Learning Tree Automata with Term Rewriting

Jakub Kopystiański and **Jan Otop**
University of Wrocław
{321897, jan.otop}@uwr.edu.pl

Abstract

We present an extension of the Angluin-style learning algorithm for tree automata that incorporates deductive inference. The learning algorithm is provided with a term rewriting system that specifies properties of the target tree language (e.g., the order of subtrees under a symbol f is irrelevant). This term rewriting system is used to infer answers to some queries, which reduces the query complexity of the learning algorithm. We present examples of rewrite systems that express natural properties of tree-structured data, which yield a significant reduction in the number of queries.

1 Introduction

Active automata learning. The aim of *active automata learning* is to generate a *deterministic finite automaton (DFA)* recognizing an unknown regular language $\mathcal{L}$ (target language) having access to an oracle answering queries about $\mathcal{L}$, i.e., the algorithm is allowed to *actively* query about the language in contrast to *passive* learning where the input dataset does not change. There are two types of queries: *membership* of a given word in $\mathcal{L}$, and *equivalence* of the language of a given DFA $\mathcal{A}$ with the target language $\mathcal{L}$, where the answer to an equivalence query is either YES, or a *counterexample* word distinguishing $\mathcal{L}$ and $\mathcal{L}(\mathcal{A})$. This framework was proposed in the seminal paper [Angluin, 1987] along with the L*algorithm, which enables learning in polynomial time. Active automata learning has been intensively studied for both efficiency [Isberner *et al.*, 2014; Vaandrager *et al.*, 2022] as well as extensions to other automata-based models. In particular, the L*algorithm has been adapted to tree automata [Drewes and Högberg, 2003] and multiplicity tree automata [Habrard and Oncina, 2006; Marusic and Worrell, 2015].

Applications of active tree-automata learning. Trees are a natural model for structured data and hence tree automata are often used in knowledge representation, including processing of XML, JSON, and unification in Description Logics [Baader and Narendran, 1998]. Motivated by these applications, active learning of tree languages and related formalisms has been extensively studied and applied to information extraction [Kosala *et al.*, 2003], the synthesis of node specifications [Grienenberger and Ritzert, 2019], structured data transformations [Yaghmazadeh *et al.*, 2016], and Description Logic $\mathcal{EL}$ terminologies [Magnini *et al.*, 2025]. Due to the close link between regular tree and context-free languages, active learning of tree languages has been applied to context-free grammar repair [Raselimo and Fischer, 2021], learning probabilistic grammars [Nitay *et al.*, 2021], and verification and explainability of RNNs [Barbot *et al.*, 2021].

Query complexity. The number of queries required by the L*algorithm is polynomial in the size of the target automaton. While it is feasible to answer that many queries automatically, the number remains too high for manual intervention. Consequently, there is a large body of work on improving query complexity of active automata learning [Isberner *et al.*, 2014; Frohme, 2019; Vaandrager *et al.*, 2022; Kruger *et al.*, 2024; Dierl *et al.*, 2024] as well as implementing the teacher using Large Language Models [Magnini *et al.*, 2025; Bhattamishra *et al.*, 2026; Vazquez-Chanlatte *et al.*, 2025]. However, LLMs only answer membership queries, and their equivalence query answers are merely approximations with a large number of membership queries. This approximation approach is similar to prior work on automating query answering [Vaandrager, 2017]. Therefore, in this work, we follow an alternative route, proposed in [Fica and Otop, 2025], which is to restrict the search space of automata by providing the learning algorithm with additional information (called advice) regarding the target language. Furthermore, we focus on inferring the answers to equivalence queries as it yields the greatest reduction in complexity and increases reliability of learning (answers to these queries are often approximated). We briefly discuss how to infer answers to membership queries.

Learning with advice. We consider an extended learning framework, in which the learning algorithm is given *advice* about the target language [Fica and Otop, 2025]. This advice constrains the search space and allows the algorithm to infer answers to certain queries. This approach bridges two synthesis paradigms: deductive synthesis from specifications and inductive synthesis based on queries. Furthermore, it allows for flexibility to express the learned language partially with queries and partially with a *term rewriting system (TRS)*, which substantially differs from automata, and hence some properties can be succinctly expressed with a TRS instead of multiple queries.

Term rewriting as advice. A Term Rewriting System (TRS) consists of rewrite rules $l \to r$, where l, r are terms (or equivalently trees with variables). *Rewriting* is the process of iteratively transforming a tree by replacing an instance of a left-hand side l with its corresponding right-hand side r. For example:

- Commutativity: The rule $f(X,Y) \to f(Y,X)$ allows for the swapping of subtrees at node f.
- Associativity: With $f(f(X,Y),Z) \to f(X,f(Y,Z))$ any tree with f symbols and constants can be transformed into a right skewed tree.

In our approach, an advice TRS defines an equivalence relation relative to the target language $\mathcal{L}$. Specifically, if a tree t is in $\mathcal{L}$, then all trees obtained by rewriting t must be in $\mathcal{L}$. Similarly rewriting is compatible with the complement of $\mathcal{L}$. We also consider one-sided relations stating that (1) trees from $\mathcal{L}$ are rewritten to trees from $\mathcal{L}$ only, while there are no restrictions for trees not in $\mathcal{L}$, or (2) trees not in $\mathcal{L}$ are rewritten to trees not in $\mathcal{L}$ only. This allows the TRS to capture high-level structural properties, such as symmetry or associativity of a given symbol, while it is not expected to characterize the learned language completely.

Contributions. Active automata learning with advice has been already introduced for finite automata on words [Fica and Otop, 2025]. In this paper we study automata over trees and present the following substantial contributions:

1. We develop an algorithm that infers answers to equivalence queries for tree automata (Section 3). The transition to trees introduces distinct technical challenges absent in the word case. In particular, general inference is computationally difficult. We therefore identify subclasses of TRSs that admit tractable inference.
2. To illustrate the efficacy of the advice mechanism, we provide TRSs expressing natural tree language properties and show, through experimental evaluation, a significant reduction in query complexity (Section 4).
3. Finally, we introduce the synthesis problem for TRS for a given regular tree language, which is the converse of active learning with advice: given a regular tree language, find a TRS consistent with it. We establish a connection between synthesis of TRS and the classical problem of finding synchronizing words in DFA (Section 5).

The extended version of the paper is available at [Kopystiański and Otop, 2026a], while the code and experiments data are available at [Kopystiański and Otop, 2026b].

Related work. A large body of work focuses on optimizing the computational and query complexity of the L*algorithm for word automata [Isberner *et al.*, 2014; Frohme, 2019; Vaandrager *et al.*, 2022; Dierl *et al.*, 2024] and tree automata [Drewes and Högberg, 2007; Kasprzik, 2013]. These works are orthogonal to our approach, as our algorithm is compatible with any active learning algorithm using equivalence queries. Our work directly extends [Fica and Otop, 2025] to the tree setting. Independently, enhancing the performance of active tree automaton learning by restricting focus to specific subclasses of tree languages remains an active area of research [van Heerdt *et al.*, 2021; Björklund *et al.*, 2017]. Furthermore, extending the types of queries to improve the efficiency of active learning has also been investigated [Tirnăucă and Tirnăucă, 2007].

2 Preliminaries

A **signature** (or ranked alphabet) $\mathcal{F}$ is a pair (F, Ar) consisting of the set of symbols F and a function $\mathsf{Ar}\colon F \to N$ assigning a unique **arity** to each symbol. To ease the notation, we will write $f \in \mathcal{F}$ meaning $f \in F$.

2.1 Terms

For a signature $\mathcal{F}$ and a set of **variables** $\mathcal{X}$, the set of **terms** over $\mathcal{F}$ and $\mathcal{X}$, denoted by $\mathcal{T}(\mathcal{F},\mathcal{X})$, is the least set containing $\mathcal{X}$ such that for all $f \in \mathcal{F}$, if $k = \mathsf{Ar}(f)$ and $t_1,\dots,t_k \in \mathcal{T}(\mathcal{F},\mathcal{X})$, then $f(t_1,\dots,t_k) \in \mathcal{T}(\mathcal{F},\mathcal{X})$. In particular, if $a \in \mathcal{F}$ and $\mathsf{Ar}(a) = 0$, then $a \in \mathcal{T}(\mathcal{F},\mathcal{X})$. The set of **ground terms** $\mathcal{T}(\mathcal{F})$ over the signature $\mathcal{F}$ is the set $\mathcal{T}(\mathcal{F},\emptyset)$, i.e., terms containing no variables. We consider signatures $\mathcal{F}$ that contain at least one constant symbol, as otherwise $\mathcal{T}(\mathcal{F})$ is empty. A term t is **linear** if every variable occurs at most once in t.

A **substitution** σ is a function from variables to terms with a finite domain. Each substitution σ can be uniquely extended to $\widehat{\sigma}$ over all terms $\mathcal{T}(\mathcal{F},\mathcal{X})$ as follows: for $X \in \mathcal{X}$ we have $\widehat{\sigma}(X) = \sigma(X)$ if $X \in \mathrm{dom}(\sigma)$ and otherwise $\widehat{\sigma}(X) = X$. For a term t, which is not a variable, we have $t = f(t_1,\dots,t_k)$, and $\widehat{\sigma}(t) = f(\widehat{\sigma}(t_1),\dots,\widehat{\sigma}(t_k))$. Since the extension from σ to $\widehat{\sigma}$ is unique, we refer to $\widehat{\sigma}$ as σ. For a variable X and terms t, s, we denote by $t[X \to s]$ the term $\sigma_s(t)$ resulting from the substitution $\sigma_s(X) = s$.

2.2 Trees as Terms

A (ranked) tree over $\mathcal{F}$ is a ground term over $\mathcal{F}$ and we refer to $\mathcal{T}(\mathcal{F})$ as the set of trees over $\mathcal{F}$. Referring to ranked trees as ground terms is common in the literature [Comon *et al.*, 2008]. In the unranked case, when symbols do not have unique arities, the order-based definition of trees is required, but in this paper we consider only ranked trees.

A **context** c is a term with a single occurrence of a variable X. For a context c and a tree (ground term) t, we define $c(t) = c[X \to t]$, i.e., the substitution $\sigma(X) = t$ applied to c.

2.3 Tree Automata

A bottom-up deterministic finite tree automaton (DFTA) is a tuple $(\mathcal{F}, Q, Q_{\text{acc}}, \delta)$ consisting of the signature $\mathcal{F}$, a finite set of states Q, a set of accepting states Q_{acc}, and a transition function $\delta\colon Q^* \times \mathcal{F} \to Q$. The transition function δ is a partial function such that $\delta(u,f)$ is defined if and only if $|u|$ is equal to $\mathsf{Ar}(f)$. We extend δ to a function $\widehat{\delta}\colon \mathcal{T}(\mathcal{F}) \to Q$ inductively:

1. If t is a constant $c \in \mathcal{F}$, then $\widehat{\delta}(c)$ is the unique state q such that $\delta(c) = q$.
2. If $t = f(t_1,\dots,t_k)$, then $\widehat{\delta}(t) = \delta(\widehat{\delta}(t_1),\dots,\widehat{\delta}(t_k), f)$.

The tree language **recognized by** $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set of all trees such that $\widehat{\delta}(t)$ is an accepting state, i.e., $\mathcal{L}(\mathcal{A}) = \{t \mid \widehat{\delta}(t) \in Q_{\text{acc}}\}$.

2.4 Myhill-Nerode Theorem for Tree Languages

For a tree language $\mathcal{L}$ over $\mathcal{F}$ we define $\sim_{\mathcal{L}}$ on trees over $\mathcal{F}$ as follows: for all trees t_1, t_2 over $\mathcal{F}$ we have $t_1 \sim_{\mathcal{L}} t_2$ if and only if $c(t_1) \in \mathcal{L} \leftrightarrow c(t_2) \in \mathcal{L}$, for every context c over $\mathcal{F}$.

Lemma 1. *A tree language $\mathcal{L}$ is regular if and only if $\sim_{\mathcal{L}}$ has finitely many equivalence classes.*

Furthermore, for every regular tree language $\mathcal{L}$ a **minimal** DFTA recognizing $\mathcal{L}$ exists and it is unique (up to an isomorphism). The states of the minimal DFTA for $\mathcal{L}$ correspond to equivalence classes of $\sim_{\mathcal{L}}$. Note that all states in the minimal DFTA are reachable.

2.5 Learning Tree Automata

The framework of active tree-automata learning assumes an oracle, called *minimally adequate teacher*, which answers two types of queries about the target tree language $\mathcal{L}$:

- **membership queries**: given a tree $t \in \mathcal{T}(\mathcal{F})$, is $t \in \mathcal{L}$?, and
- **equivalence queries**: given a DFTA $\mathcal{A}$, is $\mathcal{L}(\mathcal{A}) = \mathcal{L}$? If not the teacher returns a counterexample, which is a tree from exactly one of the sets $\mathcal{L}(\mathcal{A})$ and $\mathcal{L}$.

The tree-automata variant of the L*algorithm [Drewes and Högberg, 2003] having access to the oracle for a tree language $\mathcal{L}$ returns the minimal DFTA $\mathcal{A}_{\mathcal{L}}$ recognizing $\mathcal{L}$. It works in polynomial time in $|\mathcal{A}_{\mathcal{L}}|$ and the total size of counterexamples supplied by the oracle. Unlike the word-automaton case, where the shortest counterexample is linear in the size of the minimal DFA, a minimal-size counterexample tree can be of exponential size. However, if counterexamples are presented as directed acyclic graphs (DAGs), their size remains polynomially bounded in $|\mathcal{A}_{\mathcal{L}}|$ [Charatonik, 1999; Anantharaman *et al.*, 2005].

2.6 Term Rewriting Systems

A **term rewriting system (TRS)** $\mathcal{R}$ over a signature $\mathcal{F}$ is a finite set of pairs of terms (l, r) over $\mathcal{F}$. A pair of terms (l, r) from $\mathcal{R}$ is called a **rewrite rule** and denoted by $l \to r$. For a TRS $\mathcal{R}$, we define a *single-step rewrite relation* $\to_{\mathcal{R}}$ over terms from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ as the least relation such that (i) for all substitutions σ and all rules (l, r) from $\mathcal{R}$ we have $\sigma(l) \to_{\mathcal{R}} \sigma(r)$, and (ii) for all $f \in \mathcal{F}$, if k is the arity of f and $s \to_{\mathcal{R}} t$, then for all terms $s_1, \ldots, s_{k-1}$ and all i we have $f(s_1, \ldots, s_{i-1}, s, s_{i+1}, \ldots, s_k) \to_{\mathcal{R}} f(s_1, \ldots, s_{i-1}, t, s_{i+1}, \ldots, s_k)$. The rewriting relation $\to^*_{\mathcal{R}}$ is the transitive and reflexive closure of $\to_{\mathcal{R}}$. We will omit the $\mathcal{R}$ subscript if the TRS is clear from the context.

Normal forms. A term t is in a **normal form** if there is no t' such that $t \to_{\mathcal{R}} t'$. If for a term s there is a unique t in a normal form such that $s \to^*_{\mathcal{R}} t$, we say that t is the normal form of s (w.r.t. $\mathcal{R}$) and denote it by $\mathbf{NF}_{\mathcal{R}}(s)$.

Computing normal forms. A (finite) TRS $\mathcal{R}$ is *terminating* if every sequence of terms $s_0, s_1, \ldots$ such that $s_i \to s_{i+1}$ is finite, *confluent* if for all terms s, s_1, s_2, if $s \to^* s_1$ and $s \to^* s_2$, then there is t such that $s_1 \to^* t$ and $s_2 \to^* t$, and *convergent* if it is terminating and confluent. In a convergent TRS, every term s has the (unique) normal form, which can be computed by applying reductions in an arbitrary order as long as the term can be reduced. Termination guarantees that this process takes finitely many steps and confluence entails the uniqueness of the result. Therefore, for a convergent TRS $\mathcal{R}$, one can effectively compute $\mathbf{NF}_{\mathcal{R}}(s)$. However, showing that a TRS is convergent is generally difficult [Baader and Nipkow, 1998].

3 Learning Tree Automata With Advice

In this section, we adapt the framework of active learning with advice [Fica and Otop, 2025] to tree automata. We first formally define the active learning problem with advice, then discuss how to resolve queries using that advice.

In our framework, advice for the learning algorithm is given via TRSs, which relate to tree languages through the following notions of consistency:

Definition 2 ([Fica and Otop, 2025])**.** *Let $\mathcal{F}$ be a signature, $\mathcal{R}$ be a TRS over $\mathcal{F}$ and $\mathcal{L}$ be a (regular) tree language over $\mathcal{F}$. We say that*

- *$\mathcal{R}$ is* (fully) consistent *with $\mathcal{L}$ if and only if for all trees s, t, if $s \to^*_{\mathcal{R}} t$, then $s \in \mathcal{L} \iff t \in \mathcal{L}$.*
- *$\mathcal{R}$ is* positively consistent *with $\mathcal{L}$ if and only if for all trees s, t, if $s \to^*_{\mathcal{R}} t$, then $s \in \mathcal{L} \implies t \in \mathcal{L}$.*
- *$\mathcal{R}$ is* negatively consistent *with $\mathcal{L}$ if and only if for all trees s, t, if $s \to^*_{\mathcal{R}} t$, then $s \notin \mathcal{L} \implies t \notin \mathcal{L}$.*

Note that a TRS $\mathcal{R}$ is fully consistent with $\mathcal{L}$ if and only if it is both positively and negatively consistent. While more granular than standard consistency, positive and negative consistency are harder to infer.

Having the notions of consistency, we can formally state the active learning with advice problem.

Definition 3. *The active tree-automata learning with advice problem for an unknown regular tree language $\mathcal{U}$ is as follows:*

- ***Input**: (1) an oracle answering membership and equivalence queries about the target regular tree language $\mathcal{U}$, and (2) three advice TRSs: $\mathcal{R}_=$, which is consistent with $\mathcal{U}$, and $\mathcal{R}_+, \mathcal{R}_-$, which are respectively positively and negatively consistent with $\mathcal{U}$.*
- ***Output:** the minimal DFTA that recognizes $\mathcal{U}$.*

Reducing equivalence queries with advice. To reduce the number of equivalence queries, the algorithm first checks whether an advice TRS $\mathcal{R}$ is consistent (resp., positively or negatively consistent) with the language of a candidate automaton $\mathcal{A}$ before querying the oracle. If it is not consistent, an equivalence query is unnecessary because $\mathcal{L}(\mathcal{A})$ must differ from the target language with which $\mathcal{R}$ is known to be consistent (resp., positively or negatively consistent). In such cases, the algorithm computes a pair of trees violating consistency (resp., positive or negative consistency). For example, for positive consistency, violating trees s, t are such that $s \to^*_{\mathcal{R}} t$ but $s \in \mathcal{L}(\mathcal{A}) \implies t \in \mathcal{L}(\mathcal{A})$ does not hold. One of the trees s or t has to be the counterexample to $\mathcal{L}(\mathcal{A})$ being equal to the target language. The algorithm decides which one with a single membership query.

Remark 4 (Rewrite rules vs. regular languages)**.** *Rewrite rules can express non-regular properties. For example, $f(X, X) \to a$ expresses that the value of a tree with identical immediate subtrees the same as the value of a with respect to $\mathcal{L}$. Tree automata cannot express equality of subtrees. Similarly, while associativity $f(X, f(Y, Z)) \to f(f(X, Y), Z)$ does not compare subtrees, applying it iteratively can reshape trees, destroying information encoded in the tree structure. For instance, consider a regular tree language over constants a, b and binary f such that the string of leaf labels forms a palindrome and every palindrome appears in some tree. This language is defined to contain constants a, b and be closed under the contexts $f(a, f(X, a))$ and $f(b, f(X, b))$. A DFTA can detect whether a tree is built from these two contexts and constants. However, if such tree is reshaped to a skewed tree, no DFTA can recognize whether the leaves form a palindrome. Consequently, this language cannot be extended to satisfy associativity while retaining its key properties. Despite this, we only ask whether a language* known to be regular *satisfies rewrite rules.*

3.1 Deciding Consistency

We discuss how to efficiently decide consistency. First, we establish a condition that reduces checking consistency of a TRS with a tree language to checking combinatorial properties of the minimal DFTA recognizing that language (Lemma 6). Next, we discuss the complexity of deciding this combinatorial condition.

To conveniently define the combinatorial condition, we extend the function $\widehat{\delta}$ from trees (i.e., ground terms) to all terms, where terms with variables define *state transformations*:

Definition 5. *Let $\mathcal{A}$ be a minimal DFTA and t be a term over variables $X_1, \ldots, X_k$. We define $\vec{\delta}_{\mathcal{A}}(t)$ as a function from Q^k to Q such that $\vec{\delta}_{\mathcal{A}}(t)(q_1, \ldots, q_k)$ is the state assigned to t if in leaves labelled by X_i the DFTA $\mathcal{A}$ starts in state q_i. Formally, let $s_1, \ldots, s_k$ be trees over $\mathcal{F}$ such that $\widehat{\delta}(s_i) = q_i$. Then, $\vec{\delta}_{\mathcal{A}}(t)(q_1, \ldots, q_k) = \widehat{\delta}(t[X_1 \to s_1, \ldots, X_k \to s_k])$.*

Note that in a minimal DFTA every state is reachable, and hence trees $s_1, \ldots, s_k$ as in the above definition exist.

Lemma 6. *Let $\mathcal{R}$ be a TRS over $\mathcal{F}$ and $\mathcal{A}$ be a minimal DFTA over $\mathcal{F}$. The TRS $\mathcal{R}$ is* consistent *with $\mathcal{L}(\mathcal{A})$ if and only if the functions $\vec{\delta}_{\mathcal{A}}(l)$ and $\vec{\delta}_{\mathcal{A}}(r)$ coincide, for each rule $l \to r \in \mathcal{R}$.*

It is essential that the TRS and the DFTA are over the same signature $\mathcal{F}$, as otherwise checking consistency is undecidable [Fica and Otop, 2025].

Remark 7. *Consider a rule $f(X, X) \to a$, for which the left-hand side defines a non-regular language. Observe that Lemma 6 implies that if $f(X, X) \to a$ is consistent with the language of a minimal DFTA $\mathcal{A}$, then for all trees s_1, s_2 labelled with the same state by $\mathcal{A}$ ($\widehat{\delta}(s_1) = \widehat{\delta}(s_2)$), the tree $f(s_1, s_2)$ is labeled with the state $\widehat{\delta}(a)$, which is a substantially stronger condition than just $f(X, X) \to a$.*

Now, in the remaining part of this section, we discuss the complexity of evaluating the condition from Lemma 6.

For any tuple of states $\vec{q} \in Q^k$, the value $\vec{\delta}_{\mathcal{A}}(l)(\vec{q})$ (resp., $\vec{\delta}_{\mathcal{A}}(r)(\vec{q})$) can be computed in polynomial time in $|l|$ (resp., $|r|$). However, the number of arguments is equal to the number of variables in terms l, r and hence there may be exponentially many tuples of states to check. Still, Lemma 6 implies that consistency can be checked in coNP:

Proposition 8. *Checking consistency of a TRS with a tree language given by a DFTA is in coNP. For every $N > 0$, consistency is decidable in polynomial time over TRSs with at most N variables, i.e., in every rule $l \to r$, both terms l, r are over $\{X_1, \ldots, X_N\}$.*

In particular, the consistency problem is decidable in polynomial time if the TRS is fixed and only the DFTA is the input.

Computing counterexamples. Given a rule $l \to r \in \mathcal{R}$ such that $\vec{\delta}_{\mathcal{A}}(l) \neq \vec{\delta}_{\mathcal{A}}(r)$ and $q_1, \ldots, q_k$ witnessing this, i.e., $\vec{\delta}_{\mathcal{A}}(l)(q_1, \ldots, q_k) \neq \vec{\delta}_{\mathcal{A}}(r)(q_1, \ldots, q_k)$, we can compute two trees witnessing consistency being violated. Let $s_1, \ldots, s_k$ be trees such that $\widehat{\delta}(s_i) = q_i$. Consider trees:

- $\hat{l} = l[X_1 \to s_1, \ldots, X_k \to s_k]$, and
- $\hat{r} = r[X_1 \to s_1, \ldots, X_k \to s_k]$.

Let $q_l = \widehat{\delta}(\hat{l})$ and $q_r = \widehat{\delta}(\hat{r})$. Then, $q_l \leq q_r$. Since $\mathcal{A}$ is minimal, there is a context c distinguishing states q_l and q_r. Thus, for $s = c(\hat{l})$ and $t = c(\hat{r})$, exactly one of s, t belongs to $\mathcal{L}(\mathcal{A})$. In summary, while $s \to^* t$ in one step, $s \in \mathcal{L}(\mathcal{A}) \leftrightarrow t \in \mathcal{L}(\mathcal{A})$ does not hold. Since we know that $s \in \mathcal{U} \leftrightarrow t \in \mathcal{U}$, one of s, t is a counterexample.

In general, the coNP upper bound cannot be improved:

Lemma 9. *Checking consistency of a rule $l \to r$, where l is a (non-linear) term and r is a ground term with the language of a given DFTA is coNP-hard.*

Proof sketch. The proof is by reduction from the tautology problem. The signature consists of binary $\wedge, \vee$, unary $\neg$ and constants $\top, \bot$. The rule l encodes an input formula φ, while r is $\top$. Finally, the DFTA views input trees as propositional formulas and evaluates their truth values. Thus, $l \to \top$ is consistent exactly when φ is a tautology. □

The above lemma holds even for a fixed DFTA. Still, the hardness argument relies on variables occurring multiple times. Therefore, a natural question is what is the complexity of checking consistency of rules $l \to r$, in which both terms are linear. We leave this as an *open problem* and propose a simple heuristic.

Heuristics for linear rules. We propose a heuristic for linear terms based on the **state-counting function**, which counts the cardinality of the preimage $\vec{\delta}_{\mathcal{A}}(l)$ and $\vec{\delta}_{\mathcal{A}}(r)$ with multiplicities. This heuristic does not confirm consistency, but it can identify some rules that violate consistency. This is not a problem as consistency itself overapproximates equivalence. Consequently, any heuristic that overapproximates consistency also overapproximates equivalence, i.e., the heuristic does not err on the negative answers.

Definition 10. *For a term t and a DFTA $\mathcal{A}$, we define the state-counting function $\#_{\mathcal{A}}[t] \colon Q \to N$ as the cardinality of the preimage $\vec{\delta}_{\mathcal{A}}^{-1}[\{q\}]$, i.e., $\#_{\mathcal{A}}[t](q) = |\{(q_1, \ldots, q_k) \mid \vec{\delta}_{\mathcal{A}}(t)(q_1, \ldots, q_k) = q\}|$.*

Observe that if $\vec{\delta}_{\mathcal{A}}(l) = \vec{\delta}_{\mathcal{A}}(r)$, then $\#_{\mathcal{A}}[l] = \#_{\mathcal{A}}[r]$. Consequently, checking $\#_{\mathcal{A}}[l] = \#_{\mathcal{A}}[r]$ serves as a necessary (but not sufficient) condition for consistency of $l \to r$ with $\mathcal{L}(\mathcal{A})$ (i.e., it is an underapproximation). This provides a computationally efficient heuristic to filter out inconsistent rules before performing more expensive checks. Furthermore, $\#_{\mathcal{A}}[t]$ can be efficiently computed for linear terms t, and hence it can be used as a preliminary test even if consistency could be decidable in polynomial but superlinear time.

Lemma 11. *The function $\#_{\mathcal{A}}[t]$ can be computed in time $O(|\mathcal{A}| \cdot |t|)$ over linear terms t.*

Observe that if one of the functions $\#_{\mathcal{A}}[l]$ or $\#_{\mathcal{A}}[r]$ is constant, then $\#_{\mathcal{A}}[l] = \#_{\mathcal{A}}[r]$ implies $\vec{\delta}_{\mathcal{A}}(l) = \vec{\delta}_{\mathcal{A}}(r)$. In particular, for rules $l \to r$ such that l is a linear term and r is a ground term, we can decide consistency in linear time.

Corollary 12. *The consistency of a rule $l \to r$ with $\mathcal{L}(\mathcal{A})$ can be performed in time $O((|l| + |r|) \cdot |\mathcal{A}|)$ for rules $l \to r$ such that l is a linear term and r is a ground term.*

3.2 Deciding Positive and Negative Consistency

The notions of positive and negative consistency are dual. Observe that $\mathcal{R}$ is positively consistent with $\mathcal{L}$ if and only if $\mathcal{R}$ is negatively consistent with the complement of $\mathcal{L}$. Therefore, we will focus on positive consistency as the results carry over to negative consistency straightforwardly (note that DFTA can be complemented via swapping accepting and non-accepting states).

First, we present a counterpart of Lemma 6 for positive consistency. For that, we need to define two order relations: one on states and another on states transformations that rank states from the least accepting to the most accepting.

Definition 13. *Let $\mathcal{A}$ be a DFTA. We define an order relation $\leq_{\mathcal{A}}$ on states of $\mathcal{A}$ such that $q_1 \leq_{\mathcal{A}} q_2$ if for all trees t_1, t_2 such that $\widehat{\delta}(t_1) = q_1$ and $\widehat{\delta}(t_2) = q_2$, for every context c we have $c(t_1) \in \mathcal{L}(\mathcal{A}) \implies c(t_2) \in \mathcal{L}(\mathcal{A})$.*

Given a DFTA $\mathcal{A}$, the relation $\leq_{\mathcal{A}}$ can be computed in $O(|\mathcal{A}|^3)$ by the greatest fixed-point iteration. The algorithm starts with the relation $R = Q \times Q$. It removes pairs (q_1, q_2) such that $q_1 \in Q_{\text{acc}}$ and $q_2 \notin Q_{\text{acc}}$. Then, iteratively, for every context c of height 1 (which corresponds to a transition), if $(c(q_1), c(q_2)) \notin R$, the pair (q_1, q_2) is removed from R. The greatest fixed-point is reached after $|Q|^2$ iterations, while each iteration can be performed in $O(|\mathcal{A}|)$ by checking all contexts of height 1.

We extend $\leq_{\mathcal{A}}$ from states to state transformations:

Definition 14. *Let $\mathcal{A}$ be a DFTA and $k > 0$. We define an order relation $\preccurlyeq_{\mathcal{A}}$ on functions from Q^k to Q as follows. For all $h_1, h_2 \colon Q^k \to Q$ we have $h_1 \preccurlyeq_{\mathcal{A}} h_2$ if and only if $h_1(\vec{q}) \leq_{\mathcal{A}} h_2(\vec{q})$, for all $\vec{q} \in Q^k$.*

Finally, we present the counterpart of Lemma 6 for positive consistency:

Lemma 15. *Let $\mathcal{R}$ be a TRS over $\mathcal{F}$ and $\mathcal{A}$ be a minimal DFTA over $\mathcal{F}$. The TRS $\mathcal{R}$ is* positively consistent *with $\mathcal{L}(\mathcal{A})$ if and only if $\vec{\delta}_{\mathcal{A}}(l) \preccurlyeq_{\mathcal{A}} \vec{\delta}_{\mathcal{A}}(r)$, for each rule $l \to r \in \mathcal{R}$.*

Since $\leq_{\mathcal{A}}$ can be computed in polynomial time, the condition $\vec{\delta}_{\mathcal{A}}(l) \preccurlyeq_{\mathcal{A}} \vec{\delta}_{\mathcal{A}}(r)$ can be falsified by finding an appropriate $\vec{q}$ and computing $q_1 = \vec{\delta}_{\mathcal{A}}(l)(\vec{q}), q_2 = \vec{\delta}_{\mathcal{A}}(r)(\vec{q})$ and finally checking that $q_1 \leq_{\mathcal{A}} q_2$ does not hold. In consequence, checking positive consistency is in coNP.

Observe that for rules $l \to r$ with a ground term $r \in \mathcal{L}$, positive consistency and consistency coincide, i.e., $l \to r$ is consistent with $\mathcal{L}$ if and only if it is positively consistent. If $r \notin \mathcal{L}$, we can instead take the complement of $\mathcal{L}$ and reduce to the previous case. Therefore, for such rules the complexity of checking positive consistency is the same as checking (full) consistency: if r is a ground term, checking consistency of $l \to r$ with a regular tree language $\mathcal{L}$ is coNP-complete in general, and it is decidable in polynomial time for rules $l \to r$ with l being a linear term. When l, r are both linear terms, the complexity of checking consistency of $l \to r$ is left open. For positive consistency, we can show that it is coNP-complete. It suffices to show hardness:

Lemma 16. *Checking positive consistency of a rule $l \to r$ with the language of a given DFTA is coNP-hard for rules $l \to r$ such that l, r are linear terms.*

Still, we can utilize the heuristic based on $\#_{\mathcal{A}}[t]$, which was proposed above for consistency. We need to adapt the condition to positive consistency in the following way. We define the cumulative state-counting function $\Sigma\#_{\mathcal{A}}[t]$ as $\Sigma\#_{\mathcal{A}}[t](q) = \sum_{q \leq_{\mathcal{A}} q'} \#_{\mathcal{A}}[t](q')$, i.e., it sums the multiplicity of each state q' that is greater in the $\leq_{\mathcal{A}}$ order.

Lemma 17. *For terms l, r, if $\vec{\delta}_{\mathcal{A}}(l) \preccurlyeq_{\mathcal{A}} \vec{\delta}_{\mathcal{A}}(r)$, then $\#_{\mathcal{A}}[l](q) \leq \Sigma\#_{\mathcal{A}}[r](q)$, for every state q.*

Note that having the function $\#_{\mathcal{A}}[r](q_2)$ and the relation $\leq_{\mathcal{A}}$, we can compute $\Sigma\#_{\mathcal{A}}[t](q)$ in time $O(|Q|^2)$ from definition or in $O(|<_{\mathcal{A}}| \cdot |Q|)$, where $<_{\mathcal{A}}$ is the strict variant of the order $\leq_{\mathcal{A}}$ and $|<_{\mathcal{A}}|$ is the number of tuples in $<_{\mathcal{A}}$. The latter complexity is achieved by computing the *transitive reduction* $\mathsf{succ}_{\mathcal{A}}$ [Aho *et al.*, 1972] of $<_{\mathcal{A}}$ in $O(|<_{\mathcal{A}}| \cdot |Q|)$, i.e., the transitive closure of $\mathsf{succ}_{\mathcal{A}}$ is $<_{\mathcal{A}}$. Then the graph $(Q, \mathsf{succ}_{\mathcal{A}})$ is sorted topologically in $O(|\mathsf{succ}_{\mathcal{A}}| \cdot |Q|)$ and the summation is performed iteratively over the immediate successors only in $O(|\mathsf{succ}_{\mathcal{A}}| \cdot |Q|)$. Since $\mathsf{succ}_{\mathcal{A}} \subseteq <_{\mathcal{A}}$, the whole computation is in $O(|<_{\mathcal{A}}| \cdot |Q|)$.

3.3 Membership Queries

Typical implementations of the L*algorithm store answers to membership queries in some data structure (e.g. an observation tree) to ensure that each query is unique. We extend this idea to membership queries modulo term rewriting.

For the advice TRS $\mathcal{R}_=$ (which is fully consistent with the target language), we select a convergent subset $\mathcal{R}'_=$. Since the TRS $\mathcal{R}'_=$ is convergent, every term has a unique normal form.

The algorithm maintains a cache, which is a dictionary mapping each tree's normal form $\mathbf{NF}_{\mathcal{R}'_=}(t)$ to its membership result. Before asking a query for a tree s, the algorithm computes the normal form of s and checks the cache. If present, it fetches the stored answer without querying the oracle. Consistency of $\mathcal{R}$ with the target language $\mathcal{L}$ implies that if $\mathbf{NF}_{\mathcal{R}'_=}(s) = \mathbf{NF}_{\mathcal{R}'_=}(t)$, then $s \in \mathcal{L} \leftrightarrow t \in \mathcal{L}$. Otherwise, if

$\mathbf{NF}_{\mathcal{R}'_=}(s)$ is not in the cache, it asks the oracle the membership query and stores $\mathbf{NF}_{\mathcal{R}'_=}(s)$ along with the answer in the cache.

The TRSs $\mathcal{R}_+, \mathcal{R}_-$ can be used to infer answers to membership queries as well, but the complexity of inference makes this infeasible [Fica and Otop, 2025].

4 Examples of Advice Rewrite Rules

We present examples of TRS that express interesting properties of tree languages. While these properties originate in algebra, we discuss their relevance to knowledge representation, structured data, and Description Logic (DL) [Nardi *et al.*, 2003].

4.1 Families of TRS

We consider the following types of TRSs: *associativity*, *commutativity*, variants of *distributivity*, and *cancellation* rules. The applications of these rules are discussed in the context of full consistency. We elaborate on these TRSs below.

Associativity. While we have considered ranked trees, unranked trees with symbols of variable arities are prevalent in structured data. For example, the HTML `<body>` tag has an arbitrary number of immediate successors, and dictionaries in JSON have arbitrarily many entries. Unranked symbols can be encoded with associative binary symbols; the associativity rule for f: $f(X, f(Y, Z)) \to f(f(X, Y), Z)$ implies that the shape of a tree labelled with f is irrelevant and hence all f symbols can be contracted into a single high-arity symbol. Furthermore, the basic DL operators $\sqcap$ and $\sqcup$ are associative.

Commutativity. Commutativity of f is expressed with the rule $f(X, Y) \to f(Y, X)$. This can be more precisely described as *horizontal commutativity*, indicating that sibling subtrees can be swapped without changing the tree's value. JSON dictionaries and DL operators $\sqcap$ and $\sqcup$ are (horizontally) commutative. Furthermore, unranked and unordered trees [Boneva and Talbot, 2005] can be modeled with binary operators that are associative and (horizontally) commutative.

Distributivity. The standard distributive properties for binary symbols f, g are: (1) f is *left-distributive* over g: $f(X, g(Y, Z)) \to g(f(X, Y), f(X, Z))$, and (2) f is *right-distributive* over g: $f(g(X, Y), Z) \to g(f(X, Z), f(Y, Z))$. DL operators $\sqcap$ and $\sqcup$ are left- and right-distributive one over another, i.e., both $\sqcap$ over $\sqcup$ and $\sqcup$ over $\sqcap$.

Variants of distributivity. Distributivity can be generalized beyond binary symbols. For a binary symbol f and an unary symbol g the rule $g(f(X, Y)) \to f(g(X), g(Y))$ is a variant of distributivity. Similarly, for unary f, g the rule $g(f(X)) \to f(g(X))$ can be considered as a variant of distributivity. This rule states that the relative order of f and g along any root-to-leaf path is irrelevant, i.e., f and g commute along any path. Consequently, variants of distributivity can be regarded as *vertical commutativity*. Typical examples of vertical commutativity are independent operations such as bold and italic text tags in HTML, which can be applied in any order, or more generally an operator g that is applied to all leaves of the subtree and hence $g(f(X, Y)) \to f(g(X), g(Y))$ holds.

Context cancellation rules. The general form of a *context cancellation rule* is $c(t) \to c(X)$, where c is a context and t is a term. The idempotency property of f expressed with $f(f(X)) \to f(X)$ is a context cancellation rule. Similarly, there are various cancellation rules in DL: $X \sqcap X \to X, X \sqcup X \to X$ and $(X \sqcap Y) \sqcup X \to X$.

4.2 Experiments

Implementation. We have implemented the L*algorithm adapted to bottom-up DFTA. Subsequently, we extended this implementation to handle advice for equivalence queries using our proposed mechanism. Our implementation accepts any set of rewrite rules as input, though our evaluation focuses on the TRS classes mentioned above.

Experiments with associativity. We have evaluated the impact of the advice $f(X, f(Y, Z)) \to f(f(X, Y), Z)$ on the number of equivalence queries on randomly generated automata. To ensure that the language of a randomly generated DFTA satisfies associativity, we generate the DFTA as follows. Let $\mathcal{F}$ be a signature consisting of a binary symbol f and constant symbols Σ. The *yield* of a tree t over $\mathcal{F}$ is the sequence of constants Σ in leaves read from left to right. For a regular language $\mathcal{K}$ over Σ, we define $\mathcal{T}^{\mathcal{K}}$ as the tree language consisting of all trees with the yield from $\mathcal{K}$. The membership of a tree in $\mathcal{T}^{\mathcal{K}}$ does not depend on its shape, which can be expressed by stating that f is associative. Intuitively, a tree t over $\mathcal{F}$ can be considered as a tree of height one with a single variable-arity symbol $\mathbf{f}$ in the head and constants from Σ as arguments, which is effectively an encoding of a sequence over constant symbols Σ.

Lemma 18. *Let $\mathcal{L}$ be a regular tree language over a binary symbol f and constants Σ. The rule $f(X, f(Y, Z)) \to f(f(X, Y), Z)$ is consistent with $\mathcal{L}$ if and only if $\mathcal{L} = \mathcal{T}^{\mathcal{K}}$ for some regular language $\mathcal{K}$ over Σ.*

Results for associativity. Based on Lemma 18, we have generated random DFA over the alphabets with 2–5 letters, with 2–16 states and converted each of them to a DFTA. These DFTA are guaranteed to satisfy associativity and their number of states ranged between 20 and 200. The conversion of DFA to DFTA incurs exponential blow-up as DFTA has to process leaves in various orders depending on the shape of the tree. We have removed trivial cases, when the L^* algorithm finished learning with at most 2 equivalence queries. We have observed reduction of equivalence queries between 29% (from 7 to 5 queries) to 96% (from 49 to 2 queries) with an average reduction of 77%. On average, this corresponds to a reduction from 12.4 to 2.6 equivalence queries.

Results for commutativity. We also evaluated the commutativity advice; however, it did not yield a reduction in the number of equivalence queries. Note that if $f(t_1, t_2)$ belongs to the language if and only if $f(t_2, t_1)$ does, then in any candidate automaton computed based on membership queries, the transition function for f is commutative. Commutativity could have been used to reduce the number of membership queries, but not in our framework as $f(X, Y) \to f(Y, X)$ is non-terminating, whereas termination is necessary to compute the normal form.

Results for distributivity. We have generated random DFTA over a binary symbol f and an unary symbol g such that f is distributive over g. The random DFTA had between 5 and 256 states. We have observed reduction of equivalence queries between none to 98% (from 57 to a single equivalence query) with an average reduction of 62%. On average, this corresponds to a reduction from 12.7 to 4.78 equivalence queries.

5 Generating Advice

While we have discussed how advice provided via a TRS can facilitate the learning of an unknown tree language, this section addresses the converse: *synthesis* of rewrite rules consistent with a given regular tree language. Given a tree automaton recognizing a tree language, the goal is to find a TRS consistent with that language. A synthesized TRS can serve as an explanation of the regular tree language, capturing its key properties through concise rewrite rules. We consider the automaton to be given rather than learned, as we show that even this easier variant of the problem is already difficult.

We begin by observing that the transition function of a DFTA $\mathcal{A}$ can be viewed as a ground term rewriting system. The resulting TRS $\mathcal{R}_\mathcal{A}$ *completely characterizes* the target language, i.e., if $\mathcal{R}_\mathcal{A}$ is used as the advice TRS, the language of $\mathcal{A}$ can be learned with a single equivalence query.

Fact 19. *Given a DFTA $\mathcal{A}$ we can compute a ground TRS $\mathcal{R}_\mathcal{A}$ that completely characterizes $\mathcal{L}(\mathcal{A})$.*

While $\mathcal{R}_\mathcal{A}$ demonstrates that the number of equivalence queries can be reduced to one, it is not practically relevant, as it is derived directly from the automaton being learned. Such detailed advice essentially describes the automaton itself, rendering the learning process redundant. Furthermore, this TRS is not structurally insightful as it does not explain the tree language any better than the automaton itself. Therefore, we are interested in synthesizing **non-ground** rewrite rules, where the left-hand side is a non-ground term, that are consistent with a given tree language and possibly small.

Formally, we study the following problem:

Definition 20 (Synthesis of TRS)**.** *Given a DFTA $\mathcal{A}$ over a signature $\mathcal{F}$, the Rule Synthesis problem is to find a non-ground rewrite rule $l \to r$, where $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, such that (1) $l \to r$ is consistent with $\mathcal{L}(\mathcal{A})$, (2) the value $|l| + |r|$ is minimal among rules satisfying (1).*

However, the hardness of the Rule Synthesis problem emerges already in the case of unary and constant symbols.

The unary case. Consider a signature $\mathcal{F}_{\text{unary}}$ consisting of unary symbols a, b, c and a constant $\#$. Let $\mathcal{A}$ be a DFTA over $\mathcal{F}_{\text{unary}}$. Observe that a ground term t over $\mathcal{F}_{\text{unary}}$ corresponds to a state of $\mathcal{A}$ with $\vec{\delta}_\mathcal{A}(t) = \widehat{\delta}(t)$, while a non-ground term s corresponds to a unary state transformation, i.e., $\vec{\delta}_\mathcal{A}(s)$ is a function from Q to Q.

The rule $l \to r$ is consistent with $\mathcal{L}(\mathcal{A})$ if and only if the transformation induced by l is a constant function equal to $\widehat{\delta}(r)$. If we view the unary symbols in l (read from bottom to top) as a word w_l over $\{a, b, c\}$, then w_l is a **synchronizing word** in $\mathcal{A}$ regarded as a DFA.

While it can be decided in polynomial time whether a DFA has a synchronizing word [Černỳ, 1964], finding the shortest synchronizing words in DFA is NP-complete [Eppstein, 1990] and it is even NP-hard to approximate it within a constant factor [Berlinkov, 2014]. The hardness of approximation carries directly from DFA to DFTA.

Lemma 21. *If $\mathsf{P} \neq \mathsf{NP}$, there is no polynomial-time algorithm that, given a DFTA $\mathcal{A}$ over $\mathcal{F}_{unary}$, approximates within a constant ratio the size of the shortest rewrite rule $l \to r$ such that r is a ground term and l is a non-ground term, and $l \to r$ is consistent with $\mathcal{L}(\mathcal{A})$.*

On the positive side, if we drop the minimality condition and restrict to contexts, the existence of such a rule is decidable in polynomial time similarly to [Černỳ, 1964]:

Lemma 22. *It is decidable in polynomial time, given a DFTA $\mathcal{A}$ over $\mathcal{F}$, whether there exists a rule $l \to r$ such that (1) l is a context and r is a ground term, and (2) $l \to r$ is consistent with $\mathcal{L}(\mathcal{A})$.*

Not only is the complexity of rule synthesis challenging, but there is also no single clear objective. While non-ground rules $l \to r$ are more meaningful than ground rules, several questions remain regarding the objectives for synthesis:

- Should both terms l and r be non-ground, or only l?
- Should l be permitted to contain constants?
- Should l be restricted to linear terms, or allowed to contain repeated variables (non-linear terms)?

Investigating these variations is essential for the design of future synthesis algorithms. We leave the comprehensive synthesis of rewrite rules as an open problem for future work.

6 Conclusions

We have presented a method that leverages structural knowledge of a target tree language to significantly reduce the number of equivalence queries required during active learning. In contrast to the word-automata setting, the branching structure of trees means that checking consistency of a TRS with a tree automaton, which is the core of our method, entails a higher computational complexity. Thus, a key direction for our future work involves developing further heuristics and investigating specific subclasses of TRSs to reduce the computational cost of consistency checking. While we have established several complexity bounds, the complexity of checking consistency of linear rules remains an open problem.

Beyond theoretical refinements, we aim to explore broader applications of this framework. Building on successful experiments with the associativity rewrite rule on synthetic data, we believe the framework is well-suited for more complex domains. Specifically, since our advice mechanism can express various properties of Description Logics (DL), applying active tree automaton learning with advice to learn DL terminologies represents a promising research path.

Finally, while we have only briefly explored the synthesis of consistent TRSs, we consider this a promising frontier for future research. Identifying the right objectives for such synthesis will be a prerequisite for developing robust automated advice-generation tools.

Acknowledgments

This work was supported by the National Science Centre (NCN), Poland under grant 2024/53/B/ST6/01620. We thank the anonymous reviewers for their valuable feedback.

References

[Aho *et al.*, 1972] A. V. Aho, M. R. Garey, and J. D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

[Anantharaman *et al.*, 2005] Siva Anantharaman, Paliath Narendran, and Michaël Rusinowitch. Closure properties and decision problems of dag automata. *Inf. Process. Lett.*, 94(5):231–240, 2005.

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

[Baader and Narendran, 1998] Franz Baader and Paliath Narendran. Unification of concept terms in description logics. In Henri Prade, editor, *ECAI 1998*, pages 331–335. John Wiley and Sons, 1998.

[Baader and Nipkow, 1998] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.

[Barbot *et al.*, 2021] Benoît Barbot, Benedikt Bollig, Alain Finkel, Serge Haddad, Igor Khmelnitsky, Martin Leucker, Daniel Neider, Rajarshi Roy, and Lina Ye. Extracting context-free grammars from recurrent neural networks using tree-automata learning and a* search. In *ICGI 2021*, pages 113–129. PMLR, 2021.

[Berlinkov, 2014] Mikhail V. Berlinkov. Approximating the minimum length of synchronizing words is hard. *Theory Comput. Syst.*, 54(2):211–223, 2014.

[Bhattamishra *et al.*, 2026] Satwik Bhattamishra, Michael Hahn, and Varun Kanade. Automata learning and identification of the support of language models. In *The Fourteenth International Conference on Learning Representations*, 2026.

[Björklund *et al.*, 2017] Henrik Björklund, Johanna Björklund, and Petter Ericson. On the regularity and learnability of ordered DAG languages. In Arnaud Carayol and Cyril Nicaud, editors, *CIAA 2017*, volume 10329 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2017.

[Boneva and Talbot, 2005] Iovka Boneva and Jean-Marc Talbot. Automata and logics for unranked and unordered trees. In Jürgen Giesl, editor, *RTA 2005*, volume 3467 of *Lecture Notes in Computer Science*, pages 500–515. Springer, 2005.

[Černỳ, 1964] Ján Černỳ. Poznámka k homogénnym experimentom s konečnỳmi automatmi. *Matematicko-fyzikálny časopis*, 14(3):208–216, 1964.

[Charatonik, 1999] Witold Charatonik. Automata on dag representations of finite trees, 1999.

[Comon *et al.*, 2008] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*. 2008.

[Dierl *et al.*, 2024] Simon Dierl, Paul Fiterau-Brostean, Falk Howar, Bengt Jonsson, Konstantinos Sagonas, and Fredrik Tåquist. Scalable tree-based register automata learning. In *TACAS 2024*, pages 87–108, 2024.

[Drewes and Högberg, 2003] Frank Drewes and Johanna Högberg. Learning a regular tree language from a teacher. In Zoltán Ésik and Zoltán Fülöp, editors, *DLT 2003*, volume 2710 of *Lecture Notes in Computer Science*, pages 279–291. Springer, 2003.

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

[Eppstein, 1990] David Eppstein. Reset sequences for monotonic automata. *SIAM J. Comput.*, 19(3):500–510, 1990.

[Fica and Otop, 2025] Michal Fica and Jan Otop. Active automata learning with advice. In *ECAI 2025*, volume 413 of *Frontiers in Artificial Intelligence and Applications*, pages 1655–1662. IOS Press, 2025.

[Frohme, 2019] Markus Theo Frohme. Active automata learning with adaptive distinguishing sequences. *CoRR*, abs/1902.01139, 2019.

[Grienenberger and Ritzert, 2019] Émilie Grienenberger and Martin Ritzert. Learning definable hypotheses on trees. In Pablo Barceló and Marco Calautti, editors, *ICDT 2019*, volume 127 of *LIPIcs*, pages 24:1–24:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[Habrard and Oncina, 2006] Amaury Habrard and José Oncina. Learning multiplicity tree automata. In Yasubumi Sakakibara, Satoshi Kobayashi, Kengo Sato, Tetsuro Nishino, and Etsuji Tomita, editors, *ICGI 2006*, volume 4201 of *Lecture Notes in Computer Science*, pages 268–280. Springer, 2006.

[Isberner *et al.*, 2014] Malte Isberner, Falk Howar, and Bernhard Steffen. The TTT algorithm: A redundancy-free approach to active automata learning. In *RV 2014*, pages 307–322, 2014.

[Kasprzik, 2013] Anna Kasprzik. Four one-shot learners for regular tree languages and their polynomial characterizability. *Theor. Comput. Sci.*, 485:85–106, 2013.

[Kopystiański and Otop, 2026a] Jakub Kopystiański and Jan Otop. Learning tree automata with term rewriting, 2026.

[Kopystiański and Otop, 2026b] Jakub Kopystiański and Jan Otop. Learning tree automata with term rewriting: code and data. https://github.com/jotop/LearnDFTAwithTRS, 2026.

[Kosala *et al.*, 2003] Raymond Kosala, Maurice Bruynooghe, Jan Van den Bussche, and Hendrik Blockeel. Information extraction from web documents based on local unranked tree automaton inference. In *IJCAI 2003*, pages 403–408. Morgan Kaufmann, 2003.

[Kruger *et al.*, 2024] Loes Kruger, Sebastian Junges, and Jurriaan Rot. Small test suites for active automata learning. In *TACAS 2024*, pages 109–129, 2024.

[Magnini *et al.*, 2025] Matteo Magnini, Riccardo Squarcialupi, Martin Tunge Sterri, Ana Ozaki, and Rivera Castillo. Actively learning el terminologies from large language models. In *ECAI 2025*, volume 413 of *Frontiers in Artificial Intelligence and Applications*, pages 1792–1799. IOS Press, 2025.

[Marusic and Worrell, 2015] Ines Marusic and James Worrell. Complexity of equivalence and learning for multiplicity tree automata. *Journal of Machine Learning Research*, 16:2465–2500, 2015.

[Nardi *et al.*, 2003] Daniele Nardi, Ronald J Brachman, et al. An introduction to description logics. *Description logic handbook*, 1:40, 2003.

[Nitay *et al.*, 2021] Dolav Nitay, Dana Fisman, and Michal Ziv-Ukelson. Learning of structurally unambiguous probabilistic grammars. In *AAAI 2021*, pages 9170–9178. AAAI Press, 2021.

[Raselimo and Fischer, 2021] Moeketsi Raselimo and Bernd Fischer. Automatic grammar repair. In Eelco Visser, Dimitris S. Kolovos, and Emma Söderberg, editors, *SLE 2021*, pages 126–142. ACM, 2021.

[Tirnăucă and Tirnăucă, 2007] Catalin Ionut Tirnăucă and Cristina Tirnăucă. Learning regular tree languages from correction and equivalence queries. *J. Autom. Lang. Comb.*, 12(4):501–524, 2007.

[Vaandrager *et al.*, 2022] Frits W. Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann. A new approach for active automata learning based on apartness. In *TACAS 2022*, pages 223–243, 2022.

[Vaandrager, 2017] Frits Vaandrager. Model learning. *Communications of the ACM*, 60(2):86–95, 2017.

[van Heerdt *et al.*, 2021] Gerco van Heerdt, Tobias Kappé, Jurriaan Rot, and Alexandra Silva. Learning pomset automata. In Stefan Kiefer and Christine Tasson, editors, *ETAPS 2021*, volume 12650, pages 510–530. Springer, 2021.

[Vazquez-Chanlatte *et al.*, 2025] Marcell Vazquez-Chanlatte, Karim Elmaaroufi, Stefan J. Witwicki, Matei Zaharia, and Sanjit A. Seshia. l^*lm: Learning automata from examples using natural language oracles, 2025.

[Yaghmazadeh *et al.*, 2016] Navid Yaghmazadeh, Christian Klinger, Isil Dillig, and Swarat Chaudhuri. Synthesizing transformations on hierarchically structured data. In Chandra Krintz and Emery D. Berger, editors, *PLDI 2016*, pages 508–521. ACM, 2016.