in S \subseteq R$. Therefore, we can apply the induction hypothesis to the s_i and obtain $\delta_T(s_i) = \langle \mathit{select}_T(s_i) \rangle$; by Lemma 4.4, this implies that $\mathit{select}_T(s_i) = s_i$. Hence,

$$\begin{aligned} \delta_T(s) &= \delta_k(f, \delta_T(s_1), \ldots, \delta_T(s_k)) \\ &= \delta_k(f, \langle \mathit{select}_T(s_1) \rangle, \ldots, \langle \mathit{select}_T(s_k) \rangle) \\ &= \delta_k(f, \langle s_1 \rangle, \ldots, \langle s_k \rangle) \\ &= \langle \mathit{select}_T(s) \rangle. \end{aligned}$$

This shows in particular that $s \in df(\delta_T)$, and thus $(\underline{A_T}, s) = \lambda_T(s)$. It remains to be checked that $\lambda_T(s) = (U, s)$, as follows:

$$\begin{aligned}
\lambda_T(s) &= \lambda_k(f, \delta_T(s_1), \ldots, \delta_T(s_k)) \odot \bigodot_{i=1}^k \lambda_T(s_i) \\
&= \lambda_k(f, \langle select_T(s_1)\rangle, \ldots, \langle select_T(s_k)\rangle) \odot \bigodot_{i=1}^k (U, s_i) \\
&= \lambda_k(f, \langle s_1\rangle, \ldots, \langle s_k\rangle) \odot \bigodot_{i=1}^k (U, s_i) \\
&= (U, s) \odot \bigodot_{i=1}^k (U, s_i)^{-1} \odot \bigodot_{i=1}^k (U, s_i) \\
&= (U, s).
\end{aligned}$$

□

4.4 The learning algorithm

The main procedure of our learning algorithm, the *learner*, is the following simple loop:

```
$T = (S, R, C)$ := $(\emptyset, \emptyset, \emptyset)$;
loop
    $A$ := SYNTHESIZE($T$);
    $t$ := COUNTEREXAMPLE($A$);   (ask equivalence query)
    if $t = \bot$ then return $A$   (teacher has approved A)
    else $T$ := EXTEND($T, t$)     (teacher has rejected A; extend T using t)
end loop
```

Here, SYNTHESIZE(T) builds and returns the aa-wta synthesized from T. The function EXTEND is the central part of the learner. It is based on the following intuition. If $\delta_T(s) = \langle s'\rangle$ for a subtree s of the counterexample t, this means that A_T assumes s to be equivalent to the tree $s' \in S$ (and if $\delta_T(s)$ is undefined, it assumes that $s \notin supp(U)$). By Lemma 4.5, this assumption is necessarily correct if $s \in S$, because then $s' = s$. Thus, as A_T fails to work correctly on t, there must be a subtree $s \notin S$ for which the assumption made is incorrect. The purpose of the function EXTEND is to search for a minimal such tree s, using a technique known as contradiction backtracking [Sha83]. This technique does not only find such a tree; it guarantees also that the returned tree s is an element of $\Sigma(S)$. This is ensured by modifying t during the search. In a bottom-up manner, EXTEND chooses an arbitrary subtree $s \in \Sigma(S) \setminus S$, in effect decomposing t into $c[\![s]\!]$. If s is not the sought subtree, then $t' = c[\![select_T(s)]\!]$ is a counterexample as well (as we shall prove in Lemma 5.1), and thus EXTEND calls itself recursively, with the argument t' instead of t. Otherwise, the table resulting from the addition of s (and perhaps c) to T is returned. The resulting pseudocode for EXTEND is this:

```
(1) function EXTEND($T, t$) where $T = (S, R, C)$
(2)    decompose $t$ into $t = c[\![s]\!]$ where $s = f[s_1, \ldots, s_k] \in \Sigma(S) \setminus S$;
(3)    if $s \in R$ then
          (use coefficient queries to check the following condition:)
(4)       if $(U, c[\![select_T(s)]\!]) \odot \lambda_T(select_T(s))^{-1} = (U, c[\![s]\!]) \odot \lambda_T(s)^{-1}$ then
            ($c[\![select_T(s)]\!]$ is also a counterexample)
(5)         return EXTEND($T, c[\![select_T(s)]\!]$)
          else (we have $select_T(s) \not\sim_{C \cup \{c\}} s$)
(6)         return COMPLETE($S \cup \{s\}, R, C \cup \{c\}$)
       else (we have $(\underline{A}, t) = (\underline{A}, s) = 0$ but $(U, t) \neq 0$ and thus $(U, s) \neq 0$)
(7)       return COMPLETE($S, R \cup \{s\}, C$)
```

The auxiliary function COMPLETE is simple. It makes an observation table (S, R, C) complete by checking, for every equivalence class $[s]$ with $s \in R$, whether $s \sim_C s'$ for some $s' \in S$. If this is not the case, it adds s to S.

Let us now reconsider the formal tree series $U \in \mathbb{Z}_\infty\langle\!\langle T_\Sigma \rangle\!\rangle$ from Example 3.3 and apply the learning algorithm to U. For an observation table T_i, we will denote its first, second, and third components by S_i, R_i, and C_i, respectively. Also, in order to enhance succinctness of the presentation, for a complete observation table $T = (S, R, C)$, we introduce the mapping $rep_T \colon S \to \mathcal{P}(R)$ which is defined for every $s \in S$ by $rep_T(s) = select_T^{-1}(s)$.

At the beginning of the execution of the algorithm we have

$$T_0 = (\emptyset, \emptyset, \emptyset) \quad \text{and} \quad A_{T_0} = (\emptyset, \delta_\emptyset, \lambda_\emptyset).$$

Since the support of the tree series recognized by A_{T_0} is the empty set, the teacher returns a counterexample, say, $t = a$, and the algorithm executes EXTEND(T_0, a). This results in

$$T_1 = (\{a\}, \{a\}, \emptyset) \quad \text{and} \quad A_{T_1} = (\{\langle a \rangle\}, \delta_1, \lambda_1)$$

with $rep_{T_1}(a) = \{a\}$, and δ_1 and λ_1 are specified as follows:

$$(\delta_1)_0(a) = \langle a \rangle, \ (\lambda_1)_0(a) = 1.$$

Again the teacher returns a counterexample, say b, and the algorithm executes EXTEND(T_1, b) with the following result:

$$T_2 = (\{a\}, \{a, b\}, \emptyset) \quad \text{and} \quad A_{T_2} = (\{\langle a \rangle\}, \delta_2, \lambda_2)$$

with $rep_{T_2}(a) = \{a, b\}$ (note that $select_{T_2}(b) = a$, because the set C_2 of contexts with respect to which equality has to be checked, is empty), and thus

$$(\delta_2)_0(a) = (\delta_2)_0(b) = \langle a \rangle, \ (\lambda_2)_0(a) = 1, \ (\lambda_2)_0(b) = 0.$$

Now the teacher may return the counterexample $t = f[a, b]$. EXTEND decomposes this into $t = f[a, \square][\![b]\!]$, i.e., $c = f[a, \square]$ and $s = b$. Since $s \in R_2$, EXTEND compares two expressions (cf. line (4)):

$$(U, f[a, \square][\![select_{T_2}(b)]\!]) + (-\lambda_{T_2}(select_{T_2}(b))) = (U, f[a, a]) - \lambda_{T_2}(a) = 3 - 1 = 2$$

$$(U, f[a, \square][\![b]\!]) + (-\lambda_{T_2}(b)) = 1 - 0 = 1.$$

The discrepancy reveals that $a \nsim_U b$, and also that c is a context witnessing this fact. Therefore, EXTEND returns the new complete observation table (cf. line (6)):

$$T_3 = (\{a, b\}, \{a, b\}, \{f[a, \Box]\}) \quad \text{and} \quad A_{T_3} = (\{\langle a\rangle, \langle b\rangle\}, \delta_3, \lambda_3)$$

with $rep_{T_3}(a) = \{a\}$ and $rep_{T_3}(b) = \{b\}$. Thus,

$$(\delta_3)_0(a) = \langle a\rangle,\ (\delta_3)_0(b) = \langle b\rangle,\ (\lambda_3)_0(a) = 1,\ (\lambda_3)_0(b) = 0.$$

In other words, the context $f[a, \Box]$ has separated the trees a and b. But still, the wta A_{T_3} rejects the input tree $f[a, b]$ (i.e., $\delta_3(f[a, b])$ is undefined). So, the teacher may again return $t = f[a, b]$ as the next counterexample. Now t is decomposed into $c = \Box$ and $s = f[a, b]$, and since $s \notin R_3$, the following complete observation table is returned in line (7):

$$T_4 = (\{a, b\}, \{a, b, f[a, b]\}, \{f[a, \Box]\}) \quad \text{and} \quad A_{T_4} = (\{\langle a\rangle, \langle b\rangle\}, \delta_4, \lambda_4)$$

with $rep_{T_4}(a) = \{a\}$ and $rep_{T_4}(b) = \{b, f[a, b]\}$. The transition and coefficient mappings of A_{T_4} are extensions of the corresponding mappings of A_{T_3}:

$$(\delta_4)_0(a) = \langle a\rangle,\ (\delta_4)_0(b) = (\delta_4)_2(f, \langle a\rangle, \langle b\rangle) = \langle b\rangle,$$

$$(\lambda_4)_0(a) = 1,\ (\lambda_4)_0(b) = (\lambda_4)_2(f, \langle a\rangle, \langle b\rangle) = 0.$$

In order to enhance readability we express these extensions of δ_3 and λ_3 in a slightly informal way, as follows:

$$\delta_4 = \delta_3 \cup \big[(\delta_4)_2(f, \langle a\rangle, \langle b\rangle) = \langle b\rangle\big]$$

$$\lambda_4 = \lambda_3 \cup \big[(\lambda_4)_2(f, \langle a\rangle, \langle b\rangle) = 0\big].$$

We will keep this habit also for the wta A_{T_5} through A_{T_7} which will be synthesized next.

Since A_{T_4} rejects $t = f[a, a]$, the teacher may choose t to be the next counterexample. The decomposition of t yields $c = \Box$ and $s = f[a, a]$ and, since $s \notin R_4$, Extend returns the following complete observation table:

$$T_5 = (\{a, b\}, \{a, b, f[a, b], f[a, a]\}, \{f[a, \Box]\}) \quad \text{and} \quad A_{T_5} = (\{\langle a\rangle, \langle b\rangle\}, \delta_5, \lambda_5)$$

with $rep_{T_5}(a) = \{a, f[a, a]\}$ and $rep_{T_5}(b) = \{b, f[a, b]\}$. Thus,

$$\delta_5 = \delta_4 \cup \big[(\delta_5)_2(f, \langle a\rangle, \langle a\rangle) = \langle a\rangle\big],$$

$$\lambda_5 = \lambda_4 \cup \big[(\lambda_5)_2(f, \langle a\rangle, \langle a\rangle) = 1\big].$$

Now the teacher may answer the equivalence query by returning, e.g., $t = f[f[a, b], a]$. This tree has a possible decomposition into $c = f[\Box, a]$ and $s = f[a, b]$. The comparison in line (4) yields

$$\begin{aligned}(U, f[\Box, a]\llbracket select_{T_5}(f[a, b])\rrbracket) + (-\lambda_{T_5}(select_{T_5}(f[a, b])))\\ = (U, f[b, a]) - \lambda_{T_5}(b) = 1 - 0 = 1\end{aligned}$$

$$(U, f[\Box, a]\llbracket f[a, b]\rrbracket) + (-\lambda_{T_5}(f[a, b])) = 2 - 1 = 1.$$

Thus, the algorithm continues with line (5) and executes $\texttt{EXTEND}(T_5, f[b,a])$. The result is the following complete observation table with the corresponding synthesized wta:

$$T_6 = (\{a,b\}, \{a, b, f[a,b], f[a,a], f[b,a]\}, \{f[a,\Box]\})$$

and

$$A_{T_6} = (\{\langle a\rangle, \langle b\rangle\}, \delta_6, \lambda_6)$$

with $rep_{T_6}(a) = \{a, f[a,a])\}$ and $rep_{T_6}(b) = \{b, f[a,b], f[b,a]\}$ and

$$\delta_6 = \delta_5 \cup \big[(\delta_6)_2(f, \langle b\rangle, \langle a\rangle) = \langle b\rangle\big],$$

$$\lambda_6 = \lambda_5 \cup \big[(\lambda_6)_2(f, \langle b\rangle, \langle a\rangle) = 0\big].$$

Now $(U, f[b,b]) = 0$ whereas $(\underline{A_{T_6}}, f[b,b]) = \infty$. Hence, the teacher might return $t = f[b,b]$ as the next counterexample. The extension of T_6 with t yields the new complete observation table:

$$T_7 = (\{a,b\}, \{a, b, f[a,b], f[a,a], f[b,a], f[b,b]\}, \{f[a,\Box]\})$$

and

$$A_{T_7} = (\{\langle a\rangle, \langle b\rangle\}, \delta_7, \lambda_7)$$

with $rep_{T_7}(a) = \{a, f[a,a]\}$ and $rep_{T_7}(b) = \{b, f[a,b], f[b,a], f[b,b]\}$ and

$$\delta_7 = \delta_6 \cup [(\delta_7)_2(f, \langle b\rangle, \langle b\rangle) = \langle b\rangle],$$

$$\lambda_7 = \lambda_6 \cup [(\lambda_7)_2(f, \langle b\rangle, \langle b\rangle) = 0].$$

The next equivalence query reveals that $\texttt{COUNTEREXAMPLE}(A_{T_7}) = \bot$, because $U = \underline{A_{T_7}}$. In fact, up to the renaming $\langle a\rangle \mapsto c$ and $\langle b\rangle \mapsto p$ of states, the wta A_{T_7} equals the wta A shown in Example 3.3. $\Box$

5 Correctness and complexity

In this section, the correctness of the learner is proved formally. We show that the learner will always return the minimal aa-wta recognizing U. Afterwards, we discuss briefly the runtime complexity of the learner.

We start by proving a lemma which shows that the first two return statements in $\texttt{EXTEND}$ are appropriate.

Lemma 5.1 Let $T = (S, R, C)$ be a complete observation table and $A = \texttt{SYNTHESIZE}(T)$. Moreover, let $c \in C_\Sigma$ and $s \in R$ be such that $(\underline{A}, c[\![s]\!]) \neq (U, c[\![s]\!])$. If

$$(U, c[\![select_T(s)]\!]) \odot (U, select_T(s))^{-1} = U(c[\![s]\!]) \odot (U, s)^{-1},$$

then $(\underline{A}, c[\![select_T(s)]\!]) \neq (U, c[\![select_T(s)]\!])$. Otherwise, $(S \cup \{s\}, R, C \cup \{c\})$ is an observation table.

Proof. Let $A = (Q, \delta, \lambda)$. By Lemma 4.5, we have $\delta(s) = \mathit{select}_T(s) = \delta(\mathit{select}_T(s))$. Using Lemma 3.4, this implies that $\lambda(c[\![s]\!]) \odot \lambda(s)^{-1} \doteq \lambda(c[\![\delta(s)]\!]) \doteq \lambda(c[\![\delta(\mathit{select}_T(s))]\!]) \doteq \lambda(c[\![\mathit{select}_T(s)]\!]) \odot \lambda(\mathit{select}_T(s))^{-1}$ and thus $(\underline{A}, c[\![s]\!]) \odot \lambda(s)^{-1} = (\underline{A}, c[\![\mathit{select}_T(s)]\!]) \odot \lambda(\mathit{select}_T(s))^{-1}$.

Now recall that, by Lemma 4.5, also $\lambda(s) = (U, s)$ and $\lambda(\mathit{select}_T(s)) = (U, \mathit{select}_T(s))$. Thus, if $(U, c[\![\mathit{select}_T(s)]\!]) \odot \lambda(\mathit{select}_T(s))^{-1} = U(c[\![s]\!]) \odot \lambda(s)^{-1}$, then we get

$$\begin{aligned}(\underline{A}, c[\![\mathit{select}_T(s)]\!]) \odot \lambda(\mathit{select}_T(s))^{-1} &= (\underline{A}, c[\![s]\!]) \odot \lambda(s)^{-1}\\ &\neq (U, c[\![s]\!]) \odot \lambda(s)^{-1}\\ &= (U, c[\![\mathit{select}_T(s)]\!]) \odot \lambda(\mathit{select}_T(s))^{-1},\end{aligned}$$

i.e., $(\underline{A}, c[\![\mathit{select}_T(s)]\!]) \neq (U, c[\![\mathit{select}_T(s)]\!])$, as claimed.

If $(U, c[\![\mathit{select}_T(s)]\!]) \odot \lambda(\mathit{select}_T(s))^{-1} \neq U(c[\![s]\!]) \odot \lambda(s)^{-1}$, then we have to prove that $(S \cup \{s\}, R, C \cup \{c\})$ satisfies (1)–(4) in Definition 4.3. Requirements (1)–(3) are fulfilled by the choice of s, since T is an observation table. Requirement (4) is satisfied because the assumed inequality, together with Lemma 4.5, yields $(U, c[\![\mathit{select}_T(s)]\!]) \odot (U, \mathit{select}_T(s))^{-1} \neq U(c[\![s]\!]) \odot (U, s)^{-1}$, i.e., $s \not\sim_{C \cup \{c\}} \mathit{select}_T(s)$. □

Our next task is to prove that the last return statement in Extend is appropriate as well.

Lemma 5.2 Let $T = (S, R, C)$ be a complete observation table and $A =$ Synthesize(T). If $c \in C_\Sigma$ and $s \in \Sigma(S) \setminus R$ are such that $(\underline{A}, c[\![s]\!]) \neq (U, c[\![s]\!])$, then $(S, R \cup \{s\}, C)$ is an observation table.

Proof. The only requirement of Definition 4.3 which is not obviously satisfied is requirement (3). To check that $s \in \mathit{supp}(U)$, let $s = f[s_1, \ldots, s_k]$. By the fact that $s \in \Sigma(S) \setminus R$, $\delta_k(f, \langle s_1 \rangle, \ldots, \langle s_k \rangle)$ is undefined. Hence, by Lemma 4.5, $\delta(s)$ is undefined and thus $(\underline{A}, c[\![s]\!]) = 0$, which shows that $(U, c[\![s]\!]) \neq 0$. Since U is deterministically aa-recognizable, the latter means that $s \in \mathit{supp}(U)$ (by Lemma 3.4). □

The following easy lemma shows that each call of Extend terminates.

Lemma 5.3 For every complete observation table $T = (S, R, C)$ and every tree $t \in T_\Sigma$ with $(U, t) \neq (\underline{A_T}, t)$, the call Extend$(T, t)$ terminates.

Proof. Let $v(t)$ be the number of (occurrences of) subtrees of t which are not in S. If Extend calls itself recursively with the second parameter $t' = c[\![\mathit{select}_T(s)]\!]$, then $s \in R \setminus S$. By definition, $\mathit{select}_T(s) \in S$, which shows that $v(t') = v(t) - 1$. Consequently, there can be at most $v(t)$ recursive calls. (In fact, by Lemma 4.5, there cannot be more than $v(t) - 1$ calls since no counterexample can be in S.) □

Theorem 5.4 The learner terminates and returns the minimal aa-wta A_U recognizing U.

Proof. Until it terminates, the learner calls the function Extend in every iteration of its main loop. By Lemma 5.3 every such call of Extend terminates and produces a new observation table T. Initially, $T = (\emptyset, \emptyset, \emptyset)$ which is complete. Assuming that T is a complete observation table, it follows from Lemmata 5.1 and 5.2 that $\textsc{Extend}(T, t)$ is a complete observation table as well, for every counterexample t. Thus, after every iteration, $T = (S, R, C)$ is a complete observation table. By the results of Section 3.7 and the construction of A_T, it is clear that A_T is the minimal aa-wta recognizing $\underline{A_T}$. To complete the proof, it is thus sufficient to show that the learner terminates (as it will do so only if there does not exist any counterexample).

Let $z \in \mathbb{N}$ be the number of equivalence classes of $\sim_U$. Firstly, Definition 4.3(4) yields $s \not\sim_C s'$ for all distinct $s, s' \in S$, and thus $s \not\sim_U s'$. This shows that, throughout the execution of the learner, we have $|S| \leq z$. Secondly, the inclusion $R \subseteq \Sigma(S)$ yields a bound on the maximum number of elements R can have (namely $|\Sigma| \cdot r^k$, where r is the maximum rank of symbols in Σ). Thirdly, we have $|C| \leq |S| \leq z$ by requirement (2) of Definition 4.3. However, each execution of the main loop adds a new element to at least one of the components of the observation table. Hence, termination is guaranteed. $\square$

Let us now briefly discuss the runtime complexity of the learner. For this, we assume that the learner is implemented on a random-access machine. Furthermore, we assume that $x \odot y$ and x^{-1} can be computed in constant time for all $x, y \in G$, and that each element of G can be stored in a single register (using some appropriate representation). The time needed by the teacher to answer a coefficient or equivalence query is not taken into account; below, every query is assumed to be answered instantaneously.

In Section 4 of [DH06b], the running time of the learner for regular tree languages proposed in that paper is studied in detail. The parameters considered are the maximum rank r of symbols in Σ, the maximum size m of counterexamples, the number $|Q|$ of states of the sought automaton, and $|\delta|$, the number of transitions of this automaton. In the following, we consider the same parameters (where $|\delta| = \sum_{k \in \mathbb{N}} |df(\delta_k)|$).

The learner proposed in [DH06b] is shown to run in time $O(r \cdot |Q| \cdot |\delta| \cdot (|Q| + m))$ if certain optimizations to the basic algorithm are made. For example, if $\textsc{Extend}$ has extended the observation table according to line (8) of its definition, there is no need to recompute the synthesized aa-wta from scratch. Instead, the components δ and λ (and perhaps Q) of the previous aa-wta can be extended in the obvious way. Another important observation is that the set S can be stored efficiently as a directed acyclic graph consisting of exactly $|S|$ nodes (since $S \subseteq \Sigma(S)$). A careful examination of the arguments and techniques used in [DH06b] reveals that, with only minor modifications, the same ideas yield an efficient implementation of the learner proposed in this paper. With one exception, the resulting estimations of the running time are unaffected.

The mentioned exception concerns the fact that, in [DH06b], observations yield binary values – either $c[\![s]\!]$ is a member of the tree language in question, or it is not. As a consequence, the $|C|$ cells of the observation table that store the observations for a tree $s \in R$ can be regarded as a bit string of length $|C|$, and can thus be stored in a single register. Using this representation, comparisons can be made in a single step if an explicit observation table is used.

In the present case, the cells of the observation table contain arbitrary elements of G, which implies that we have to use lists of length $|C|$ rather than single registers in order to store the observations made for each $s \in R$. As a consequence, comparisons and similar operations now require $|C| \leq |S| \leq |Q|$ steps rather than a single step. For this reason, the running time of the algorithm is increased by the factor $|Q|$ if compared with the one in [DH06b] (regardless of whether we use implicit or explicit observation tables). More precisely, we have the following theorem.

Theorem 5.5 If $A_U = (Q, \delta, \lambda, F)$ is the minimal aa-wta recognizing U, then the learner runs in time $O(r \cdot |Q|^2 \cdot |\delta| \cdot (|Q| + m))$, where m is the maximum size of counterexamples returned by the teacher, and r is the maximum rank of symbols in Σ.

References

[Ang87] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[Ang04] D. Angluin. Queries revisited. *Theoretical Computer Science*, 313:175–194, 2004.

[BFGM05] B. Borchardt, Z. Fülöp, Z. Gazdag, and A. Maletti. Bounds for Tree Automata with Polynomial Costs. *J. Autom. Lang. Comb.*, 2005. accepted for publication.

[Bor03] B. Borchardt. The Myhill-Nerode Theorem for Recognizable Tree Series. In *7th International Conference on Developments in Language Theory, DLT'03, Szeged, Hungary, July 7-11, 2003, Proceedings*, volume 2710 of *LNCS*, pages 146–158. Springer-Verlag, July 2003.

[Bor04a] B. Borchardt. A Pumping Lemma and Decidability Problems for Recognizable Tree Series. *Acta Cybernet.*, 16(4):509–544, 2004.

[Bor04b] B. Borchardt. *The Theory of Recognizable Tree Series.* PhD thesis, TU Dresden, 2004.

[Boz91] S. Bozapalidis. Effective construction of the syntactic algebra of a recognizable series on trees. *Acta Informatica*, 28:351–363, 1991.

[Boz99] S. Bozapalidis. Equational Elements in Additive Algebras. *Theory of Computing Systems*, 32(1):1–33, 1999.

[BR82] J. Berstel and C. Reutenauer. Recognizable formal power series on trees. *Theoret. Comput. Sci.*, 18(2):115–148, 1982.

[BR87] P. Berman and R. Roos. Learning one-counter languages in polynomial time (extended abstract). In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 61–67. IEEE Press, 1987.

[BR88] J. Berstel and Ch. Reutenauer. *Rational Series and Their Languages*, volume 12 of *EATCS-Monographs*. Springer-Verlag, 1988.

[BV03] B. Borchardt and H. Vogler. Determinization of Finite State Weighted Tree Automata. *Journal of Automata, Languages and Combinatorics*, 8(3):417–463, 2003.

[DH06a] F. Drewes and J. Högberg. Extensions of a MAT learner for regular tree languages. In M.J. Minock, P. Eklund, and H. Lindgren, editors, *Proc. 23rd Annual Workshop of the Swedish Artificial Intelligence Society (SAIS 2006)*, pages 35–44, 2006.

[DH06b] F. Drewes and J. Högberg. Query learning of regular tree languages: how to avoid dead states. *Theory of Computing Systems*, 2006. to appear.

[DPV05] M. Droste, C. Pech, and H. Vogler. A Kleene theorem for weighted tree automata. *Theory of Computing Systems*, 38:1–38, 2005.

[DV05] M. Droste and H. Vogler. Weighted tree automata and weighted logics. Technical Report TUD-FI-05-10, July 2005, TU Dresden, 2005.

[Eil74] S. Eilenberg. *Automata, Languages, and Machines – Volume A*, volume 59 of *Pure and Applied Mathematics*. Academic Press, 1974.

[ÉK03] Z. Ésik and W. Kuich. Formal tree series. *J. of Automata, Languages, and Combinatorics*, 8(2):219–285, 2003.

[Fer02] H. Fernau. Even linear matrix languages: Formal language properties and grammatical inference. *Theoretical Computer Science*, 289:425–456, 2002.

[FR95] A.F. Fahmy and R. Roos. Efficient learning of real time one-counter automata. In K.P. Jantke, T. Shinohara, and T. Zeugmann, editors, *Proceedings of the 6th International Workshop on Algorithmic Learning Theory (ALT'95)*, volume 997 of Lecture Notes in Artificial Intelligence, pages 25–40, Springer, Berlin, Heidelberg, 1995.

[FV89] Z. Fülöp and S. Vágvölgyi. Congruential tree languages are the same as recognizable tree languages. *Bulletin of EATCS*, 30:175–185, 1989.

[Gol67] E.M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[GS84] F. Gécseg and M. Steinby. *Tree Automata.* Akadémiai Kiadó, Budapest, 1984.

[GS97] F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer-Verlag, 1997.

[Ish90] H. Ishizaka. Polynomial time learnability of simple deterministic languages. *Machine Learning*, 5:151–164, 1990.

[Koz92] D. Kozen. On the Myhill-Nerode theorem for trees. *EATCS Bulletin*, 47:170–173, June 1992.

[KS86] W. Kuich and A. Salomaa. *Semirings, Automata, Languages.* EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1986.

[Kui97] W. Kuich. Semirings and formal power series: Their relevance to formal languages and automata. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol. 1*, chapter 9, pages 609–677. Springer-Verlag, 1997.

[Kui98] W. Kuich. Formal power series over trees. In S. Bozapalidis, editor, *3rd International Conference on Developments in Language Theory, DLT 1997, Thessaloniki, Greece, Proceedings*, pages 61–101. Aristotle University of Thessaloniki, 1998.

[Pec03a] Chr. Pech. *Kleene-Type Results for Weighted Tree Automata.* PhD thesis, TU Dresden, 2003.

[Pec03b] Chr. Pech. Kleene's theorem for weighted tree-automata. In *14th International Symposium on Fundamentals of Computation Theory FCT 2003, Malmö, Sweden*, number 2751 in LNCS, pages 387–399. Springer-Verlag, 2003.

[Sak90] Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoret. Comput. Sci.*, 76:223–242, 1990.

[Sei90] H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal of Computing*, 19(3):424–437, 1990.

[Sei94] H. Seidl. Finite tree automata with cost functions. *Theoret. Comput. Sci.*, 126(1):113–142, 1994.

[Sha83] E.Y. Shapiro. *Algorithmic Program Debugging.* MIT Press, Cambridge, MA, USA, 1983.

[SS78] A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series.* Texts and Monographs in Computer Science, Springer-Verlag, 1978.

[SY93] H. Shirakawa and T. Yokomori. Polynomial-time MAT learning of c-deterministic context-free grammars. *Transactions of the Information Processing Society of Japan*, 34:380–390, 1993.

[Wec78] W. Wechler. *The Concept of Fuzziness in Automata and Language Theory.* Studien zur Algebra und ihre Anwendungen. Akademie-Verlag Berlin, 5. edition, 1978.

[Yok94] T. Yokomori. Learning nondeterministic finite automata from queries and counterexamples. In K. Furukawa, D. Michie, and S. Muggleton, editors, *Machine Intelligence*, volume 13, chapter 7, pages 169–189. Clarendon Press, Oxford, 1994.

[Yu97] S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 2, pages 41–110. Springer-Verlag, 1997.