Journal of Machine Learning Research 1–17

Extracting Context-Free Grammars from Recurrent Neural Networks using Tree-Automata Learning and A* Search

Anonymous Authors

Abstract

This paper presents (i) an active learning algorithm for visibly pushdown grammars and (ii) shows its applicability for learning surrogate models of recurrent neural networks (RNNs) accepting context-free languages. Such surrogate models may be used for verification or explainability. Our learning algorithm makes use of the proximity of visibly pushdown languages and regular tree languages and builds on an existing learning algorithm for regular tree languages. Equivalence tests between a given RNN and a hypothesis grammar rely on a mixture of A* search and random sampling. Our approach scores by simplicity and accuracy, shown by its evaluation on a set of given RNNs.

1. Introduction

Context-free languages (CFLs), which are generated by context-free grammars (CFGs), abound in many application areas, for example when facing *formal* languages and applications such as programming languages and compilers, but especially also when processing natural language or controlled natural language. *Visibly pushdown languages* (VPLs), introduced by Alur and Madhusudan (2004, 2009), are a robust subclass of CFLs with interesting closure and decidability properties, as explained in further detail below—and are the class of languages studied in this paper. The idea is that the underlying pushdown automata are *input-driven* (Mehlhorn, 1980), i.e., every letter from the given alphabet is assigned a type among *push*, *pop*, and *internal* (we therefore deal with a *visibly pushdown alphabet*).

Learning *representations* or *models* such as (neural) networks, grammars, or automata from given examples or by querying underlying systems is an important tool when working with such languages. It has been considered in the field of machine learning for sequence-processing tasks such as time-series prediction or sentiment analysis, but also in the field of grammatical inference (Vaandrager, 2017). While in the first setting, typically, a finite set of words is given as a training set from which a model such as a recurrent neural network (RNN) is derived, in the second setting, further queries to a so-called *minimally adequate teacher* (MAT) may be asked to shape the learning result. A prominent MAT learning algorithm is Angluin's L* for regular word languages (Angluin, 1987).

In this paper, as a first contribution, we present a novel learning algorithm for VPLs, given a minimally adequate teacher.

An important application area of such a learning algorithm, as pursued in this paper, is to derive so-called *surrogate models* of recurrent neural networks (RNNs). RNNs play an important role in natural-language processing or time-series prediction, amongst others. While a neural network is often difficult to analyze and to understand, the surrogate model shares essential features of the underlying network but allows for simpler means for its analysis and explainability.

As a second contribution, we show that our algorithm can indeed be used for deriving a so-called visibly pushdown grammar (VPG) usable for explaining the language accepted by an underlying RNN. To this end, we perform queries to the network and infer an automaton model which is then translated into a grammar. The latter provides structural information of the underlying network which can hardly be obtained from the network directly.

Besides learning surrogate models applicable in verification and explainability settings, our algorithm in conjunction with machine learning may be used for grammatical inference-based learning in the setting with only positive examples (words to be accepted by the language). Surprisingly, machine learning works rather well with only positive examples. The subsequent learning based on the learned RNN leads to a grammar generalizing the given examples. However, we do not elaborate further on this idea in the present paper.

Our Approach. As mentioned above, our learning algorithm is for the class of VPLs. Alur and Madhusudan (2004) established a close relationship between VPLs and regular tree languages. We exploit this relationship and use an existing learning algorithm for regular tree languages (Sakakibara, 1992; Drewes and Högberg, 2007) to derive a grammar-based representation of a VPL, resulting in a MAT learning algorithm.

This is similar to Sakakibara's algorithm (Sakakibara, 1992), which infers CFGs in terms of tree automata learned using *structural* queries. In particular, a structural membership query comes with a skeleton, i.e., a tree that puts additional parse structure on top of the query and whose internal nodes are unlabeled. In our case, we do not have the same notion of skeleton, but use tree interpretations of the words that are queried.

In fact, Kumar et al. (2006) and Isberner (2015) had already pointed out that it would be possible to use the algorithm of Sakakibara (1992) for learning regular tree languages to obtain a tree representation of a VPL, albeit mentioning two potential obstacles for this. First, the final *visibly pushdown automaton* is non-deterministic, requiring thus the exponential cost in obtaining a deterministic one. Furthermore, in contrast to Kumar et al. (2006) and Isberner (2015), certain structural properties cannot be guaranteed that are expected from recursive programs. Our work focusing on practical learning of VPLs shows that these critical issues can be well handled by adapting the improved version of (Sakakibara, 1992) by Drewes and Högberg (2007) and by leveraging the computational power of RNNs.

One advantage of our algorithm as opposed to other algorithms for classes of CFLs is its simplicity: it is essentially based on the case for tree languages, so it is easy to understand. Moreover, its correctness essentially follows from the correctness of the tree-learning algorithm so that, in principle, we can plug-in any other tree-automata learning algorithm having the same interfaces. Another advantage is its extensibility to non-context-free languages, insofar as they have a representation as tree languages (Madhusudan and Parlato, 2011).

Application to RNNs. Our work is inspired by Yellin and Weiss (2021a), who infer CFGs from RNNs by extracting a sequence of DFAs using the algorithm proposed by Weiss et al. (2018), and exploiting the notion of pattern rule sets (PRSs), from which the CFG rules are derived. Experiments show that many interesting context-free RNN languages can be learned. There are nevertheless some difficulties to overcome. For example, very often a sequence of extracted DFAs contains some noise, either from the RNN training phase or from the application of the L^* algorithm. Consequently, incorrect patterns are frequently inserted into the DFA sequence, which can thus deviate from the PRS. To handle this, a useful voting and threshold scheme has been proposed. As a result, the languages of the majority of the given RNNs could be recovered in terms of CFGs, while several others were partially or incorrectly learned.

The class of VPLs is incomparable to the language class handled by Yellin and Weiss (2021a) (cf. Example 2 in Section 2). It must be fairly noted that our algorithm relies on a partitioning of the input alphabet into *push*, *pop*, and *internal* symbols, which is not required by Yellin and Weiss (2021a). However, it turns out that all the 15 benchmark languages considered by Yellin and Weiss (2021a) are VPLs.

In Yellin and Weiss (2021a), checking equivalence between the given RNN and a hypothesis grammar relies on an orthogonal learned abstraction of the RNN. In our case, the equivalence query relies on two complementary tests: The first uses an A*-based search in the given RNN to look for words that are in its language but not in the language of a hypothesis grammar; the second test samples words from the language of a hypothesis grammar, which are then checked for (not) being in the language of the RNN.

Apart from two exceptions, the languages from Yellin and Weiss (2021a) are very well learned with our approach, even some of the languages that are only partially generalized by applying the other approach. This demonstrates that our algorithm may be a worthwhile alternative when dealing with structured data (annotated linguistic data, programs, XML documents, etc.), i.e., in presence of a visibly pushdown alphabet.

Further Related Work. There are a wide range of learning algorithms for regular languages. Let us mention some of them. Angluin (1982) used reversibility to identify a class of regular languages from positive data alone using deterministic finite automata (DFAs), whose states are based on the residual languages or right congruence classes. Then, Angluin (1987) showed that the class of all regular languages could be learned using the L^* algorithm in the MAT model, where the teacher can answer both membership queries and equivalence queries. Rivest and Schapire (1993) proposed binary search to determine a single suffix of a counterexample that causes refinement, while Kearns and Vazirani (1994) suggested constructing a discrimination tree instead of the observation table. Then, Balcázar et al. (1997) provided a unified view on these learning algorithms, resulting in the observation pack framework, i.e., a family of observations that are organized in a certain way to satisfy certain properties such that one can construct an automaton from them. Isberner et al. (2014) presented the TTT algorithm, which is extremely efficient, especially in presence of long counterexamples, thanks to a refined counterexample-analysis and redundancy-free organization of observations.

Some researchers addressed learning CFLs by adapting learning algorithms for regular languages. For example, Clark and Eyraud (2007) presented an exact analogue of that proposed by Angluin (1982) for a limited class of CFLs, i.e., a learnability result could be established from positive data alone by combining the correspondence of non-terminals to the syntactic congruence class with weak substitutability. Then, Clark (2010) expanded this approach by adopting an extended MAT to answer equivalence queries where the hypothesis may not be in the learnable class. Yoshinaka and Clark (2010) extended the syntactic congruence to tuples of strings, with which one can efficiently learn some sorts of multiple CFGs. The hypothesis grammar calculated by their algorithm is however not always consistent with the observation tables with respect to the target grammar. This can be remedied by expanding observation tables, which requires exponential-time computation. Even though the above algorithms for learning CFLs have shown some promising results, they are limited to some constrained class. The learnability of the whole class of CFLs is widely believed to be intractable (de la Higuera, 2005).

Since decades, some approaches have been developed to extract simpler and explainable surrogate models from a neural network to facilitate comprehension and verification (Thrun, 1994; Omlin and Giles, 1996). New algorithms for extracting (weighted or unweighted) DFAs from RNNs have been proposed recently, with promising applications in verification (Weiss et al., 2018; Mayr and Yovine, 2018; Weiss et al., 2019; Mayr et al., 2020). They may also turn out to be useful for generalizations to other, more complex classes of languages. Up to now, however, there has been little research on extracting CFGs from RNNs. With the exception of (Yellin and Weiss, 2021a), existing approaches rely on an RNN augmented with external stack memory, either continuous or discrete (Das et al., 1992; Sun et al., 1997). In such a hybrid system, besides the classical input symbols, the input includes also what is read from the top of the stack.

Outline. Section 2 presents basic notions such as CFLs, CFGs, and VPLs. Trees and tree automata are presented in Section 3. In Section 4, we recall the tree-automata learning algorithm that we exploit, in Section 5, to learn grammars for VPLs. In Section 6, we apply our algorithm to inferring grammars from RNNs. We conclude in Section 7.

2. Context-Free and Visibly Pushdown Grammars

In this section, we recall standard notions and concepts such as context-free languages and grammars. We also present their subclass of visibly pushdown languages and the associated grammars our learning algorithm is based on.

2.1. Context-Free Languages and Grammars

Let Σ be an alphabet, i.e., a nonempty finite set. A word over Σ is a finite sequence $w = a_1 \ldots a_n$ of letters $a_i \in \Sigma$. The length $|w|$ of w is n. The unique word of length 0 is denoted by ε and is called the empty word. By Σ^*, we denote the set of all finite words over Σ.

Any set $L \subseteq \Sigma^*$ is called a language. For two languages $L_1, L_2 \subseteq \Sigma^*$, we let $L_1 \oplus L_2$ denote their symmetric difference, i.e., the language $(L_1 \setminus L_2) \cup (L_2 \setminus L_1)$.

For a finite set U, we denote by $\mathcal{P}(U)$ its powerset and by $|U|$ its size (the number of elements it contains).

Definition 1 *A* context-free grammar (CFG) *over* Σ *is a tuple* $G = (N, S, \rightarrow)$ *where* N *is a finite set of* nonterminal symbols *with* $N \cap \Sigma = \emptyset$*,* $S \in N$ *is the* start symbol*, and* $\rightarrow \subseteq N \times (\Sigma \cup N)^*$ *is the finite set of* rules*. A rule* $(A, w) \in \rightarrow$ *is usually written as* $A \rightarrow w$*.*

The language $L(G) \subseteq \Sigma^*$ of G is defined using a global rewrite relation $\Rightarrow \subseteq (\Sigma \cup N)^* \times (\Sigma \cup N)^*$ defined by $uAv \Rightarrow uwv$ for all rules $A \rightarrow w$ and $u, v \in (\Sigma \cup N)^*$. With this, we let $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$.

We say that $L \subseteq \Sigma^*$ is a *context-free language* (CFL) if there is a CFG over Σ such that $L(G) = L$. It should be noted that the class of CFLs enjoys equivalent characterizations, e.g., via pushdown automata.

Example 1 *For $n \geq 1$, consider the grammar G_n given by $S \to p_i S q_i S \mid \varepsilon$ (for all $i \in \{1, \ldots, n\}$) over the alphabet $\Sigma_n = \{p_1, \ldots, p_n, q_1, \ldots, q_n\}$. Then, $L(G_n)$ is the Dyck language of order n of well-bracketed words, where p_i is an opening and q_i its corresponding closing bracket.*

2.2. Visibly Pushdown Languages and Their Grammars

The class of *visibly pushdown languages* has been introduced by Alur and Madhusudan (2004, 2009). It was originally defined in terms of visibly pushdown automata, but can be equivalently characterized by a subclass of CFGs. VPLs constitute a robust class that, unlike the class of all CFLs, is closed under complement.

The idea is to assign to every letter from the given alphabet a precise role. Speaking in terms of automata, every symbol is either a push, a pop, or an internal symbol. This clearly is a restriction: A pushdown automaton recognizing the CFL $\{a^n b a^n \mid n \in \mathbb{N}\}$ has to perform a certain number of push operations while reading the first n occurrences of a, and pop operations when reading the remaining occurrences of a. On the other hand, $\{a^n b^n \mid n \in \mathbb{N}\}$ can be recognized by a pushdown automaton where a stack symbol is pushed when reading an a and a stack symbol is popped when reading a b. Accordingly, a *visibly pushdown alphabet* is an alphabet $\Sigma = \Sigma_{\mathsf{push}} \uplus \Sigma_{\mathsf{pop}} \uplus \Sigma_{\mathsf{int}}$ that is partitioned into *push*, *pop*, and *internal letters*. In the following, Σ will always denote a given visibly pushdown alphabet.

Definition 2 *A* visibly pushdown grammar (VPG) *over Σ is a CFG $(N, S, \to)$ such that every rule has one of the following forms (where $A, B, C \in N$): $A \to \varepsilon$ or $A \to bB$ with $b \in \Sigma_{\mathsf{int}}$ or $A \to aBbC$ with $a \in \Sigma_{\mathsf{push}}$ and $b \in \Sigma_{\mathsf{pop}}$.*

A language $L \subseteq \Sigma^*$ is called a *visibly pushdown language (VPL)* over Σ if there is a VPG G over Σ such that $L(G) = L$.

Example 2 *For $n \geq 1$, consider again the grammar G_n from Example 1. In fact, G_n is a VPG for $\Sigma_{\mathsf{push}} = \{p_1, \ldots, p_n\}$, $\Sigma_{\mathsf{pop}} = \{q_1, \ldots, q_n\}$, and $\Sigma_{\mathsf{int}} = \emptyset$ so that $L(G_n)$ is a VPL. Another example of a VPL is $\{a^n x b^n \mid n \in \mathbb{N}\}$ where $\Sigma_{\mathsf{push}} = \{a\}$, $\Sigma_{\mathsf{pop}} = \{b\}$, and $\Sigma_{\mathsf{int}} = \{x\}$. This language is not captured by the PRS-formalism presented by Yellin and Weiss (2021a) (cf. (Yellin and Weiss, 2021b, Section C.3)).*

We observe that, due to the form of permitted rules, a VPL L can only contain words $w \in \Sigma^*$ that are well-formed in a certain sense. The set $\mathcal{W}_\Sigma$ of *well-formed* words over Σ is defined as the language $L(G_\Sigma)$ of the "most permissive" VPG $G_\Sigma = (\{S\}, S, \to)$ with set of rules $\{S \to \varepsilon\} \cup \{S \to bS \mid b \in \Sigma_{\mathsf{int}}\} \cup \{S \to aSbS \mid a \in \Sigma_{\mathsf{push}} \text{ and } b \in \Sigma_{\mathsf{pop}}\}$.

Remark 3 *The general framework by Alur and Madhusudan (2004, 2009) can also cope with words that have unmatched push or pop positions. We restrict here to well-formed words, as the presentation is slightly simpler. However, the algorithms can be extended straightforwardly to the general case.*

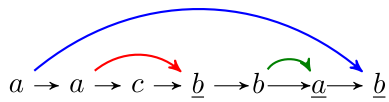

Figure 1: A nested word

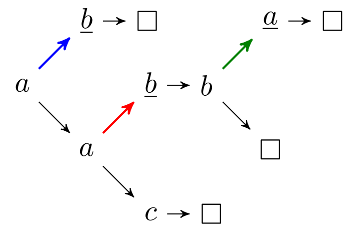

Figure 2: Its encoding as a tree

With $w = a_1 \ldots a_n \in \mathcal{W}_\Sigma$, we can associate a unique binary relation $\curvearrowright \subseteq \{1,\ldots,n\} \times \{1,\ldots,n\}$ connecting a push with a unique pop position: For $i, j \in \{1,\ldots,n\}$, we let $i \curvearrowright j$ if $i < j$, $a_i \in \Sigma_{\mathsf{push}}$, $a_j \in \Sigma_{\mathsf{pop}}$, and $a_{i+1} \ldots a_{j-1}$ is well-formed. We call the pair $(w, \curvearrowright)$ (with $w \in \mathcal{W}_\Sigma$) a *nested word*. A nested word over Σ with $\Sigma_{\mathsf{push}} = \{a, b\}$, $\Sigma_{\mathsf{pop}} = \{\underline{a}, \underline{b}\}$, and $\Sigma_{\mathsf{int}} = \{c\}$ is depicted in Figure 1. We do not exploit nested words in this paper, but it is helpful to think of well-formed words as nested words when we encode them as trees.

3. Trees and Regular Tree Languages

The reason why VPLs are so robust is that they are close to tree languages. In fact, nested words as introduced in the previous section can be represented as trees. Trees are defined over a ranked alphabet, i.e., an alphabet $\Gamma = \Gamma_0 \uplus \Gamma_1 \uplus \ldots \uplus \Gamma_{k_{max}}$ that is partitioned into letters of arity $k \in \{0,\ldots,k_{max}\}$ where $k_{max} \in \mathbb{N}$ is the maximal arity. Unless otherwise stated, we let Γ be a fixed ranked alphabet.

A tree t over Γ is a term that is generated according to the grammar $t ::= a(\underbrace{t,\ldots,t}_{k \text{ times}})$, where k ranges over $\{0,\ldots,k_{max}\}$ and a over Γ_k. Figure 2 depicts a syntax-tree-based representation of the tree $a(a(c(\Box()), \underline{b}(b(\Box(), \underline{a}(\Box())))), \underline{b}(\Box()))$ over the ranked alphabet given by $\Gamma_0 = \{\Box\}$, $\Gamma_1 = \{\underline{a}, \underline{b}, c\}$, and $\Gamma_2 = \{a, b\}$.

The size $|t|$ of t is the number of its nodes, i.e., the number of occurrences of symbols from Γ. Let $\mathsf{Trees}(\Gamma)$ denote the set of all trees over Γ.

The algorithm from Drewes and Högberg (2007), on which our approach is based, infers regular tree languages in terms of tree automata (later, when a tree automaton represents a VPL, we will be able to extract a corresponding VPG representation).

Definition 4 *A* nondeterministic finite (bottom-up) tree automaton (NTA) *over Γ is a tuple $\mathcal{B} = (Q, \delta, F)$ where Q is the nonempty finite set of states, $F \subseteq Q$ is the set of final states, and $\delta : \bigcup_{k \in \{0,\ldots,k_{max}\}} (\Gamma_k \times Q^k) \to \mathcal{P}(Q)$ is the transition function. One usually writes $\delta(a(q_1,\ldots,q_k))$ instead of $\delta(a, q_1,\ldots,q_k)$.*

We call $\mathcal{B}$ is deterministic *(a DTA) if $|\delta(a(q_1,\ldots,q_k))| = 1$ for all arguments $a, q_1,\ldots,q_k$. Then, the transition function can be seen to be of type $\delta : \bigcup_{k \in \{0,\ldots,k_{max}\}} (\Gamma_k \times Q^k) \to Q$. We let $\mathsf{DTA}(\Gamma)$ denote the set of DTAs over Γ.*

From δ, we obtain a function $\hat{\delta} : \mathsf{Trees}(\Gamma) \to \mathcal{P}(Q)$ letting, for an arity $k \in \{0,\ldots,k_{max}\}$, $a \in \Gamma_k$, and $t_1,\ldots,t_k \in \mathsf{Trees}(\Gamma)$, $\hat{\delta}(a(t_1,\ldots,t_k)) = \bigcup_{q_1 \in \hat{\delta}(t_1),\ldots,q_k \in \hat{\delta}(t_k)} \delta(a(q_1,\ldots,q_k))$. We can now define the tree language recognized by $\mathcal{B}$ as $T(\mathcal{B}) = \{t \in \mathsf{Trees}(\Gamma) \mid \hat{\delta}(t) \cap F \neq \emptyset\}$.

In the following, we will call a tree language $T \subseteq \mathsf{Trees}(\Gamma)$ *regular* if it is recognized by some NTA.

We now state some important and well-known facts about tree automata. For more details, we refer the reader to (Comon et al., 2007).

Fact 1 (minimal DTA) *For every NTA $\mathcal{B} = (Q, \delta, F)$, there is a unique (up to isomorphism) minimal DTA $\mathcal{B}' = (Q', \delta', F')$ such that $T(\mathcal{B}') = T(\mathcal{B})$. We can assume $|Q'| \leq 2^{|Q|}$.*

The *index* of a regular tree language T is the number of states of the minimal DTA recognizing T.

While DTAs capture the class of regular tree languages, deterministic *top-down* finite tree automata (Comon et al., 2007), which we do not define here, are strictly less expressive.

Fact 2 (membership and emptiness) *(i) Given an NTA $\mathcal{B}$ and a tree $t \in \mathsf{Trees}(\Gamma)$, one can decide in polynomial time whether $t \in T(\mathcal{B})$. For DTAs, there is a linear-time algorithm. (ii) For a given NTA $\mathcal{B}$, one can decide in polynomial time whether $T(\mathcal{B}) \neq \emptyset$.*

4. Learning Deterministic Tree Automata

In her seminal work, Angluin (1987) provided the algorithm L*, which can infer a deterministic finite automaton for a given regular word language that can only be accessed via two types of queries: *membership queries (MQs)* and *equivalence queries (EQs)*. The algorithm has later been extended, first by Sakakibara to CFGs and then by Drewes and Högberg (2007) to tree automata. The latter algorithm, called TL* in this paper, can infer a DTA over a fixed ranked alphabet Γ for a given (unknown) regular tree language T. Hereby, T can be accessed through membership queries and equivalence queries, which are implemented by "oracle" mappings $\mathsf{MQ}_{\text{tree}} : \mathsf{Trees}(\Gamma) \to \{\mathsf{yes}, \mathsf{no}\}$ and $\mathsf{EQ}_{\text{tree}} : \mathsf{DTA}(\Gamma) \to \{\mathsf{yes}\} \cup \mathsf{Trees}(\Gamma)$:

- We say that $\mathsf{MQ}_{\text{tree}}$ is *sound* for T if, for all $t \in \mathsf{Trees}(\Gamma)$, $\mathsf{MQ}_{\text{tree}}(t) = \mathsf{yes}$ iff $t \in T$.
- We say that $\mathsf{EQ}_{\text{tree}}$ is *counterexample-sound* for T if, for all $\mathcal{B} \in \mathsf{DTA}(\Gamma)$ and $t \in \mathsf{Trees}(\Gamma)$ such that $\mathsf{EQ}_{\text{tree}}(\mathcal{B}) = t$, we have $t \in T \oplus T(\mathcal{B})$.
- We call $\mathsf{EQ}_{\text{tree}}$ *equivalence-sound* for T if, for all $\mathcal{B} \in \mathsf{DTA}(\Gamma)$ such that $\mathsf{EQ}_{\text{tree}}(\mathcal{B}) = \mathsf{yes}$, we have $T = T(\mathcal{B})$.

Ideally, one assumes that $\mathsf{EQ}_{\text{tree}}$, which checks the current *hypothesis* computed by the learning algorithm, is both counterexample- and equivalence-sound. However, in practice, this is not always the case. In fact, in our experiments, we will make weaker assumptions on the mapping $\mathsf{EQ}_{\text{tree}}$.

The algorithm TL* by Drewes and Högberg (2007) takes as input a ranked alphabet Γ and two functions $\mathsf{MQ}_{\text{tree}} : \mathsf{Trees}(\Gamma) \to \{\mathsf{yes}, \mathsf{no}\}$ and $\mathsf{EQ}_{\text{tree}} : \mathsf{DTA}(\Gamma) \to \{\mathsf{yes}\} \cup \mathsf{Trees}(\Gamma)$. If TL$^*(\Gamma, \mathsf{MQ}_{\text{tree}}, \mathsf{EQ}_{\text{tree}})$ terminates, it outputs a DTA over Γ.

Fact 3 (Drewes and Högberg (2007)) *Let $T \subseteq \mathsf{Trees}(\Gamma)$ be a regular tree language, say with index n. Suppose $\mathsf{MQ}_{\text{tree}}$ is sound for T and that $\mathsf{EQ}_{\text{tree}}$ is both counterexample- and equivalence-sound for T. Then, TL$^*(\Gamma, \mathsf{MQ}_{\text{tree}}, \mathsf{EQ}_{\text{tree}})$ terminates and outputs the unique DTA $\mathcal{B}$ with n states such that $T(\mathcal{B}) = T$. The overall running time is polynomial in $|\Gamma|$, $n^{k_{max}}$, and the maximal size of a counterexample returned by $\mathsf{EQ}_{\text{tree}}$.*

In the next sections, k_{max} will be fixed so that we deal with a polynomial-time algorithm.

5. Learning Visibly Pushdown Grammars

In this section, we exploit tree-automata learning for the inference of VPLs in terms of VPGs. The derived algorithm will then be exploited to extract grammars from RNNs.

5.1. Encoding Nested Words as Trees

The main link between words and trees is provided by an encoding of well-formed words as trees over a suitable ranked alphabet (Alur and Madhusudan, 2004, 2009).

Let $\Sigma = \Sigma_{\mathsf{push}} \uplus \Sigma_{\mathsf{pop}} \uplus \Sigma_{\mathsf{int}}$ be a visibly pushdown alphabet. To encode words from $\mathcal{W}_\Sigma$ as trees, we introduce a suitable ranked alphabet $\Gamma = \Gamma_0 \uplus \Gamma_1 \uplus \Gamma_2$ letting $\Gamma_0 = \{\Box\}$, $\Gamma_1 = \Sigma_{\mathsf{pop}} \cup \Sigma_{\mathsf{int}}$, and $\Gamma_2 = \Sigma_{\mathsf{push}}$. That is, the maximal arity is 2. To a well-formed word $w \in \mathcal{W}_\Sigma$, we inductively assign a tree $\langle\!\langle w \rangle\!\rangle \in \mathsf{Trees}(\Gamma)$ as follows: (i) We let $\langle\!\langle \varepsilon \rangle\!\rangle = \Box()$. (ii) If $w = aw_1bw_2$ such that $a \in \Sigma_{\mathsf{push}}$, $b \in \Sigma_{\mathsf{pop}}$, and w_1 and w_2 are well-formed, then $\langle\!\langle w \rangle\!\rangle = a(\langle\!\langle w_1 \rangle\!\rangle, b(\langle\!\langle w_2 \rangle\!\rangle))$. (iii) If $c \in \Sigma_{\mathsf{int}}$ and w is well-formed, then $\langle\!\langle cw \rangle\!\rangle = c(\langle\!\langle w \rangle\!\rangle)$. The encoding of the well-formed word from Figure 1 is illustrated in Figure 2.

Given $L \subseteq \mathcal{W}_\Sigma$, we let $\langle\!\langle L \rangle\!\rangle = \{\langle\!\langle w \rangle\!\rangle \mid w \in L\} \subseteq \mathsf{Trees}(\Gamma)$. Moreover, we let $\mathcal{T}_\Gamma = \langle\!\langle \mathcal{W}_\Sigma \rangle\!\rangle$ be the set of trees that encode a well-formed word. Note that $\langle\!\langle . \rangle\!\rangle : \mathcal{W}_\Sigma \to \mathcal{T}_\Gamma$ is injective and, therefore, a bijection. Indeed, its inverse mapping, which we denote by $[\![.]\!]$, is given by $[\![\Box()]\!] = \varepsilon$, $[\![a(t_1, b(t_2))]\!] = a[\![t_1]\!]b[\![t_2]\!]$ and $[\![c(t)]\!] = c[\![t]\!]$. For $T \subseteq \mathcal{T}_\Gamma$, let $[\![T]\!] = \{[\![t]\!] \mid t \in T\}$.

Let us state some known facts on the relation between VPGs and NTAs/DTAs due to Alur and Madhusudan (2004, 2009).

Fact 4 *For every VPL L over Σ, there is an NTA (or DTA) $\mathcal{B}$ over Γ such that $T(\mathcal{B}) = \langle\!\langle L \rangle\!\rangle$. In particular, there is a DTA $\mathcal{B}_{\mathsf{parse}}$ over Γ with a constant number of states such that $T(\mathcal{B}_{\mathsf{parse}}) = \mathcal{T}_\Gamma$.*

As we will extract grammars from tree automata, the following is particularly important:

Fact 5 *Let $\mathcal{B}$ be an NTA over Γ such that $T(\mathcal{B}) \subseteq \mathcal{T}_\Gamma$. One can compute, in polynomial time, a VPG $\mathrm{nta2vpg}(\mathcal{B})$ over Σ such that $L(\mathrm{nta2vpg}(\mathcal{B})) = [\![T(\mathcal{B})]\!]$.*

We give the translation of an NTA into a VPG, as the latter will yield the representation of a VPL learned in terms of the NTA. Suppose $\mathcal{B} = (Q, \delta, F)$ is an NTA over Γ such that $T(\mathcal{B}) \subseteq \mathcal{T}_\Gamma$. We define $\mathrm{nta2vpg}(\mathcal{B}) = (N, \mathcal{I}, \to)$ as follows. In fact, instead of just one start symbol, we assume a set of start symbols $\mathcal{I} \subseteq N$, which is easily seen to be equivalent. Intuitively, the grammar derives a run of the NTA top-down, where states are successively replaced with input letters. So we let $N = Q$ and $\mathcal{I} = F$. Moreover, the set of rules contains (i) $\hat{q} \to \varepsilon$ for all $\hat{q} \in \delta(\Box())$; (ii) $\hat{q} \to cq$ for all $c \in \Sigma_{\mathsf{int}}$, $q \in Q$, and $\hat{q} \in \delta(c(q))$; (iii) $\hat{q} \to apbq$ for all $a \in \Sigma_{\mathsf{push}}$, $b \in \Sigma_{\mathsf{pop}}$, and $p, q, q', \hat{q} \in Q$ such that $q' \in \delta(b(q))$ and $\hat{q} \in \delta(a(p, q'))$.

For completeness, let us mention some connections with visibly pushdown automata (VPAs), which are effectively equivalent to VPGs wrt. expressive power so that we could also learn VPAs instead of VPGs (cf. (Alur and Madhusudan, 2004, 2009) or Appendix A for the definition of VPAs). For an NTA $\mathcal{B}$ over Γ such that $T(\mathcal{B}) \subseteq \mathcal{T}_\Gamma$, one can compute,

Algorithm 1 Implementing $\mathsf{MQ}_{\text{tree}}$ in terms of MQ_{vpl}

1 $\mathsf{MQ}_{\text{tree}}(t)$:
2   **if** $t \in T(\mathcal{B}_{\mathsf{parse}})$
3   **then return** $\mathsf{MQ}_{\text{vpl}}(\llbracket t \rrbracket)$
4   **else return** no

Algorithm 2 Implementing $\mathsf{EQ}_{\text{tree}}$ in terms of EQ_{vpl}

1 $\mathsf{EQ}_{\text{tree}}(\mathcal{B})$:
2   **if** $T(\mathcal{B}) \subseteq T(\mathcal{B}_{\mathsf{parse}})$
3   **then return** $\mathsf{EQ}_{\text{vpl}}(\mathcal{B})$
4   **else**
5   pick small $t \in T(\mathcal{B}) \setminus T(\mathcal{B}_{\mathsf{parse}})$
6   **return** t

in polynomial time, a VPA $\mathcal{A}$ over Σ such that $L(\mathcal{A}) = \llbracket T(\mathcal{B}) \rrbracket$. Conversely, for a VPA $\mathcal{A}$ over Σ, one can compute, in polynomial time, an NTA $\mathcal{B}$ over Γ such that $T(\mathcal{B}) = \langle\!\langle L(\mathcal{A}) \rangle\!\rangle$. Hence, there is also a DTA for $\langle\!\langle L(\mathcal{A}) \rangle\!\rangle$ of exponential size. In general, this exponential blow-up cannot be avoided even when we start from a deterministic VPA.

5.2. Learning VPLs in Terms of VPGs

We now present an algorithm, called VPL* in the following, that learns a VPL $L \subseteq \mathcal{W}_\Sigma$ in terms of a DTA for the tree language $\langle\!\langle L \rangle\!\rangle \subseteq \mathcal{T}_\Gamma$ that can then be translated into a VPG according to Fact 5. Essentially, we rely on the algorithm TL*. However, equivalence and membership queries are now answered wrt. the VPL L. More precisely, we deal with a mapping $\mathsf{MQ}_{\text{vpl}} : \mathcal{W}_\Sigma \to \{\mathsf{yes}, \mathsf{no}\}$ and a partial mapping $\mathsf{EQ}_{\text{vpl}} : \mathsf{DTA}(\Gamma) \to \{\mathsf{yes}\} \cup \mathcal{T}_\Gamma$ whose domain is the set of DTAs $\mathcal{B} \in \mathsf{DTA}(\Gamma)$ such that $T(\mathcal{B}) \subseteq \mathcal{T}_\Gamma$:

- We call MQ_{vpl} *sound* for L if, for all $w \in \mathcal{W}_\Sigma$, we have $\mathsf{MQ}_{\text{vpl}}(w) = \mathsf{yes}$ iff $w \in L$.
- We say that EQ_{vpl} is *counterexample-sound* for L if, for all $\mathcal{B} \in \mathsf{DTA}(\Gamma)$ such that $T(\mathcal{B}) \subseteq \mathcal{T}_\Gamma$ and all $t \in \mathcal{T}_\Gamma$, $\mathsf{EQ}_{\text{vpl}}(\mathcal{B}) = t$ implies $\llbracket t \rrbracket \in L \oplus \llbracket T(\mathcal{B}) \rrbracket$.
- We say that EQ_{vpl} is *equivalence-sound* for L if, for all $\mathcal{B}$ over Γ such that $T(\mathcal{B}) \subseteq \mathcal{T}_\Gamma$, $\mathsf{EQ}_{\text{vpl}}(\mathcal{B}) = \mathsf{yes}$ implies $L = \llbracket T(\mathcal{B}) \rrbracket$.

Our algorithm VPL* for learning VPLs uses TL* as a black-box. Therefore, we define a mapping $\mathsf{MQ}_{\text{tree}} : \mathsf{Trees}(\Gamma) \to \{\mathsf{yes}, \mathsf{no}\}$ and a mapping $\mathsf{EQ}_{\text{tree}} : \mathsf{DTA}(\Gamma) \to \{\mathsf{yes}\} \cup \mathsf{Trees}(\Gamma)$ that implement the membership and equivalence queries for tree languages, respectively (cf. Algorithms 1 and 2). The algorithm VPL* then simply calls TL* with parameters $(\Gamma, \mathsf{MQ}_{\text{tree}}, \mathsf{EQ}_{\text{tree}})$ and translates the resulting DTA into a VPG (Algorithm 3).

Algorithm 1. Membership query $\mathsf{MQ}_{\text{tree}}(t)$ with $t \in T(\mathcal{B}_{\mathsf{parse}}) = \mathcal{T}_\Gamma$ is answered in terms of $\mathsf{MQ}_{\text{vpl}}(\llbracket t \rrbracket)$ (line 3). If, on the other hand, $t \notin T(\mathcal{B}_{\mathsf{parse}})$, the query returns no (line 4).

Algorithm 2. Recall that we are looking for a tree automaton for the language $T = \langle\!\langle L \rangle\!\rangle$, which is included in $T(\mathcal{B}_{\mathsf{parse}})$. We will, therefore, first check whether this inclusion also applies to the current hypothesis DTA $\mathcal{B}$, i.e., whether $T(\mathcal{B}) \subseteq T(\mathcal{B}_{\mathsf{parse}})$. If not, then we can find (efficiently) a "small" tree $t \in T(\mathcal{B}) \setminus T(\mathcal{B}_{\mathsf{parse}})$, which serves as a counterexample to the equivalence query (line 5). So suppose that $T(\mathcal{B}) \subseteq T(\mathcal{B}_{\mathsf{parse}})$. Let us assume that EQ_{vpl} is both counterexample- and equivalence-sound. If it returns a tree $t = \mathsf{EQ}_{\text{vpl}}(\mathcal{B})$, then

Algorithm 3 VPL*

```
B ← TL*(Γ, MQ_tree, EQ_tree)   /* MQ_tree and EQ_tree from Algorithms 1 and 2 */
return nta2vpg(B)
```

$[\![t]\!] \in L \oplus [\![T(\mathcal{B})]\!]$ so that t can indeed be used to refine the hypothesis $\mathcal{B}$. If, on the other hand, $\mathsf{EQ}_{\mathrm{vpl}}(\mathcal{B}) = \mathsf{yes}$, then $L = [\![T(\mathcal{B})]\!]$, i.e., $T(\mathcal{B}) = \langle\!\langle L \rangle\!\rangle$, so that we can return $\mathcal{B}$ as a suitable tree-language representation. Algorithm 3 then returns the VPG $G = \mathrm{nta2vpg}(\mathcal{B})$. According to Fact 5, we have $L(G) = [\![T(\mathcal{B})]\!] = L$.

Correctness of VPL* are stated in the following theorem (cf. Appendix B for the proof):

Theorem 5 *Let L be a VPL and $\hat{\mathcal{B}}$ be the minimal DTA such that $T(\hat{\mathcal{B}}) = \langle\!\langle L \rangle\!\rangle$. Assume $\mathsf{MQ}_{\mathrm{vpl}}$ is sound for L and that $\mathsf{EQ}_{\mathrm{vpl}}$ is both counterexample- and equivalence-sound for L. Then,* VPL* *(Algorithm 3) terminates and eventually returns a VPG G of size polynomial in the size of $\hat{\mathcal{B}}$ such that $L(G) = L$. The overall running time is polynomial in $|\Sigma|$, the index of $\hat{\mathcal{B}}$, and the maximal size of a counterexample returned in lines 3 and 6 of Algorithm 2.*

Note that the size of the returned VPG G is at most exponential in the size of a minimal (nondeterministic) VPA recognizing L.

6. Experiments

We applied Algorithm 3 to recurrent neural networks (RNNs) in order to extract VPGs. We implemented it in Python 3.6, using the Numpy library.[1] Moreover some code from (Yellin and Weiss, 2021a, `https://github.com/tech-srl/RNN_to_PRS_CFG`) was used for comparisons. All benchmarks were performed on a computer equipped by Intel i5-8250U CPU with 4 cores, 16GB of memory, and Ubuntu Linux 18.03.

Recurrent Neural Networks. Just like automata, RNNs can be seen as language acceptors. For the purpose of this paper, it is enough to think of an RNN R as an infinite automaton with state space $\mathbb{R}^{dim}$, for some dimension $dim \geq 1$, along with a mapping from $\mathbb{R}^{dim}$ to $\mathbb{R}$ so that R finally computes a (score) function $R : \Sigma^* \to \mathbb{R}$. Given a threshold $t \in \mathbb{R}$, we then obtain its language as $L(R) = \{w \in \Sigma^* \mid R(w) > t\}$. Several well-known architectures are available to effectively represent RNNs, such as (simple) Elman RNNs, LSTM (Hochreiter and Schmidhuber, 1997), and GRUs (Cho et al., 2014). Generally, depending on the architecture, the expressive power of RNNs goes beyond the regular languages. This is why it is worthwhile to study extraction methods for classes of CFLs.

Methodology and Results. The 15 CFLs considered by Yellin and Weiss (2021a) are given in Table 2, together with the CFGs from Table 1. For conciseness, they are defined in terms of general CFGs. However, it turns out that all of them are VPLs. In most cases, there is arguably a canonical partition of the alphabet into a visibly pushdown alphabet. For all these VPLs, we considered the RNNs provided by Yellin and Weiss (2021a), which were trained on sample sets generated by a probabilistic version of a corresponding CFG.

1. In accordance with the double-blind review process, if the paper is accepted, the code will be made available publicly on github at publication time.

Table 1: Definition of some CFLs (X and Y are finite sets of words)

$\mathcal{L}(X,Y)$:	$\mathit{RE\text{-}Dyck}(X,Y)$:	$\mathit{Dyck}_1 \triangleleft X$:
$S \to \varepsilon \mid xSy$	$S \to xAy$	$S \to p_1 A q_1$
(for all $x \in X$ and $y \in Y$)	$A \to xAy \mid AA \mid \varepsilon$	$A \to p_1 A q_1 \mid AA \mid \varepsilon \mid x$
	(for all $x \in X$ and $y \in Y$)	(for all $x \in X$)
Dyck_n:	*Alternating*:	$\mathit{Dyck}_2 \triangleleft X$:
$S \to p_i A q_i$	$S \to A \mid B$	$S \to p_i A q_i$
$A \to p_i A q_i \mid AA \mid \varepsilon$	$A \to (B) \mid \varepsilon$	$A \to p_i A q_i \mid AA \mid \varepsilon \mid x$
(for all $i \in \{1, \ldots, n\}$)	$B \to [A] \mid \varepsilon$	(for all $i \in \{1, 2\}$ and $x \in X$)

In our experiments, we used a Kearns-Vazirani variation of TL*. A query $\mathsf{MQ}_{\mathrm{vpl}}(w)$ for a well-formed word w was answered according to the given RNN R, i.e., $\mathsf{MQ}_{\mathrm{vpl}}(w) = \mathsf{yes}$ iff $w \in L(R)$. To answer a query $\mathsf{EQ}_{\mathrm{vpl}}(\mathcal{B})$, we used two independent subroutines that look for counterexample words (of length under 30):

(i) We chose 1500 random words in the current hypothesis language $[\![T(\mathcal{B})]\!]$.

(ii) We generated a set P (performed once in the beginning of the run) of positive examples from the RNN language (only *well-formed words*; timeout of 60 seconds). To do so, we used the A* search algorithm (cf. (Russell and Norvig, 2020)) on the reachability graph induced by the RNN states, along with the evaluation function $f(w) = |w|^{-2} \left(\sum_{w' \in \Sigma^d} R(ww')\right) / |\Sigma^d|$, where $d = 4$ and $R(w)$ is the score the RNN gives to w. By adding $|w|^{-2}$, we gave priority to shorter length. Note that the function f strongly depends on the implementation of the RNN and its language. Then, we sorted P according to the length of the words (the shorter the better) and the score of the RNN (the higher the better). We kept only the best 1500 examples.

Note that $\mathsf{EQ}_{\mathrm{vpl}}$ is counterexample-sound for $L(R)$ but not necessarily equivalence-sound. However, it is sufficiently precise in practice. Though the given trained RNNs have imperfections, the intended languages are learned in most cases. Table 2 indicates the time needed to learn a VPG, averaging across several runs, and the number of rules extracted. In most runs, the extracted VPGs are equivalent to the respective CFGs the RNNs were trained on. Exceptions are L_{14} and L_{15} for which we obtain grammars under-approximating the respective languages. This happens due to structural errors in the given RNNs.

To give a (successful) example, Table 3 depicts the grammar that was output for L_{10}.

Fixing Mistakes Variation. The previous result can easily be ruined by a wrong sample of words. For example, we could pick a word that is in the RNN language but not in the original language. To mitigate this problem, one can do the following: Denote by P the set of positive examples generated from the RNN, let H be the current hypothesis grammar, and let $\mathit{pos}(H) = |P \cap L(H)| \,/\, |P|$. Assume that H comes with a counterexample w_c and a new hypothesis H'. If $\mathit{pos}(H') < \mathit{pos}(H)$, then we keep refining both of them, but making sure that w_c cannot be a counterexample for H. In the end, we return the hypothesis with the highest "probability". For example, by increasing the sampling length from the

Table 2: Results for learning RNNs

	Language	Visibly Pushdown Alphabet Push	Pop	Int	#Rules	Time
L_1	$\mathcal{L}(\{a\},\{b\})$	$\{a\}$	$\{b\}$		3	1s
L_2	$\mathcal{L}(\{a,b\},\{c,d\})$	$\{a,b\}$	$\{c,d\}$		9	23s
L_3	$\mathcal{L}(\{ab,cd\},\{ef,gh\})$	$\{a,b,c,d\}$	$\{e,f,g,h\}$		13	74s
L_4	$\mathcal{L}(\{ab\},\{cd\})$	$\{a,b\}$	$\{c,d\}$		4	1s
L_5	$\mathcal{L}(\{abc\},\{def\})$	$\{a,b,c\}$	$\{d,e,f\}$		5	1s
L_6	$\mathcal{L}(\{ab,c\},\{de,f\})$	$\{a,c\}$	$\{d,f\}$	$\{b,e\}$	10	49s
L_7	$Dyck_2$	$\{p_1,p_2\}$	$\{q_1,q_2\}$		19	69s
L_8	$Dyck_3$	$\{p_1,p_2,p_3\}$	$\{q_1,q_2,q_3\}$		28	74s
L_9	$Dyck_4$	$\{p_1,\dots,p_4\}$	$\{q_1,\dots,q_4\}$		37	79s
L_{10}	$RE\text{-}Dyck(\{(abcd\},\{wxyz)\})$	$\{(,a,b,c,d\}$	$\{w,x,y,z,)\}$		10	7s
L_{11}	$RE\text{-}Dyck(\{ab,c\},\{de,f\})$	$\{a,c\}$	$\{d,f\}$	$\{b,e\}$	27	59s
L_{12}	*Alternating*	$\{(,[\}$	$\{),]\}$		5	2s
L_{13}	$Dyck_1 \triangleleft \{a,b,c\}$	$\{p_1\}$	$\{q_1\}$	$\{a,b,c\}$	19	66s
L_{14}	$Dyck_2 \triangleleft \{a,b,c\}$	$\{p_1,p_2\}$	$\{q_1,q_2\}$	$\{a,b,c\}$	–	65s
L_{15}	$Dyck_1 \triangleleft \{abc,d\}$	$\{p_1\}$	$\{q_1\}$	$\{a,b,c,d\}$	–	51s

Table 3: Learned VPG for L_{10} with start symbol $\mathtt{A}_1$

$\mathtt{A}_1 \to (\,\mathtt{A}_2\,)\,\mathtt{A}_0$	$\mathtt{A}_2 \to a\,\mathtt{A}_3\,z\,\mathtt{A}_0$	$\mathtt{A}_4 \to c\,\mathtt{A}_5\,x\,\mathtt{A}_0$	$\mathtt{A}_5 \to d\,\mathtt{A}_1\,w\,\mathtt{A}_0$	$\mathtt{A}_6 \to (\,\mathtt{A}_2\,)\,\mathtt{A}_1$
$\mathtt{A}_0 \to \varepsilon$	$\mathtt{A}_3 \to b\,\mathtt{A}_4\,y\,\mathtt{A}_0$	$\mathtt{A}_5 \to d\,\mathtt{A}_0\,w\,\mathtt{A}_0$	$\mathtt{A}_5 \to d\,\mathtt{A}_6\,w\,\mathtt{A}_0$	$\mathtt{A}_6 \to (\,\mathtt{A}_2\,)\,\mathtt{A}_6$

RNN ($30 \to 40$) and the size of the sample set ($1500 \to 2000$), we ruined the extraction of language L_8, but using the procedure above we manage to fixed this issue.

Agnostic Learning. Some criticism might be given to the fact that we assume the visibly pushdown alphabet to be known. To solve this issue, we generated a set P of positive words from the RNN like before. Using P, we examined all the possible visibly alphabets (there may be several), picked the best suited alphabet (with the least number of internal symbols), and continued learning. We succeeded in 8 of the 13 languages that were successful in the non-agnostic case.

7. Conclusion

We presented an algorithm to learn VPLs in the MAT framework. As an application, we focused on the extraction of grammars from RNNs. Our experiments suggest that the algorithm is a suitable alternative to current approaches when we deal with structured data. Learning VPLs has potential applications in formal verification (Alur and Madhusudan, 2004, 2009), so it would be worthwhile to conduct an evaluation in that domain, too.

References

Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004. doi: 10.1145/1007352.1007390. URL https://doi.org/10.1145/1007352.1007390.

Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):16:1–16:43, 2009. doi: 10.1145/1516512.1516518. URL https://doi.org/10.1145/1516512.1516518.

Dana Angluin. Inference of reversible languages. *J. ACM*, 29(3):741–765, 1982.

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

José L. Balcázar, Josep Díaz, and Ricard Gavaldà. Algorithms for learning finite automata from queries: A unified view. In *Advances in Algorithms, Languages, and Complexity - In Honor of Ronald V. Book*, pages 53–72. Kluwer, 1997.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proc. EMNLP*, pages 1724–1734. ACL, 2014.

Alexander Clark. Distributional learning of some context-free languages with a minimally adequate teacher. In José M. Sempere and Pedro García, editors, *Proc. of ICGI 2010*, volume 6339 of *LNCS*, pages 24–37. Springer, 2010. doi: 10.1007/978-3-642-15488-1_4. URL https://doi.org/10.1007/978-3-642-15488-1_4.

Alexander Clark and Rémi Eyraud. Polynomial identification in the limit of substitutable context-free languages. *J. Mach. Learn. Res.*, 8:1725–1745, 2007.

H. Comon, M. Dauchet, F. Jacquemard, R. Gilleron, C. Löding, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, 2007. release October, 12th 2007.

Sreerupa Das, C. Lee Giles, and Guo-Zheng Sun. Using prior knowledge in a {NNPDA} to learn context-free languages. In *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pages 65–72. Morgan Kaufmann, 1992.

Colin de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognit.*, 38(9):1332–1348, 2005.

Frank Drewes and Johanna Högberg. Query learning of regular tree languages: How to avoid dead states. *Theory Comput. Syst.*, 40(2):163–185, 2007. doi: 10.1007/s00224-005-1233-3. URL https://doi.org/10.1007/s00224-005-1233-3.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9 (8):1735–1780, 1997.

Malte Isberner. *Foundations of active automata learning: an algorithmic perspective.* PhD thesis, Technical University Dortmund, Germany, 2015. URL http://hdl.handle.net/2003/34282.

Malte Isberner, Falk Howar, and Bernhard Steffen. The TTT algorithm: A redundancy-free approach to active automata learning. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Proc. of RV 2014*, volume 8734 of *LNCS*, pages 307–322. Springer, 2014. doi: 10.1007/978-3-319-11164-3_26. URL https://doi.org/10.1007/978-3-319-11164-3_26.

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

Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. Minimization, learning, and conformance testing of boolean programs. In *Proc. of CONCUR 2006*, volume 4137 of *LNCS*, pages 203–217. Springer, 2006. doi: 10.1007/11817949_14. URL https://doi.org/10.1007/11817949_14.

P. Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. In Thomas Ball and Mooly Sagiv, editors, *Proc. of POPL 2011*, pages 283–294. ACM, 2011. doi: 10.1145/1926385.1926419. URL https://doi.org/10.1145/1926385.1926419.

Franz Mayr and Sergio Yovine. Regular inference on artificial neural networks. In *Proc. of CD-MAKE 2018*, volume 11015 of *LNCS*, pages 350–369. Springer, 2018.

Franz Mayr, Ramiro Visca, and Sergio Yovine. On-the-fly black-box probably approximately correct checking of recurrent neural networks. In *Proc. of CD-MAKE 2020*, volume 12279 of *LNCS*, pages 343–363. Springer, 2020. doi: 10.1007/978-3-030-57321-8_19. URL https://doi.org/10.1007/978-3-030-57321-8_19.

Kurt Mehlhorn. Pebbling moutain ranges and its application of dcfl-recognition. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *LNCS*, pages 422–435. Springer, 1980. doi: 10.1007/3-540-10003-2_89. URL https://doi.org/10.1007/3-540-10003-2_89.

Christian W. Omlin and C. Lee Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52, 1996. doi: 10.1016/0893-6080(95)00086-0.

Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Inf. Comput.*, 103(2):299–347, 1993.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (4th Edition).* Pearson, 2020.

Yasubumi Sakakibara. Efficient learning of context-free grammars from positive structural examples. *Inf. Comput.*, 97(1):23–60, 1992.

Guo-Zheng Sun, C. Lee Giles, and Hsing-Hen Chen. The neural network pushdown automaton: Architecture, dynamics and training. In *Adaptive Processing of Sequences and Data Structures, International Summer School on Neural Networks, "E.R. Caianiello", Vietri sul Mare, Salerno, Italy, September 6-13, 1997, Tutorial Lectures*, volume 1387 of *LNCS*, pages 296–345. Springer, 1997.

Sebastian Thrun. Extracting rules from artifical neural networks with distributed representations. In *Advances in Neural Information Processing Systems 7, [NIPS Conference, Denver, Colorado, USA, 1994]*, pages 505–512. MIT Press, 1994.

Frits W. Vaandrager. Model learning. *Commun. ACM*, 60(2):86–95, 2017. doi: 10.1145/2967606.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In *Proc. of ICML 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5244–5253. PMLR, 2018.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Learning deterministic weighted automata with queries and counterexamples. In *Proc. of NeurIPS 2019*, pages 8558–8569, 2019.

Daniel M. Yellin and Gail Weiss. Synthesizing context-free grammars from recurrent neural networks. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Proc. of TACAS 2021, Part I*, volume 12651 of *LNCS*, pages 351–369. Springer, 2021a. doi: 10.1007/978-3-030-72016-2_19. URL https://doi.org/10.1007/978-3-030-72016-2_19.

Daniel M. Yellin and Gail Weiss. Synthesizing context-free grammars from recurrent neural networks (extended version). *CoRR*, abs/2101.08200, 2021b. URL https://arxiv.org/abs/2101.08200.

Ryo Yoshinaka and Alexander Clark. Polynomial time learning of some multiple context-free languages with a minimally adequate teacher. In *Proc. of FG 2010*, volume 7395 of *LNCS*, pages 192–207. Springer, 2010. doi: 10.1007/978-3-642-32024-8_13. URL https://doi.org/10.1007/978-3-642-32024-8_13.

Appendix A. Visibly Pushdown Automata

Though we are principally interested in inferring grammars, we give here the definition of visibly pushdown automata, which also constitute a characterization of the class of VPLs.

Definition 6 *A* visibly pushdown automaton (VPA) *over* Σ *is a tuple* $\mathcal{A} = (Q, \mathcal{S}, \delta, \iota, F)$ *containing a finite set of control states* Q*, a nonempty finite set of stack symbols* $\mathcal{S}$*, an initial state* ι*, and a set of final states* $F \subseteq Q$*. Moreover,* $\delta = (\delta_{\mathsf{push}}, \delta_{\mathsf{pop}}, \delta_{\mathsf{int}})$ *is a collection of transition functions* $\delta_{\mathsf{push}} : Q \times \Sigma_{\mathsf{push}} \to \mathcal{P}(Q \times \mathcal{S})$*,* $\delta_{\mathsf{pop}} : Q \times \Sigma_{\mathsf{pop}} \times \mathcal{S} \to \mathcal{P}(Q)$*, and* $\delta_{\mathsf{int}} : Q \times \Sigma_{\mathsf{int}} \to \mathcal{P}(Q)$*. We call* $\mathcal{A}$ deterministic *if all transition functions map all arguments to singleton sets.*

A VPA recognizes a language $L(\mathcal{A}) \subseteq \Sigma^*$. Intuitively, it is the language of an infinite automaton whose states (we actually say configurations) are pairs (q, σ) where $q \in Q$ is the current control state and $\sigma \in \mathcal{S}^*$ is the current stack contents. With this, in the infinite automaton, we have a transition $(q, \sigma) \xrightarrow{a} (q', \sigma')$ if there is $A \in \mathcal{S}$ such that one of the following holds:

- $a \in \Sigma_{\mathsf{push}}$ and $(q', A) \in \delta_{\mathsf{push}}(q, a)$ and $\sigma' = \sigma \cdot A$
- $a \in \Sigma_{\mathsf{pop}}$ and $q' \in \delta_{\mathsf{pop}}(q, a, A)$ and $\sigma = \sigma' \cdot A$
- $a \in \Sigma_{\mathsf{int}}$ and $q' \in \delta_{\mathsf{int}}(q, a)$ and $\sigma' = \sigma$

We call (q, σ) a final configuration if $q \in F$ and $\sigma = \varepsilon$. Moreover, (ι, ε) is the only initial configuration. Finally, we define $L(\mathcal{A})$ to be the language recognized by this infinite automaton in the expected way.

Fact 6 (Alur and Madhusudan (2004, 2009)) *Let* $L \subseteq \Sigma^*$*. Then,* L *is a VPL over* Σ *iff there is a VPA* $\mathcal{A}$ *over* Σ *such that* $L(\mathcal{A}) = L$*.*

Appendix B. Proof of Theorem 5

Proof By Fact 5, we have to show that calling $\mathrm{TL}^*(\Gamma, \mathsf{MQ}_{\mathrm{tree}}, \mathsf{EQ}_{\mathrm{tree}})$ returns, in polynomial time, a DTA $\mathcal{B}$ such that $T(\mathcal{B}) = T(\hat{\mathcal{B}})$. We will show that $\mathsf{MQ}_{\mathrm{tree}}$ is sound for $T(\hat{\mathcal{B}})$ and $\mathsf{EQ}_{\mathrm{tree}}$ is counterexample- and equivalence-sound for $T(\hat{\mathcal{B}})$. By Fact 3, this implies that $\mathrm{TL}^*(\Gamma, \mathsf{MQ}_{\mathrm{tree}}, \mathsf{EQ}_{\mathrm{tree}})$ returns a DTA $\mathcal{B}$ such that $T(\mathcal{B}) = T(\hat{\mathcal{B}})$. The running time is polynomial since all additional operations in Algorithms 1 and 2 and can be performed in polynomial time (cf. Fact 2).

To show that $\mathsf{MQ}_{\mathrm{tree}}$ is sound for $T(\hat{\mathcal{B}})$, let $t \in \mathsf{Trees}(\Gamma)$. Assume $\mathsf{MQ}_{\mathrm{tree}}(t) = \mathsf{yes}$. By Algorithm 1, this implies $t \in T(\mathcal{B}_{\mathsf{parse}})$ and $\mathsf{MQ}_{\mathrm{vpl}}([\![t]\!]) = \mathsf{yes}$. As $\mathsf{MQ}_{\mathrm{vpl}}$ is sound for L, we have $[\![t]\!] \in L$. Since $T(\hat{\mathcal{B}}) = \langle\!\langle L \rangle\!\rangle$, we get $t \in T(\hat{\mathcal{B}})$. Conversely, assume $\mathsf{MQ}_{\mathrm{tree}}(t) = \mathsf{no}$. If $t \notin T(\mathcal{B}_{\mathsf{parse}})$, then $t \notin T(\hat{\mathcal{B}})$. So suppose $t \in T(\mathcal{B}_{\mathsf{parse}})$ and $\mathsf{MQ}_{\mathrm{vpl}}([\![t]\!]) = \mathsf{no}$. As $\mathsf{MQ}_{\mathrm{vpl}}$ is sound for L, we have $[\![t]\!] \notin L$, which implies $t \notin T(\hat{\mathcal{B}})$.

Let us show that $\mathsf{EQ}_{\mathrm{tree}}$ is counterexample-sound for $T(\hat{\mathcal{B}})$. Suppose $\mathcal{B} \in \mathsf{DTA}(\Gamma)$ and $t \in \mathsf{Trees}(\Gamma)$ such that $\mathsf{EQ}_{\mathrm{tree}}(\mathcal{B}) = t$. There are two cases. First, suppose $t \in T(\mathcal{B}) \setminus T(\mathcal{B}_{\mathsf{parse}})$. As $T(\hat{\mathcal{B}}) \subseteq T(\mathcal{B}_{\mathsf{parse}})$, we have $t \in T(\mathcal{B}) \setminus T(\hat{\mathcal{B}})$ and, hence, $t \in T(\mathcal{B}) \oplus T(\hat{\mathcal{B}})$. Second, assume

$T(\mathcal{B}) \subseteq T(\mathcal{B}_{\mathsf{parse}})$. As $\mathsf{EQ}_{\mathrm{vpl}}$ is counterexample-sound for L, this implies $[\![t]\!] \in L \oplus [\![T(\mathcal{B})]\!]$. Due to $T(\hat{\mathcal{B}}) = \langle\!\langle L \rangle\!\rangle$, we get $t \in T(\hat{\mathcal{B}}) \oplus T(\mathcal{B})$.

Finally, we show that $\mathsf{EQ}_{\mathrm{tree}}$ is equivalence-sound for $T(\hat{\mathcal{B}})$. Suppose $\mathcal{B} \in \mathsf{DTA}(\Gamma)$ such that $\mathsf{EQ}_{\mathrm{tree}}(\mathcal{B}) = \mathsf{yes}$. Then, $T(\mathcal{B}) \subseteq T(\mathcal{B}_{\mathsf{parse}})$ and $\mathsf{EQ}_{\mathrm{vpl}}(\mathcal{B}) = \mathsf{yes}$. As $\mathsf{EQ}_{\mathrm{vpl}}$ is equivalence-sound for L, we get $L = [\![T(\mathcal{B})]\!]$, which implies $T(\hat{\mathcal{B}}) = T(\mathcal{B})$. ∎