Technical Report 08-2

A learning algorithm for multi-dimensional trees, or: Learning beyond context-freeness

Anna Kasprzik

University of Trier

May 23, 2008

Abstract

We generalize a learning algorithm by Drewes and Högberg [1] for regular tree languages based on a learning model proposed by Angluin [2] to recognizable tree languages of arbitrarily many dimensions, so-called multi-dimensional trees. Multi-dimensional trees over multi-dimensional tree domains have been defined by Rogers [3, 4]. However, since the algorithm by Drewes and Högberg relies on classical finite state automata, these structures have to be represented in another form to make them a suitable input for the algorithm: We give a new representation for multi-dimensional trees which establishes them as a direct generalization of classical trees over a partitioned alphabet, and then show that with this notation Drewes' and Högberg's algorithm is able to learn tree languages of arbitrarily many dimensions. Via the correspondence between trees and string languages known as the "yield operation" this is equivalent to the statement that now even some string language classes beyond context-freeness have become learnable with respect to Angluin's learning model as well.

Keywords: MAT learning, multi-dimensional trees, finite-state methods

1 Introduction

In the area of grammatical inference the problem of how to algorithmically infer (or "learn") a description of a formal language (e.g., a grammar or an automaton) on the basis of given examples or other information on that language is considered. Several learning models have been formulated, and based on those quite an amount of learning algorithms (mainly for regular languages or subclasses thereof) have been developed.

In one of those models, which was proposed by Angluin [2] along with a P-time learning algorithm L^* for regular string languages, the "learner" is helped by a "minimally adequate teacher" (MAT) who can answer two types of queries, namely if a given word is or is not a member of the language U to be learned, and, for some finite-state automaton $\mathcal{A}$, if $\mathcal{A}$ correctly recognizes U. If not, the teacher will return a counterexample. The algorithm L^* has been adapted by Sakakibara [5] to skeletal regular tree languages (i.e., regular sets of trees where the inner nodes are not labeled) and then generalized by Drewes and Högberg [1] to regular tree languages throughout. As regular tree languages are a well-known generalization of regular string languages, this is a logical step.

We will generalize the algorithm by Drewes and Högberg even further to recognizable tree languages of arbitrarily many dimensions (sets of so-called *multi-dimensional trees*). Multi-dimensional trees based on multi-dimensional tree domains have been defined by Rogers [3, 4], along with finite-state automata recognizing these trees. Labeled one-dimensional trees correspond to strings, and the automata recognizing them are equivalent to classical finite-state automata. The automata recognizing labeled two-dimensional trees are equivalent to classical finite-state tree automata.

Every multi-dimensional tree language has a set of strings – the *string yields* of its elements – associated with it which is obtained by reducing the number of dimensions of the trees step by step, down to the first, only retaining the outermost nodes and their connecting structure in each step (see [3] or Subsection 3.2 for a definition). As is well known, the sets of string yields of the languages recognized by (two-dimensional) finite-state tree automata coincide with the class of context-free languages. The sets obtained when reducing the number of dimensions of recognizable three-dimensional tree languages by one have an interesting linguistic aspect: They correspond exactly to the sets of trees generated by *(non-strict) Tree Adjoining Grammars* (see [3, 4]), a special kind of tree formalism in which trees are built via *adjunction*, an operation which can be seen as a particular form of context-free tree rewriting. TAGs have been developed by Joshi [6] in connection with studies on the formal treatment of natural languages. Joshi [6] claimed the least class of formal languages containing all natural languages to be situated between the context-free and the context-sensitive languages in the Chomsky Hierarchy, and named it the class of *mildly context-sensitive languages*. The string sets associated with TAGs fulfil all necessary conditions for this class. TAGs are considered the standard model for mild context-sensitivity and are the foundation of a considerable amount of current work in applied computational linguistics. Rogers [4] conjectures that there might also be some linguistic phenomena that can best be handled via structures of more than three dimensions, and gives an amelioration of the standard TAG account of modifiers using four dimensions (see [3]).

The classes of sets of string yields associated with the recognizable multi-dimensional tree languages ordered by the number of dimensions form a (proper) infinite hierarchy properly contained in the context-sensitive class, with the classes of context-free languages (sets of string yields of two-dimensional tree sets) and the string languages associated with TAGs (sets of string yields of three-dimensional tree sets) as the first two steps. According to Rogers [3, 4], this hierarchy coincides with Weir's Control Language Hierarchy [7].

It is a consequence of these correspondencies that by processing recognizable higher-dimensional descriptions of non-regular string languages instead of the string sets themselves, finite-state meth-

ods become applicable again (see [8, 9]). Thus, just as by adapting Angluin's learning algorithm for regular string languages [2] to (skeletal) regular tree languages [5, 1] context-free string languages have been made MAT-learnable, generalizing the adapted algorithm to recognizable tree languages of arbitrarily many dimensions and then recurring to the concept of yield can make even string language classes beyond context-freeness learnable under Angluin's MAT learning model as well.

As the learning algorithm by Drewes and Högberg [1] is based on classical finite-state tree automata and consequently on the concept of trees as terms over a partitioned alphabet, but Rogers' definition of multi-dimensional trees is based on tree domains, the algorithm cannot be used on these structures without representing them in another form first. We will therefore give a new term-like representation for multi-dimensional trees, which was introduced in [8] and which establishes them as a direct generalization of classical trees, along with an adapted definition of finite-state automata and of the yield operation, and then show that this innovation enables Drewes' and Högberg's algorithm to learn languages of trees of arbitrarily many dimensions by proving that despite the modified input all its essential properties (including the ability to yield the desired output) stay preserved.

2 The learning algorithm for trees by Drewes and Högberg

In this section, we are going to describe the learner for trees by Drewes and Högberg [1]. First of all, we need some basic notions about trees (mostly taken in a shortened, modified form from [1]).

A *ranked alphabet* is a finite set of symbols, each associated with a rank $n \in \mathbb{N}$ (including 0). By Σ_n we denote the set of all symbols in Σ with rank n. Traditionally, every symbol is associated with a single rank only, but it is just as possible to admit several ranks for one symbol (see for example [5]), as long as there is a maximal admissible rank and the alphabet stays finite.

The set T_Σ of all trees over Σ is defined inductively as the smallest set of expressions such that $f[t_1, \ldots, t_n] \in T_\Sigma$ for every $f \in \Sigma_n$ and all $t_1, \ldots, t_n \in T_\Sigma$. $t_1, \ldots, t_n$ are the *direct subtrees* of the tree. The set $subtrees(t)$ consists of t itself and all subtrees of its direct subtrees. Given a set T of trees, $\Sigma(T)$ denotes the set of all trees of the form $f[t_1, \ldots, t_n]$ such that $f \in \Sigma_n$ for some n and $t_1, \ldots, t_n \in T$. A subset of T_Σ is called a tree language.

Let $\square$ be a special symbol of rank 0. A tree $c \in T_{\Sigma \cup \{\square\}}$ in which $\square$ occurs exactly once is a *context*, the set of all contexts over Σ is denoted by C_Σ. For $c \in C_\Sigma$ and $s \in T_\Sigma$, $c[[s]]$ denotes the tree obtained by substituting s for $\square$ in c. $depth(c)$ is the length of the path from the root to $\square$.

A *(total, deterministic) bottom-up finite-state tree automaton (fta)* is a tuple $\mathcal{A} = (\Sigma, Q, \delta, F)$ with ranked input alphabet Σ, finite state set Q, transition function δ assigning to every $f \in \Sigma_n$ and all $q_1, \ldots, q_n \in Q$ a state $\delta(q_1 \cdots q_n, f) \in Q$, and set of accepting states $F \subseteq Q$. The transition function extends to trees: $\delta : T_\Sigma \longrightarrow Q$ is defined such that if $t = f[t_1, \ldots, t_n] \in T_\Sigma$ then $\delta(t) = \delta(\delta(t_1) \cdots \delta(t_n), f)$. The set of trees accepted by $\mathcal{A}$ is $L(\mathcal{A}) = \{t \in T_\Sigma | \delta(t) \in F\}$. Such a tree language is called *regular*.

It is well known that the Myhill-Nerode theorem carries over to regular tree languages: Let $L \subseteq T_\Sigma$. Given two trees $s, s' \in T_\Sigma$, let $s \sim_L s'$ iff for every $c \in C_\Sigma$, either both of $c[[s]]$ and $c[[s']]$ are in L or none of them is. Obviously, $\sim_L$ is an equivalence relation on T_Σ. The equivalence class containing $s \in T_\Sigma$ is denoted by $[s]_L$. The *index* of L equals $|\{[s]_L | s \in T_\Sigma\}|$. The Myhill-Nerode theorem states that L is a regular tree language iff L is of finite index iff L is the union of all equivalence classes $[s]_L$ with $s \in L$. It follows from this that for every fta $\mathcal{A}$, $L(\mathcal{A})$ is of finite index. Conversely, if a tree language is of finite index, we can easily build an fta $\mathcal{A}_L$ recognizing L, with the states being the equivalence classes of L, $F = \{[s]_L | s \in L\}$, and, given some $f \in \Sigma_k$ and states $[s_1]_L, \ldots, [s_k]_L$, $\delta_L([s_1]_L, \ldots, [s_k]_L, f) = [f[s_1, \ldots, s_k]]_L$. Moreover, this fta is the unique minimal fta recognizing L, up to a bijective renaming of states (see for example [10], p. 35, for a proof).

As indicated before, Drewes and Högberg [1] have designed a learning algorithm for regular tree languages, based on the one for strings by Angluin [2]. As strings can be seen as special trees over an alphabet containing only symbols of rank 1 or 0 (ε being the only constant – the string *abc* is then noted as the expression $c(b(a(\varepsilon)))$, for example), this algorithm represents a generalization.

The aim of the learner is to construct an fta that recognizes an unknown regular tree language $U \subseteq T_\Sigma$. The learner is helped by a teacher who is able to perform two tasks: The teacher can check whether $t \in U$ for some $t \in T_\Sigma$, and, given an fta $\mathcal{A}$, the teacher can return a *counterexample*$(\mathcal{A}) \in (U \setminus L(\mathcal{A})) \cup (L(\mathcal{A}) \setminus U)$. At any stage of the computation, the learner maintains two sets $S \subseteq T_\Sigma$ and $C \subseteq C_\Sigma$ satisfying certain conditions. Intuitively, one may imagine that the learner builds a table whose rows are indexed by the elements of $S \cup \Sigma(S)$ and the columns by the elements of C. The cell in row s and column c contains a truth value indicating whether $c[[s]] \in U$.

Definition 1 *The pair (S, C) ($S \subseteq T_\Sigma, C \subseteq C_\Sigma$ finite, C non-empty) is called an* observation table *if the following conditions hold:*

- *For every tree $f[s_1, \ldots, s_n]$ the trees $s_1, \ldots, s_n$ are in S as well – S is* subtree-closed*, and*
- *for every context c_0 of the form $c[[f[s_1, \ldots, s_{i-1}, \Box, s_{i+1}, \ldots, s_n]]] \in C$, c is in C as well and $s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n \in S$ – we say that C is* generalization-closed*.*

The elements of S can be seen as candidates for representatives of the equivalence classes of $\sim_U$, and the elements of C can be seen as witnesses that these representatives do indeed belong to different equivalence classes.

Given an observation table $T = (S, C)$ and a tree $s \in S \cup \Sigma(S)$, the observed behaviour of s is denoted by $obs_T(s)$ (formally, $obs_T(s)$ denotes the function $obs : C \longrightarrow \{1, 0\}$ such that $obs(c) = 1$ iff $c[[s]] \in U$ for all $c \in C$).

Definition 2 *Let $T = (S, C)$ be an observation table. T is* closed *if $obs_T(\Sigma(S)) \subseteq obs_T(S)$, and* consistent *if, for all $f \in \Sigma_n$ and all $s_1, \ldots, s_n, s'_1, \ldots, s'_n \in S$, if $obs_T(s_i) = obs_T(s'_i)$ for all i with $1 \leq i \leq n$ then $obs_T(f[s_1, \ldots, s_n]) = obs_T(f[s'_1, \ldots, s'_n])$.*

These two properties are essential for observation tables if they are to be used by the learner to build a candidate for the desired automaton: From a closed and consistent observation table $T = (S, C)$ one can synthesize an fta $\mathcal{A}_T$ whose set of states is $Q_T = \{obs_T(s) | s \in S\}$, the set of accepting states is $F_T = \{obs_T(s) | s \in S \cap U\}$, and $\delta_T(obs_T(s_1) \cdots obs_T(s_n), f) = obs_T(f[s_1, \ldots, s_n])$ for all $f \in \Sigma_n$ and $s_1, \ldots, s_n \in S$. Drewes and Högberg [1] formulate the following lemma, adapted from a corresponding one in [2]:

Lemma 1 *Let $T = (S, C)$ be a closed and consistent observation table. Then*

- *$\delta(s) = obs_T(s)$ for all $s \in S \cup \Sigma(S)$, and*
- *for all $s \in S \cup \Sigma(S)$ and all $c \in C$, $\mathcal{A}_T$ accepts $c[[s]]$ iff $c[[s]] \in U$. Moreover, $\mathcal{A}_T$ is the unique minimal fta with this property (up to a bijective renaming of states).*

We will prove a similar lemma for our generalized learning algorithm in Section 4.

The algorithm by Drewes and Högberg [1] can be seen below. I is the index of U.[1] The learner starts with a table $T_1 = (\{a\}, \{\Box\})$ (for some $a \in \Sigma_0$). The procedure `CLOSURE` adds suitable

[1] Drewes and Högberg [1] use I as a termination criterion. However, this does not affect the computation as such – they prove that their algorithm always returns the desired automaton in time, i.e., it never halts without result because of the termination criterion alone – and is therefore equivalent to assuming that the teacher, when asked for a counterexample, first checks if $\mathcal{A} = \mathcal{A}_U$, which is the termination criterion for the algorithm by Angluin [2].

candidates to S as long as T is not closed (which corresponds to asking the teacher membership queries for these candidates and noting the result in the table). The procedure `RESOLVE` adds elements to C as long as T is not consistent. The procedure `EXTEND` synthesizes an fta from the current observation table (assume that an operation $synthesize(T)$ just exists) and asks the teacher for a counterexample. If the counterexample is unnecessarily large it is "pruned" via the procedure `EXTRACT`, which is an amelioration introduced by Drewes and Högberg [1] with respect to the original learner for strings by Angluin [2]. The counterexample is then added to the table, along with its membership status. See [1] for details.

```
T = (S, C) := ({a}, {□}) for some arbitrary a ∈ Σ_0;
while | {obs_T(s) | s ∈ S} | < I do
    if T is not closed then T := CLOSURE(T)
    else if T is not consistent then T := RESOLVE(T)
    else T := EXTEND(T)
end while;
return A_T;

procedure CLOSURE(T) where T = (S, C)
    find s ∈ Σ(S) such that obs_T(s) ∉ obs_T(S);
    return (S ∪ {s}, C);

procedure RESOLVE(T) where T = (S, C)
    find c[[s]], c[[s']] ∈ Σ(S) where s, s' ∈ S and depth(c) = 1 such that
       obs_T(c[[s]]) ≠ obs_T(c[[s']]) and obs_T(s) = obs_T(s');
    find t, t' ∈ S such that
       obs_T(t) = obs_T(c[[s]]) and obs_T(t') = obs_T(c[[s']]);
    find c' ∈ C such that obs_T(t)(c') ≠ obs_T(t')(c');
    return (S, C ∪ {c'[[c]]});

procedure EXTEND(T) where T = (S, C)
    A_T := synthesize(T);
    return EXTRACT(T, counterexample(A_T));

procedure EXTRACT(T, t) where T = (S, C)
    choose c ∈ C_Σ and s ∈ subtrees(t) ∩ (Σ(S) \ S) such that t = c[[s]];
    if there exists s' ∈ S such that
       obs_T(s') = obs_T(s) and t ∈ U ⇔ c[[s']] ∈ U then
       return EXTRACT(T, c[[s']]);
    else
       return (S ∪ {s}, C)
    end if;
```

Let $T_l = (S_l, C_l)$ be the table obtained after $l - 1$ steps. Note that according to Drewes and Högberg [1] (and as is easy to see) the procedures `CLOSURE`, `RESOLVE`, and `EXTEND` all guarantee that each constructed observation table satisfies the following conditions: (A) T_l is indeed an observation table, (B) for all distinct trees $s, s' \in S_l$, $s \not\sim_U s'$, (C) $|S_l| + |C_l| = l + 1$, and (D) $|C_l| \leq |obs_{T_l}(S_l)|$.

Drewes and Högberg [1] prove that their algorithm always terminates after less than $2I$ loop executions and returns the desired fta (see their paper for the proof).

In the following section we will introduce multi-dimensional trees and some related concepts.

3 Multi-dimensional trees and automata

3.1 Multi-dimensional trees as defined by Rogers [3, 4]

Starting from a definition of ordinary trees based on two-dimensional tree domains, Rogers [3, 4] generalizes the concept both downwards (to strings and points) and upwards and defines *labeled multi-dimensional trees* based on a hierarchy of multi-dimensional tree domains:

Definition 3 *Let* ${}^d1$ *be the class of all* d*th-order sequences of* 1*s:* ${}^01 := \{1\}$*, and* ${}^{n+1}1$ *is the smallest set satisfying (i)* $\langle\rangle \in {}^{n+1}1$*, and (ii) if* $\langle x_1, \ldots, x_l\rangle \in {}^{n+1}1$ *and* $y \in {}^n1$*, then* $\langle x_1, \ldots, x_l, y\rangle \in {}^{n+1}1$*. Let* $\mathbb{T}^0 := \{\emptyset, \{1\}\}$ *(point domains). A* $(d+1)$*-dimensional tree domain is a set of hereditarily prefix closed* $(d+1)$*st-order sequences of* 1*s, i.e.,* $\mathsf{T} \in \mathbb{T}^{d+1}$ *iff*

- $\mathsf{T} \subseteq {}^{d+1}1$,
- $\forall s, t \in {}^{d+1}1 : s \cdot t \in \mathsf{T} \Rightarrow s \in \mathsf{T}$,
- $\forall s \in {}^{d+1}1 : \{w \in {}^d1 | s \cdot \langle w\rangle \in \mathsf{T}\} \in \mathbb{T}^d$.

A Σ*-labeled* $\mathsf{T}d$ *(*d-dimensional tree*) is a pair* $\langle T, \tau\rangle$*) where* T *is a* d*-dimensional tree domain and* $\tau : T \longrightarrow \Sigma$ *is an assignment of labels in the (non-partitioned) alphabet* Σ *to nodes in* T*. We will denote the class of all* Σ*-labeled* $\mathsf{T}d$ *as* $\mathbb{T}^d_\Sigma$.

Every d-dimensional tree can be conceived to be built up from one or more d-dimensional *local trees*, that is, trees of depth at most one in their major dimension. Each of these smaller trees consists of a root and an arbitrarily large $(d-1)$-dimensional "child tree" consisting of the root's children (a formal definition of the set $\mathbb{T}^{d,loc}_\Sigma$ of all local trees over some alphabet Σ would be for example $\mathbb{T}^{d,loc}_\Sigma = \{\langle T, \tau\rangle | \langle T, \tau\rangle$ is a Σ-labeled $\mathsf{T}d$, and $\forall s \in T : |s| \leq 1\}$). Local strings (i.e., one-dimensional trees), for example, consist of a root and a point as its child tree. Local two-dimensional trees consist of a root and a string. Local three-dimensional trees would have a pyramidal form, with a two-dimensional tree as its base. There are also trivial local trees (consisting of a single root), and even empty ones. Composite trees can be built from local ones by identifying the root of one local tree with a node in the child tree of another (and adapting the addresses in order to incorporate them into the newly created tree domain). Figure 1 shows examples of local and composite trees for the first four steps of the hierarchy (only some composite trees are labeled, and in the three-dimensional case, only the addresses of nodes that do not appear in the rightmost local tree as well are given, for clarity. ε_d denotes an empty sequence of order d).

Rogers [4] defines automata for labeled $\mathsf{T}d$s as well:

Definition 4 *A* $\mathsf{T}d$ *automaton with finite state set* Q *and (non-ranked) alphabet* Σ *is a finite set of triples* $\mathcal{A}^d \subseteq \Sigma \times Q \times \mathbb{T}^{d-1}_Q$.

The interpretation of a triple $\langle \sigma, q, \mathcal{T}\rangle \in \mathcal{A}^d$ is that if a node of a $\mathsf{T}d$ is labeled with σ and $\mathcal{T}$ encodes the assignment of states to its children, then that node may be assigned state q. A *run* of a $\mathsf{T}d$ automaton on a Σ-labeled $\mathsf{T}d$ $\mathcal{T} = \langle T, \tau\rangle$ is a mapping $r : T \longrightarrow Q$ in which each assignment is licensed by $\mathcal{A}^d$. Note that this implies that a maximal node (wrt the major dimension, i.e., a leaf) labeled with σ may be assigned state q only if there is a triple $\langle \sigma, q, \emptyset\rangle \in \mathcal{A}^d$, where $\emptyset$ is the empty $\mathsf{T}(d-1)$. If $F \in Q$ is the set of accepting states, then the set of (finite) Σ-labeled $\mathsf{T}d$ recognized by $\mathcal{A}^d$ is that set for which there is a run of $\mathcal{A}^d$ that assigns the root a state in F.

As mentioned in the Introduction, $\mathsf{T}1$ automata correspond to finite-state automata for strings, i.e., they recognize the regular languages. $\mathsf{T}2$ automata correspond to (non-deterministic) finite-state automata for trees, i.e., they recognize the regular tree languages, the associated string sets of

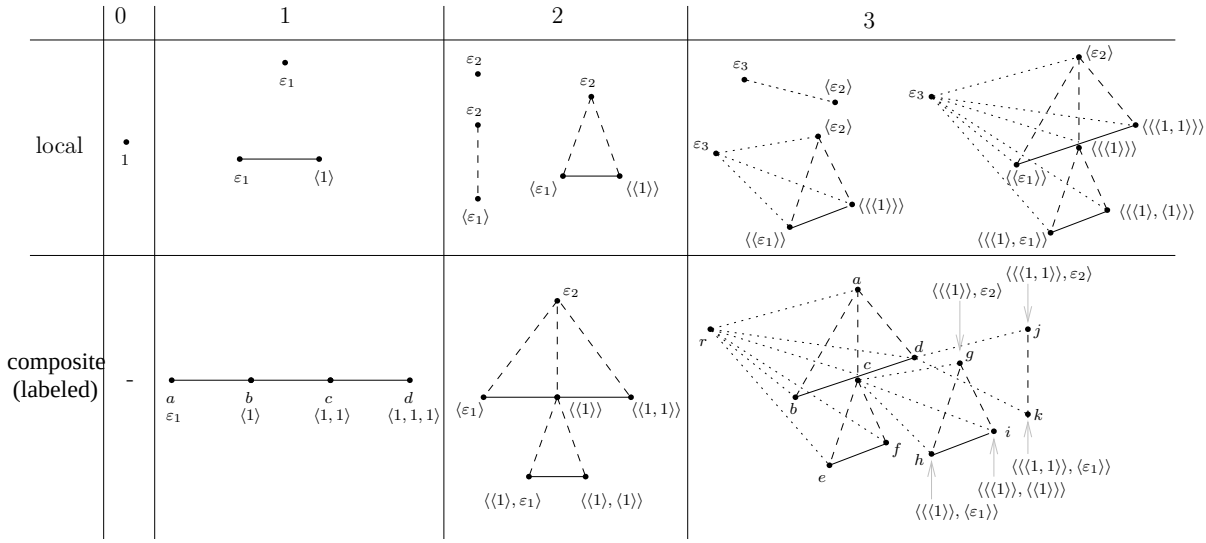

Figure 1: Local and composite trees for $d = 0, 1, 2, 3$

which are the context-free languages. For $d \geq 3$, Td automata recognize languages of d-dimensional trees whose sets of string yields are situated between the classes of context-free and context-sensitive languages in the Chomsky Hierarchy, where for every d the class of string yields of the d-dimensional tree languages is properly contained in the next (i.e., for $d + 1$).

3.2 Multi-dimensional trees as terms

In this subsection we will give a representation for multi-dimensional trees which establishes them as a direct generalization of the one on which (classical) finite-state tree automata are based, i.e., one that allows multi-dimensional trees to be noted as expressions over a partitioned alphabet. This notation was first introduced in [8].

We use finite d-dimensional tree labeling alphabets Σ^d where each symbol $f \in \Sigma^d$ is associated with at least one unlabeled $(d-1)$-dimensional tree t specifying the admissible child structure for a root labeled with f (note that as before it is just as possible to associate several admissible child structure trees with one symbol). t can be given in any form suitable for trees, as long as it is compatible with the existence of an empty tree. For consistency's sake we will use the definition of multi-dimensional trees given below and write t as an expression over a special kind of "alphabet" containing just one symbol ρ for which any child structure is admissible.

Let Σ^d_t for $d \geq 1$ be the set of all symbols associated with t and Σ^0 a set of constant symbols. The set of all d-dimensional trees $\mathbb{T}_{\Sigma^d}$ can then be defined inductively as follows:

Definition 5 *Let ε^d be the empty d-dimensional tree. Then*

- $\mathbb{T}_{\Sigma^0} := \{\varepsilon^0\} \cup \Sigma^0$, *and*
- *for $d \geq 1$: $\mathbb{T}_{\Sigma^d}$ is the smallest set such that $\varepsilon^d \in \mathbb{T}_{\Sigma^d}$ and $f[t_1, \ldots, t_n]_t \in \mathbb{T}_{\Sigma^d}$ for every $f \in \Sigma^d_t$, n the number of nodes in t, $t_1, \ldots, t_n \in \mathbb{T}_{\Sigma^d}$ and $t_1, \ldots, t_n$ are rooted breadth-first in that order[2] at the nodes of t.*

Multi-dimensional trees in this notation can be translated one-to-one into trees in Rogers' notation and vice versa – see [8] for the translation and proof.

[2]This is an ad hoc settlement, any other spatial arrangement would do as well.

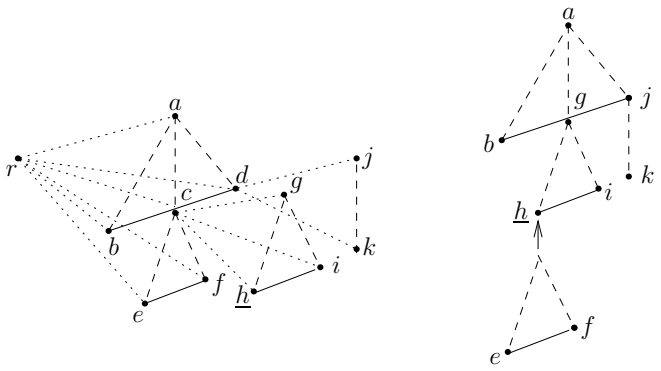

Figure 2: Ambiguity in the yield for $d \geq 3$ (resolved by marked splicing points)

For some tree $t_p = f[t_1, \ldots, t_n]_t$ with $f \in \Sigma_t^d$, $t_1, \ldots, t_n$ are the direct subtrees of the tree, and the rest of the usual tree terminology can be applied in a similar manner. Also, for some fixed d, let $\Box$ be a special symbol associated with ε^{d-1} (i.e., a leaf label). A tree $c \in \mathbb{T}_{\Sigma^d \cup \{\Box\}}$ in which $\Box$ occurs exactly once is still called a context, and the set of all contexts over Σ^d is denoted by $\mathbb{C}_{\Sigma^d}$. $c[[s]]$ for $c \in \mathbb{C}_{\Sigma^d}$ and $s \in \mathbb{T}_{\Sigma^d}$ is defined via substitution as before as well.

We can now represent automata for d-dimensional trees as direct generalization of classical fta's:

Definition 6 *A (total, deterministic) finite-state d-dimensional tree automaton is a quadruple $\mathcal{A}^d = (\Sigma^d, Q, \delta, F)$ with input alphabet Σ^d, finite state set Q, set of accepting states $F \subseteq Q$ and transition function δ with with $\delta(t(q_1, \ldots, q_n), f) \in Q$ for every $f \in \Sigma_t^d$ where $t(q_1, \ldots, q_n)$ encodes the assignment of states to the nodes of t (i.e., $t(q_1, \ldots, q_n)$ is isomorphic to t and its nodes are labeled with $q_1, \ldots, q_n$ breadth-first in that order if Q is taken as a partitioned alphabet in which every element is associated with all the child structures it occurs with in δ). The transition function extends to d-dimensional trees: $\delta : \mathbb{T}_{\Sigma^d} \longrightarrow Q$ is defined such that if $t_p = f[t_1, \ldots, t_n]_t \in \mathbb{T}_{\Sigma^d}$ then $\delta(t_p) = \delta(t(\delta(t_1), \ldots, \delta(t_n)), f)$. The set of trees accepted by $\mathcal{A}^d$ is $L(\mathcal{A}^d) = \{t_p \in \mathbb{T}_{\Sigma^d} | \delta(t_p) \in F\}$.*

The equivalence between this definition and the one by Rogers [4] is easy to see. For two corresponding automata $\mathcal{A}^d = (\Sigma^d, Q, \delta, F)$ and $\mathcal{A}_R^d \subseteq \Sigma_R \times Q_R \times \mathbb{T}_{Q_R}^{d-1}$ (with accepting state set F_R) in the two notations the sets of states Q and Q_R, and F and F_R coincide, the construction of Σ_R from Σ^d is trivial, and Σ^d is constructed from $\mathcal{A}_R^d$ as follows: $f \in \Sigma_t^d$ for all triples $\langle f, q, t_0 \rangle \in \mathcal{A}_R^d$, where $t \in \mathbb{T}_{\{\rho\}^{d-1}}$ is isomorphic to t_0. Most importantly, there is a one-to-one correspondence between the elements of $\mathcal{A}_R^d$ and δ: Every triple $\langle f, q, t_0 \rangle \in \mathcal{A}_R^d$ can be translated to an assignment $\delta(\Psi(t_0), f) = q$ of $\mathcal{A}^d$, and every assignment $\delta(t(q_1, \ldots, q_n), f) = q$ of $\mathcal{A}^d$ to a triple $\langle f, q, \Phi(t(q_1, \ldots, q_n)) \rangle \in \mathcal{A}_R^d$, where Φ and Ψ are translations from one notation into the other (see [8] for a definition). From this and from the identical state sets it follows that $L(\mathcal{A}_R^d) = \Psi(L(\mathcal{A}^d))$ and $L(\mathcal{A}^d) = \Phi(L(\mathcal{A}_R^d))$.

Finally, we give a definition of the yield operation for multi-dimensional trees in the new notation. As for $d \geq 3$ the yield is not unambiguous (see Figure 2), the structures have to be enriched with additional information. Assume that, for $d \geq 2$, in every tree $t_p \in \mathbb{T}_{\Sigma^d}$ every labeling symbol $f \in \Sigma^d$ is indexed with a set $S \subseteq \{2, \ldots, d\}$. If $x \in S$ then we call a node labeled by f_S a *foot node for the $(x-1)$-dimensional yield of t_p*. For every subtree t_q of t_p the distribution of these foot nodes must fulfil certain conditions:

(1) If t_q has depth 0 the index set in its root label must contain d, otherwise $t_q = f_S[t_1, \ldots, t_n]_t$ with $f \in \Sigma_t^d$, $S \subseteq \{2, \ldots, d\}$, and $t_1, \ldots, t_n \in \mathbb{T}_{\Sigma^d}$ must have exactly one direct subtree $t_i \in \{t_1, \ldots, t_n\}$ whose root labeling symbol is indexed with a set containing d and this

subtree is attached to a *leaf* in t. In both cases, we will refer to that root as the d-dimensional foot node of t_q.

(2) The foot nodes are distributed in such a way that for every n-dimensional yield of t_p with $n < d$, condition (1) is fulfilled as well.

For $d \geq 2$, the direct yield of a tree $t_p \in \mathbb{T}_{\Sigma^d}$ is then defined recursively as

$$yd_{d-1}(t_p) = \begin{cases} \varepsilon^{d-1} & \text{for } t_p = \varepsilon^d, \\ a_S & \text{for } t_p = a_S \text{ with } a \in \Sigma^d_{\varepsilon^{d-1}} \text{ and } S \subseteq \{2, \ldots, d\}, \\ op_{t_p}(t_1) & \text{for } t_p = f_S[t_1, \ldots, t_n]_t \text{ with } t_1, \ldots, t_n \in \mathbb{T}_{\Sigma^d},\ f \in \Sigma^d_t, \\ & t \neq \varepsilon^{d-1}, \text{ and } S \subseteq \{2, \ldots, d\}, \end{cases}$$

where $op_{t_p}(t_i)$ for $t_i \in \{t_1, \ldots, t_n\}$ is defined as the $(d-1)$-dimensional tree that is obtained by replacing the d-dimensional foot node of t_i in $yd_{d-1}(t_i)$ by $e_R[op_{t_p}(t_j), \ldots, op_{t_p}(t_k)]_{t_x}$, where e_R with $e \in \Sigma^d$ and $R \subseteq \{2, \ldots, d\}$ is the label of the foot node, t_x is the $(d-2)$-dimensional child structure of the node at which t_i is attached in t and $t_j, \ldots, t_k$ are the direct subtrees of t_p that are attached (breadth-first in that order) at the nodes of t_x. The result $yd_{d-1}(t_p)$ is a $(d-1)$-dimensional tree over an alphabet Σ^{d-1} containing at least all the node labels in $yd_{d-1}(t_p)$, each associated at least with the child structures it occurs with. Obviously, the string yield of a d-dimensional tree for $d \geq 2$ can be obtained by taking the direct yield $d-1$ times.

Example 1 defines an automaton $\mathcal{A}^3_{ww}$ recognizing a three-dimensional tree language whose set of string yields $yd_1(L(\mathcal{A}^3_{ww}))$ equals the copy language $L_{ww} = \{ww | w \in \{a,b\}^+\}$:

Example 1 *$\mathcal{A}^3_{ww} = (\Sigma^3, \{q_a, q_b, q_g, q_y, q_z, q_f, q_x\}, \delta, \{q_f\})$ where $\Sigma^3 = \{a, b, f, g, g_{\{3\}}\}$ with $a, b, f, g, g_{\{3\}} \in \Sigma^3_{\varepsilon^2}$ and $f \in \Sigma^3_{t_1}$ for $t_1 = \rho[]_{\varepsilon^1}, \rho[\rho[]_{\varepsilon^0}]_{\rho[]_{\varepsilon^0}}]_{\rho[\rho[]_{\varepsilon^0}]_\rho}$ and $f \in \Sigma^3_{t_2}$ for $t_2 = \rho[\rho[]_{\varepsilon^1}, \rho[\rho[]_{\varepsilon^1}, \rho[]_{\varepsilon^1}]_{\rho[\rho[]_{\varepsilon^0}]_\rho}]_{\rho[\rho[]_{\varepsilon^0}]_\rho}$. Note that Σ^3 contains symbols that have already been distinguished by the index sets relevant for the yield (however, only index sets containing 3 have been considered, as the distribution of foot nodes for the string yield is never ambiguous). δ is defined as follows:*

$$\begin{aligned} \delta(\varepsilon^2, a) &= q_a & \qquad \delta(t_1(q_g, q_a, q_z, q_a)) &= q_f \\ \delta(\varepsilon^2, b) &= q_b & \qquad \delta(t_1(q_g, q_b, q_z, q_b)) &= q_f \\ \delta(\varepsilon^2, f) &= q_z & \qquad \delta(t_2(q_g, q_a, q_z, q_y, q_a)) &= q_z \\ \delta(\varepsilon^2, g) &= q_g & \qquad \delta(t_2(q_g, q_b, q_z, q_y, q_b)) &= q_z \\ \delta(\varepsilon^2, g_{\{3\}}) &= q_y \end{aligned}$$

and $\delta(t_0, x) = q_x$ for all other admissible t_0 and all symbols $x \in \Sigma^3$. Figure 3 shows t_1 and t_2 on the left, three trees $t_a, t_b, t_c \in L(\mathcal{A}^3_{ww})$ in the middle, and the two-dimensional yield for t_c on the right, whose one-dimensional yield is the string abab.

With the slightly adapted definitions of contexts and automata, the Myhill-Nerode theorem (see Section 2) carries over quite naturally to multi-dimensional trees, and consequently, for every recognizable d-dimensional tree language L there exists a unique minimal automaton $\mathcal{A}^d_L$ recognizing L. It is this fact that enables us to give a learning algorithm for languages of trees of arbitrarily many dimensions based on the same principle as the one by Drewes and Högberg [1].

4 The learner for multi-dimensional tree languages

We will now generalize the learning algorithm for regular tree languages by Drewes and Högberg [1] to recognizable tree languages of arbitrarily many dimensions. The necessary concepts for the learning algorithm can be adapted in an obvious way (assume that t has n nodes):

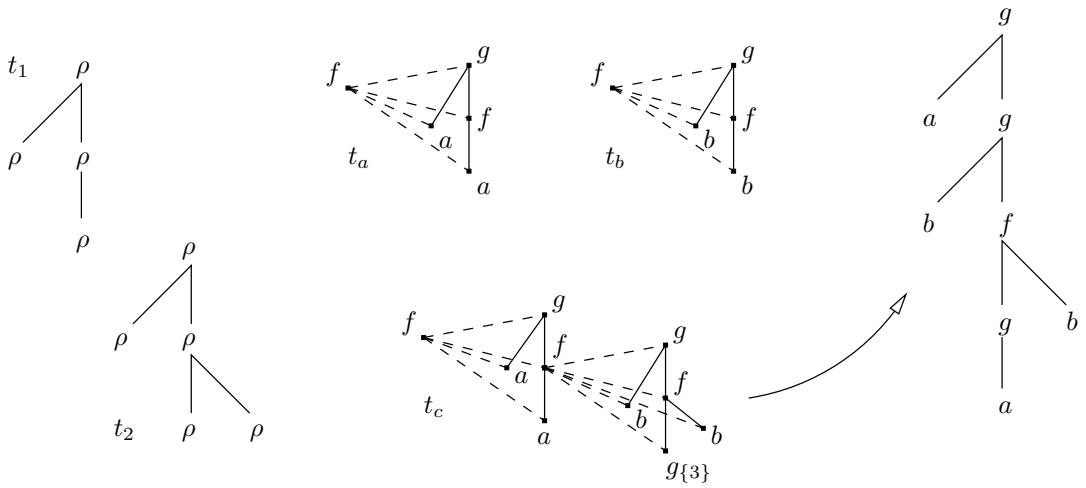

Figure 3: Example 1

- *Subtree-closed*: For every tree $f[t_1, \ldots, t_n]_t \in S$, the trees $t_1, \ldots, t_n$ are in S as well.
- *Generalization-closed*: For every context of the form $c[[f[t_1, \ldots, t_{i-1}, \Box, t_{i+1}, \ldots, t_n]_t]]$, c is in C as well and $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$ are in S.
- $\Sigma^d(S)$ denotes the set of all trees of the form $f[t_1, \ldots, t_n]_t$ such that $f \in \Sigma^d_t$ for some t and $t_1, \ldots, t_n \in S$.
- *Closed (for an observation table T)*: T is closed if $obs_T(\Sigma^d(S)) \subseteq obs_T(S)$.
- *Consistent (for an observation table T)*: For all $f \in \Sigma^d_t$ and all $s_1, \ldots, s_n$, $s'_1, \ldots, s'_n \in S$, if $obs_T(s_i) = obs_T(s'_i)$ for all i with $1 \leq i \leq n$ then $obs_T(f[s_1, \ldots, s_n]_t) = obs_T(f[s'_1, \ldots, s'_n]_t)$.

From a closed and consistent observation table $T = (S, C)$ one can derive a finite-state d-dimensional tree automaton $\mathcal{A}^d_T$ whose set of states is $Q_T = \{obs_T(s) | s \in S\}$, the set of accepting states is $F_T = \{obs_T(s) | s \in S \cap U\}$, and $\delta_T(t(obs_T(s_1), \ldots, obs_T(s_n)), f) = obs_T(f[s_1, \ldots, s_n]_t)$ for all $f \in \Sigma^d_t$ and $s_1, \ldots, s_n \in S$.

With this settled, the learning algorithm for recognizable d-dimensional tree languages (not containing the empty tree, and for some $d \geq 1$, since $\mathbb{T}_{\Sigma^0}$ is finite, i.e., trivial to learn) is very easy to formulate; in fact, it is identical to the one given in Section 2 (just change $\Sigma(S)$ to $\Sigma^d(S)$ throughout and start with $T = (\{a\}, \{\Box\})$ for some arbitrary $a \in \Sigma^d_{\varepsilon^{d-1}}$). We will prove that the validity of Lemma 1, repeated below in a slightly adapted form as Lemma 2, hasn't been changed by our generalization to trees of arbitrarily many dimensions. The proofs are inspired by the corresponding ones in [2].

Lemma 2 *Let $T = (S, C)$ be a closed, consistent observation table. Then*

(a) $\delta_T(t_p) = obs_T(t_p)$ for all $t_p \in S \cup \Sigma^d(S)$,

(b) for all $t_p \in S \cup \Sigma^d(S)$ and all contexts $c \in C$, $\mathcal{A}^d_T$ accepts $c[[t_p]]$ iff $c[[t_p]] \in U$, and

(c) $\mathcal{A}^d_T$ is the unique minimal automaton with property (b), up to a bijective renaming of states.

Proofs. (a) is easily proved by induction via the definitions of δ and δ_T: It certainly holds for all $a \in \Sigma^d_{\varepsilon^{d-1}} \cap S$ (trees consisting of one node in S), as $\delta_T(a) = \delta_T(\varepsilon^{d-1}, a) = obs_T(a)$. Now let

$t_p = f[s_1, \ldots, s_n]_t$ for an arbitrary $f \in \Sigma_t^d$ and $s_1, \ldots, s_n \in S \cup \Sigma^d(S)$. As $t_p \in S \cup \Sigma^d(S)$, $s_1, \ldots, s_n$ must be in S (which is clear for $t_p \in \Sigma^d(S)$ – for $t_p \in S$ recall that S is subtree-closed). If (a) holds for $s_1, \ldots, s_n$ then it also holds for t_p, as $\delta_T(f[s_1, \ldots, s_n]_t) = \delta_T(t(\delta_T(s_1), \ldots, \delta_T(s_n)), f) = \delta_T(t(obs_T(s_1), \ldots, obs_T(s_n)), f) = obs_T(f[s_1, \ldots, s_n]_t)$. ■

(b) is proved by induction over the depth of the contexts in C. For $c = \Box$ and all $t_p \in S \cup \Sigma^d(S)$, $\delta_T(c[[t_p]]) = \delta_T(t_p) = obs_T(t_p)$ by (a). t_p is either in S or in $\Sigma^d(S)$. Case 1, $t_p \in S$: $\delta_T(t_p) \in F \Leftrightarrow obs_T(t_p) \in F \Leftrightarrow obs_T(t_p) \in \{obs_T(s) | s \in S \cap U\} \Leftrightarrow t_p \in S \cap U \Leftrightarrow t_p \in U$. Case 2, $t_p \in \Sigma^d(S)$: As T is closed, there exists $t_q \in S$ with $obs_T(t_p) = obs_T(t_q)$, and thus $\delta_T(t_p) = \delta_T(t_q)$ (and the rest of the argument runs as in Case 1). This proves (b) for $c = \Box$.

Now assume that $c \in C$ is of depth $k+1$, and (b) holds for all contexts in C up to depth k and all $t_p \in S \cup \Sigma^d(S)$. As C is generalization-closed, there exists $c_2 \in C$ of depth k and $s_1, \ldots, s_n \in S$ such that $c = c_2[[f[s_1, \ldots, s_{i-1}, \Box, s_{i+1}, \ldots, s_n]_t]]$ for some $f \in \Sigma_t^d$. (b) holds for c_2, i.e., $\delta_T(c_2[[t_p]]) \in F \Leftrightarrow c_2[[t_p]] \in U$ for all $t_p \in S \cup \Sigma^d(S)$. Again, as T is closed, there exists $t_q \in S$ with $\delta_T(t_p) = obs_T(t_p) = obs_T(t_q) = \delta_T(t_q)$. Obviously, $f[s_1, \ldots, s_{i-1}, t_q, s_{i+1}, \ldots, s_n]_t$ is in $\Sigma^d(S)$. Then (with $\delta_T(t_q) = \delta_T(t_p)$, and as T is consistent) $\delta_T(c_2[[f[s_1, \ldots, s_{i-1}, t_q, s_{i+1}, \ldots, s_n]_t]]) = \delta_T(c_2[[f[s_1, \ldots, s_{i-1}, t_p, s_{i+1}, \ldots, s_n]_t]]) \in F \Leftrightarrow c_2[[f[s_1, \ldots, s_{i-1}, t_p, s_{i+1}, \ldots, s_n]_t]] \in U \Leftrightarrow c[[t_p]] \in U$, which proves (b) for all $c \in C$ and all $t_p \in S \cup \Sigma^d(S)$. ■

(c) is equivalent to the claim that any other automaton $\mathcal{A}^{d\prime} = (\Sigma^d, Q', \delta', F')$ consistent with T that has equally many or fewer states than $\mathcal{A}_T^d$ is isomorphic to $\mathcal{A}_T^d$. Let $\mathcal{A}_T^d$ have n states. Define, for each $q' \in Q'$, $obs_T(q')$ to be the finite function $g : C \longrightarrow \{0, 1\}$ such that $g(c) = 1$ iff $\delta'(c[[q']]) \in F'$, where $\delta'(c[[q']]) = \delta'(c[[t]])$ for all t with $\delta'(t) = q'$. Since $\mathcal{A}^{d\prime}$ is consistent with T, for each $t_p \in S \cup \Sigma^d(S)$ and each $c \in C$, $\delta'(c[[t_p]]) \in F'$ iff $obs_T(t_p)(c) = 1$, which also means that $\delta'(c[[\delta'(t_p)]]) \in F'$ iff $obs_T(t_p)(c) = 1$, so $obs_T(\delta'(t_p)) = obs_T(t_p)$. As t_p ranges over all of S, $obs_T(\delta'(t_p))$ ranges over all of Q, so $\mathcal{A}^{d\prime}$ must have at least, i.e., exactly, n states.

Thus, for each $t_p \in S$ there is a unique $q' \in Q'$ such that $obs_T(t_p) = obs_T(q')$, namely $\delta'(t_p)$. Define for each $t_p \in S$, $\phi(obs_T(t_p)) = \delta'(t_p)$. This mapping is bijective. We must verify that it preserves the transition function and maps F to F'. For $s_1, \ldots, s_n \in S$ and $f \in \Sigma_t^d$, let $t_p \in S$ such that $obs_T(f[s_1, \ldots, s_n]_t) = obs_T(t_p)$. Then $\phi(\delta(t(obs_T(s_1), \ldots, obs_T(s_n)), f)) = \phi(obs_T(f[s_1, \ldots, s_n]_t)) = \phi(obs_T(t_p)) = \delta'(t_p)$, and also $\delta'(t(\phi(obs_T(s_1)), \ldots, \phi(obs_T(s_n))), f) = \delta'(t(\delta'(s_1), \ldots, \delta'(s_n)), f) = \delta'(f[s_1, \ldots, s_n]_t)$. Since $obs_T(\delta'(t_p)) = obs_T(\delta'(f[s_1, \ldots, s_n]_t))$, $\delta'(t_p)$ and $\delta'(f[s_1, \ldots, s_n]_t)$ must be the same state of $\mathcal{A}^{d\prime}$, and so we can conclude that $\phi(\delta(t[obs_T(s_1), \ldots, obs_T(s_n)],\ f)) = \delta'(t[\phi(obs_T(s_1)), \ldots, \phi(obs_T(s_n))], f)$ for all $s_1, \ldots, s_n \in S$ and $f \in \Sigma_t^d$. To complete the proof we must see that ϕ maps F to F', but this is clear since if $obs_T(t_p) \in F$ then $t_p \in U$ for all $t_p \in S$, so as $\phi(obs_T(t_p))$ is mapped to a state q' with $obs_T(q') = obs_T(t_p)$, q' must be in F. Conversely, if $obs_T(t_p)$ is mapped to a state $q' \in F'$, then since $obs_T(q') = obs_T(t_p)$, $t_p \in U$, so $obs_T(t_p) \in F$. ϕ is indeed an isomorphism, and (c) is proved. ■

Theorem 3 *The learner returns $\mathcal{A}_U^d$ after less than $2I$ loop executions.*

The proof stays the same as in [1]: According to property (D) of the obtained observation tables there cannot be more contexts in C_l than there are trees in S_l, for all l. Since (C) states that $|C_l| + |S_l| = l + 1$, this means $|S_l| > l/2$. We also know that the learner halts when S_l has I elements (by (B)), so we conclude that it will halt before $l = 2I$.

Now, let $\mathcal{A}_{T_m}^d$ be the returned automaton. Then T_m is a closed, consistent observation table and $\mathcal{A}_{T_m}^d$ is the unique minimal automaton such that, for all $s \in S_m$ and $c \in C_m$, $c[[s]] \in L(\mathcal{A}_{T_m}^d)$ iff $c[[s]] \in U$ (see Lemma 2(b)). However, $\mathcal{A}_U^d$ has the same property and the same number of states, so $\mathcal{A}_{T_m}^d = \mathcal{A}_U^d$ up to a bijective renaming of states. ■

We will now sketch an example run of the algorithm for the language $L(\mathcal{A}_{ww}^3)$ from Example 1.

Example 2 *The learner starts with* $T_1 = (\{a\}, \{\Box\})$*, and* $obs_T(a) = 0$*.* T_1 *is closed and consistent, so the learner proposes* $\mathcal{A}_{T_1}$*, which accepts nothing, and gets the counterexample* $t_b = f[g, b, f, b]_{t_1}$ *(see Figure 3), from which the procedure* `EXTRACT` *derives* T_2*.* T_2 *is closed, but not consistent (for example,* $obs_T(a) = 0$ *and* $obs_T(b) = 0$*, but* $obs_T(c[[a]]) = 0$ *and* $obs_T(c[[b]]) = 1$ *for* $c = f[g, b, f, \Box]_{t_1}$*). Several invocations of* `RESOLVE` *yield* T_3*, from which the learner synthesizes an automaton* $\mathcal{A}_{T_3}$ *with five states and one accepting state* $obs_T(t_b)$*, and gets the counterexample* $t_c = f[g, a, f[g, b, f, g_{\{3\}}, b]_{t_2}, a]_{t_1}$ *(see Figure 3). The procedure* `EXTRACT` *adds two more rows to* T_3*, and* `RESOLVE` *another column, yielding* T_4*, which is closed and consistent.* $\mathcal{A}_{T_4}$ *has* $I = 7$ *states (which is the termination criterion, see Section 2) and recognizes* $L(\mathcal{A}^3_{ww})$*.*

T_2	$\Box$
a	0
b	0
f	0
g	0
t_b	1

T_3	$\Box$	c_1	c_2	c_3
a	0	0	0	0
b	0	1	0	0
f	0	0	1	0
g	0	0	0	1
t_b	1	0	0	0

T_4	$\Box$	c_1	c_2	c_3	c_4
a	0	0	0	0	0
b	0	1	0	0	0
f	0	0	1	0	0
g	0	0	0	1	0
t_b	1	0	0	0	0
$g_{\{3\}}$	0	0	0	0	1
t_c	1	0	1	0	0

$t_b = f[g, b, f, b]_{t_1}$
$t_c = f[g, a, f[g, b, f, g_{\{3\}}, b]_{t_2}, a]_{t_1}$
$t_1 = \rho[\rho[]_{\varepsilon^1}, \rho[\rho[]_{\varepsilon^0}]_{\rho[]_{\varepsilon^0}}]_{\rho[\rho[]_{\varepsilon^0}]_\rho}$
$t_2 = \rho[\rho[]_{\varepsilon^1}, \rho[\rho[]_{\varepsilon^1}, \rho[]_{\varepsilon^1}]_{\rho[\rho[]_{\varepsilon^0}]_\rho}]_{\rho[\rho[]_{\varepsilon^0}]_\rho}$
$c_1 = f[g, b, f, \Box]_{t_1}$
$c_2 = f[g, b, \Box, b]_{t_1}$
$c_3 = f[\Box, b, f, b]_{t_1}$
$c_4 = f[g, a, f[g, b, f, \Box, b]_{t_2}, a]_{t_1}$

We have shown that the algorithm by Drewes and Högberg [1] can be used in an almost unchanged form to learn multi-dimensional trees in the new notation introduced in Subsection 3.2. This is tantamount to the claim that the algorithm is also able to learn even string languages that lie beyond the context-free class, provided that the learned multi-dimensional tree languages are enriched with the information that is needed in order to take the yields. Probably the easiest way to do this is to integrate the index sets directly into the alphabet (as has been done in Example 1), i.e., to multiply the symbols of the alphabet by the power set of $S^d = \{2, ..., n\}$ for $d \geq 2$. String languages situated beyond the regular class can then be learned in a two-step approach by first letting the algorithm learn a higher-dimensional representation of the language and then taking the string yields of the set that is recognized by the resulting automaton.

5 Conclusion

Generalizing the MAT learning algorithm L^* to regular tree languages of arbitrarily many dimensions is only one of the first steps to a more thorough understanding of the interaction between grammatical inference and formal language theory. The next steps would be to find L^*-like learning algorithms for finite-state recognizable languages of all kinds of objects, such as for example graphs or pictures, or take existing ones, such as the learning algorithm L^ω for ω-regular string languages [11], and try to integrate these often very similar looking algorithms into a single one that can process as many different types of inputs as possible. The same can then be attempted for other learning models and algorithms.

Ultimately, it is our goal to understand which general mathematical properties of formal language classes of all kinds of suitable objects underlie algorithmical learnability. Which conditions does a class of formally definable objects have to fulfil to be learnable under a certain model? Are they best captured in terms of universal algebra, or mathematical logics? At least for the class of regular languages, which up until now is the most explored formal language class in the area of grammatical inference, there is some evidence pointing to universal algebra as a convenient foundation, such as the Myhill-Nerode theorem (see [10], p. 34) or the fact that finite-state automata can be best defined for objects with term-like representations. Since we do not want to restrict ourselves to string or tree languages, this opens up another interesting question: What are the

exact properties (if they can be formulated at all) that characterize the term of "regularity" in general? And, for that matter, what about the language classes beyond that? For many kinds of formally definable objects the question whether the classical terms of the Chomsky Hierarchy can actually be applied in a reasonable manner is still to be settled.

To come back to the results of our paper: Possibly the finding that recognizable three-dimensional tree languages are learnable and the fact that these are connected to the linguistically inspired grammar formalism TAG can bring about more research and consequently more knowledge about natural language learning as well. The connection between formal learnability and human language acquisition is an ample field of speculations which are yet to be verified.

Another goal for the near future would be to try and implement the results of this paper. As an implementation for the algorithm by Drewes and Högberg [1] already exists, this should not be too hard to accomplish. Such a project might also be an impulse to reflect further on complexity issues, as Drewes and Högberg have already done in [1] and pursued in [12].

References

[1] F. Drewes and J. Högberg, "Learning a regular tree language from a teacher," in *Proc. Developments in Language Theory 2003* (Z. Ésik and Z. Fülöp, eds.), vol. 2710 of Lecture Notes in Computer Science, pp. 279–291, Springer, 2003.

[2] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, 1987.

[3] J. Rogers, "Syntactic structures as multi-dimensional trees," *Research on Language and Computation*, vol. 1, no. 3-4, pp. 265–305, 2003.

[4] J. Rogers, "wMSO theories as grammar formalisms," *Theor. Comp. Sci.*, vol. 293, no. 2, pp. 291–320, 2003.

[5] Y. Sakakibara, "Learning context-free grammars from structural data in polynomial time," *Theor. Comp. Sci.*, vol. 76, no. 2-3, pp. 223–242, 1990.

[6] A. K. Joshi, *Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural description*, pp. 206–250. Natural Language Processing, Cambridge University Press, 1985.

[7] D. J. Weir, "A geometric hierarchy beyond context-free languages," *Theor. Comp. Sci.*, vol. 104, no. 2, pp. 235–261, 1992.

[8] A. Kasprzik, "Making finite-state methods applicable to languages beyond context-freeness via multi-dimensional trees," Tech. Rep. 08-3, University of Trier, 2008.

[9] A. Kasprzik, "Two equivalent regularization methods for tree adjoining grammars," Tech. Rep. 08-1, University of Trier, 2008.

[10] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi, "Tree automata techniques and applications." www.grappa.univ-lille3.fr/tata, 2005.

[11] O. Maler and A. Pnueli, "On the learnability of infinitary regular sets," in *Proc. Fourth Annual Workshop on Computational Learning Theory*, pp. 128–136, Morgan Kaufmann, 1991.

[12] F. Drewes and J. Högberg, "Query learning of regular tree languages: How to avoid dead states," *Theory of Computing Systems*, vol. 40, no. 2, pp. 163–185, 2007.