Natural Language Engineering **17** (2): 203–219. © Cambridge University Press 2011
doi:10.1017/S1351324911000064

A randomised inference algorithm for regular tree languages

JOHANNA HÖGBERG
Department of Computing Science , Umeå University
SE–901 87 Umeå, Sweden
email: johanna@cs.umu.se

(*Received 23 May 2010; revised 23 September 2010; accepted 22 October 2010*)

Abstract

We present a randomised inference algorithm for regular tree languages. The algorithm takes as input two disjoint finite nonempty sets of trees $\mathscr{P}$ and $\mathscr{N}$ and outputs a nondeterministic finite tree automaton that accepts every tree in $\mathscr{P}$ and rejects every tree in $\mathscr{N}$. The output automaton typically represents a nontrivial generalisation of the examples given in $\mathscr{P}$ and $\mathscr{N}$. To obtain compact output automata, we use a heuristics similar to bisimulation minimisation. The algorithm has time complexity of $O\left(n_{\mathscr{N}} \cdot n_{\mathscr{P}}^2\right)$, where $n_{\mathscr{N}}$ and $n_{\mathscr{P}}$ are the size of $\mathscr{N}$ and $\mathscr{P}$, respectively. Experiments are conducted on a prototype implementation, and the empirical results appear to second the theoretical results.

1 Introduction

We present an algorithmic solution to the consistency problem for regular tree languages (i.e. languages recognised by finite tree automata). This inference task was first introduced by Gold (1978) under the name *learning from given data*. The objective is to identify a language that is known to be realisable by a finite automaton, but the necessary number of states is not known. Only a finite amount of data is available, and as opposed to query learning, the algorithm has no choice about the data. It is therefore impossible to prove that a pair of trees belong to the same congruence class. This means that even under the added restriction that the representation of the unknown language should be minimal with respect to some normal form (e.g. deterministic or residual automata), a given instance may have several nonequivalent solutions.

The inference algorithm in Section 3 takes as input two disjoint finite sets of trees $\mathscr{P}$ and $\mathscr{N}$, and outputs a finite tree automaton (FTA) that is *consistent* with $\mathscr{P}$ and $\mathscr{N}$ in that it accepts every tree in $\mathscr{P}$ and rejects every tree in $\mathscr{N}$.[1] To obtain a compact representation, the algorithm uses a heuristics similar to bisimulation minimisation (Buchholz 2008). The algorithm is randomised, so executing it several times on the same input may yield nonequivalent automata. In this case, each

[1] The algorithm is thus supervised in the sense of Russell and Norvig (2003).

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press

output automaton typically represents a nontrivial generalisation of the examples in $\mathscr{P}$ and $\mathscr{N}$. Since the output automata are not restricted to a normal from, the algorithm does not convergence when the input sample is expanded and is therefore not useful for in-the-limit learning.

Owing to the close connection between context-free grammars and finite tree automata, the algorithm may be useful in natural language processing: it is well known that every context-free string language is the yield of a regular tree language. Finite automata usually offer a more succinct form of representation than corpora and are better suited for algorithmic processing. By using domain-specific knowledge to guide the nondeterministic choices made in the algorithm, it may be possible to find interesting generalisations from the input sample. Over-generalisation can be countered by supplying additional negative examples, either online or after termination and then resuming computation.

Whether or not the algorithm of Section 3 works with annotated data depends on the context. Seen as an inference algorithm for regular tree languages, it does not. Seen as an inference algorithm for context-free grammars, it does because from this point of view the input sets contain structurally annotated strings.

1.1 The consistency problem

The notion of learning from given data was introduced to provide insight into the in-the-limit learning model (Gold 1967). Because of this historical setting, many algorithmic solutions to the consistency problem are designed to infer a target language in the limit. Our discussion of related work will therefore begin with a brief introduction to the in-the-limit learning model.

An algorithm (henceforth, the learner) is to infer a formal representation of a language $\mathscr{L}$. It is known that $\mathscr{L}$ is a subset of a countable domain $\mathscr{U}$ and belongs to a class of languages $\mathscr{C}$. A problem instance builds upon an enumeration $\mathscr{E}$ of $\mathscr{U}$. The learner is revealed membership information about the elements in $\mathscr{U}$ in the order dictated by $\mathscr{E}$, and must, after processing each item in the enumeration, guess the identity of $\mathscr{L}$. The class $\mathscr{C}$ is *learnable in the limit (from positive and negative data)* if there is a learner with the following property. For every $\mathscr{L}$ in $\mathscr{C}$ and enumeration $\mathscr{E}$ of $\mathscr{U}$, there is a finite index i such that after processing the first i elements with respect to $\mathscr{E}$, the learner will always make the same guess, and it will be correct.

An equivalent definition can be made in terms of the consistency problem. The class $\mathscr{C}$ is learnable in the limit if there is a consistency algorithm $\mathfrak{L}$ and a mapping $f_{\mathfrak{L}} : \mathscr{C} \to pow(\mathscr{U})$ such that for every $\mathscr{L} \in \mathscr{C}$ and finite disjoint sets $\mathscr{P}, \mathscr{N} \subseteq \mathscr{U}$ with $\mathscr{P} \subseteq \mathscr{L}$, $\mathscr{N} \cap \mathscr{L} = \emptyset$, and $f_{\mathfrak{L}}(\mathscr{L}) \subseteq \mathscr{P} \cup \mathscr{N}$, the algorithm $\mathfrak{L}$ returns a representation of $\mathscr{L}$ on input $\mathscr{P}$ and $\mathscr{N}$. In this case, $f_{\mathfrak{L}}(\mathscr{L})$ is a *characteristic set* for $\mathscr{L}$.

It is known that unless P = NP there is no polynomial-time algorithm that finds a minimal deterministic finite automaton (DFA) A that is consistent with $\mathscr{P}$ and $\mathscr{N}$. The problem cannot even be efficiently approximated because computing a DFA that is only polynomially larger than A is also NP-hard (Pitt and Warmuth 1989). The difficulty is that to synthesise A, missing data must be guessed, and the hypothesis

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press

space is exponential in the size of $\mathscr{P} \cup \mathscr{N}$. On the other hand, computing an arbitrary consistent DFA takes only linear time, but this approach yields unnecessarily large automata and does not allow for identification in the limit.

The above observation is the basis for the *timid state characterisation algorithm* that infers regular languages in the limit with polynomial time and data (Gold 1978). To compute a conjecture for $\mathscr{P}$ and $\mathscr{N}$, Gold's algorithm searches for subsets $\mathscr{P}_0 \subseteq \mathscr{P}$ and $\mathscr{N}_0 \subseteq \mathscr{N}$ from which a DFA A can be synthesised without having to guess missing data. If A is consistent with $\mathscr{P}$ and $\mathscr{N}$, then it becomes the next conjecture. If no such A can be obtained, then a prefix tree acceptor for $\mathscr{P}$ is used as a dummy conjecture. In essence, the learner waits for a characteristic set for the target language and disregards data which it cannot use easily (Gold 1978).

Gold's work inspired the search for polynomial-time consistency algorithms that produce small automata also when conditions are less than perfect, i.e. when the input data does not contain a characteristic set. Early attempts inferred deterministic automata by building a prefix tree acceptor matching the positive examples, and then merging as many states as possible while still respecting the negative examples. The first algorithm of this kind was given by Trakhtenbrot and Barzdin (1973). The state-merge technique for DFA was later refined to obtain the regular positive and negative induction algorithm (Oncina and Garcia 1991), which was further developed into the control strategy blue-fringe (Lang, Pearlmutter and Price 1998). An extension of the technique to tree automata has been given by Garcia and Oncina (1993).

The question naturally arose whether the ideas used for inference of deterministic automata could be generalised to solve the consistency problem also for nondeterministic automata. A first step in this direction was made possible by the introduction of residual finite state automata by Denis, Lemay and Terlutte (2001). A residual finite state automata is a restricted type of nondeterministic finite automata that shares many nice algorithmic properties with DFA. For example, every regular language is recognised by a canonical residual finite state automata which has a minimal number of states and a maximal number of transitions. Denis, Lemay and Terlutte (2000) have also presented a state-merge algorithm that infers residual finite state automata in the limit. The unambiguous finite automata constitute another class of mildly nondeterministic finite automata that was first studied by Schmidt (1978). The class of languages recognisable by unambiguous finite automata can also be learnt from given data, again by merging states in a prefix automaton (Coste and Fredouille 2003).

The consistency problem for unrestricted (i.e. fully nondeterministic) finite automata is treated by de Parga, García and Ruiz (2006). The authors present a family of in-the-limit algorithms that use maximal automata and state merging to infer regular languages. The work is continued by García *et al.* (2008) who present two variations on the stochastic order-independent learning algorithm (OIL): Minimal and Voting OIL. Both variations use the fact that OIL returns several output automata for a given input, but whereas Minimal OIL returns the smallest automaton, Voting OIL keeps all of them and uses a voting procedure to classify test samples. The algorithm presented in Section 3 differs from the OIL family in that it is based on partition

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press

refinement rather than state merging, and thus intuitively searches the hypothesis space in the opposite direction.

1.2 Learning tree languages and context-free grammars

Several learning algorithms for finite tree automata rely on access to a *minimal adequate teacher*, as introduced by Angluin (1987). In this model, the algorithm is informed about the target language $\mathscr{L}$ by querying a minimal adequate teacher oracle. The oracle is capable of answering *membership* and *equivalence* queries. Membership queries are of the form *Is the tree t in* $\mathscr{L}$*?* and are answered with a yes or no. Equivalence queries are phrased *Does the finite automaton A recognise* $\mathscr{L}$*?* and are answered with an element in the symmetric difference of $\mathscr{L}(A)$ and $\mathscr{L}$, or with an acknowledgement that $\mathscr{L}$ has been correctly identified if no such tree exists. The learning algorithm can thus use the oracle to sharpen its hypothesis until the right answer has been found. For regular tree languages, the minimal deterministic FTA that recognises $\mathscr{L}$ can be inferred in polynomial time (Drewes and Högberg 2007). There are also algorithms that infer deterministic weighted tree automata (Drewes and Vogler 2007; Maletti 2007). A minimal adequate teacher algorithm for nondeterministic weighted tree automata over fields has been presented by Habrard and Oncina (2006). The algorithm uses the nice and somewhat surprising fact that weighted automata over fields have a Myhill-Nerode-like theorem of their own. For further reading, we recommend the survey of minimal adequate teacher algorithms that learn tree automata by Drewes (2009).

A related learning model, also by Angluin (1981), is *learning from positive examples and membership queries*. In this model, the oracle can only answer membership queries, but in addition to the oracle, the learning algorithm has access to a finite set $\mathscr{P} \subseteq \mathscr{L}$. In Besombes and Marion (2007), the authors present a learning algorithm for regular tree languages that produces the unique minimal deterministic FTA $A_{\mathscr{L}}$ that recognises $\mathscr{L}$ in polynomial time, provided that $\mathscr{P}$ is a *representative set* for $\mathscr{L}$. A subset $\mathscr{P}$ of $\mathscr{L}$ is a representative set for $\mathscr{L}$ if, after running $A_{\mathscr{L}}$ on every tree in $\mathscr{P}$, every transition in $A_{\mathscr{L}}$ has been applied at least once.

In syntax-based computational linguistics, human language is processed based on its parse structure with respect to some formal grammar (Knight and May 2009). An important learning task within this subfield is to derive a small context-free grammar that is consistent with a treebank, i.e. a set of labelled phrase-structure trees. This can be done through *rule extraction*, where the trees are divided into fractions representing productions in the output grammar. As shown by Bod (1999), rule extraction can be used to infer stochastic grammars from treebanks.

A different approach is suggested by Sakakibara and Muramatsu (2000) who present a genetic algorithm for inferring context-free grammars from (partially structured) positive and negative samples. The algorithm organises a maximal set of nonterminals in a table similar to that used in chart parsing and then uses a genetic-algorithm-based technique to merge nonterminals, so as to obtain a small output grammar.

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press

The automatic distillation of structure algorithm derives a language from a positive sample of unannotated sentences by searching for sequential patterns in the data. Whereas automatic distillation of structure uses statistical analysis to structure and generalise from the input sample, our algorithm (in the context of string languages) additionally requires structural annotation and a negative sample to compute a representation that may generalise more strongly from the given input (Edelman *et al.* 2005).

A polynomial time in-the-limit algorithm that infers *substitutable* context-free languages from positive examples has been presented by Clark and Eyraud (2005). A context-free string language $\mathscr{L}$ is substitutable if the existence of strings u, v, l and r such that $lvr, lur \in \mathscr{L}$ implies that $l'ur' \in \mathscr{L} \iff l'vr' \in \mathscr{L}$ for all strings l' and r'. This corresponds to Harris' (1951) principle that if a pair of word sequences constitute the same grammatical category, then they can be interchanged in any sentence without altering the grammatical correctness of the sentence. Harris' principle is also the foundation for alignment-based learning, an inference framework for unsupervised learning. Alignment-based learning algorithms typically operate in two phases. In the first phase, sample sentences are aligned to generate a set of hypotheses, each suggesting that a pair of sub-sentences constitute the same grammatical category. In the second phase, the most probable combination of hypothesis is selected through an expectation-maximisation search (van Zaanen 2001; Geertzen and van Zaanen 2004).

The algorithm Emile learns a *shallow* context-free language in the limit from set of positive examples. A context-free language is shallow if it is generated by a grammar G such that every production in G is used in to generate at least one sentence of a length that is logarithmic in the size of G. Emile works by dividing and recombining the sample sentences, so as to discover syntactical categories. For efficiency reasons, some presentations of the algorithm include a membership oracle that can answer whether a given combination of sentence fragments is grammatically correct (Adriaans 1999; Adriaans, Trautwein and Vervoort 2000).

2 Preliminaries

This section reviews common notions and notations from tree automata theory. For an introduction to the field, see e.g. the survey by Comon *et al.* (1997).

2.1 Sets and numbers

The set of real numbers is denoted by $\mathbb{R}$ and the set of natural numbers (including 0) by $\mathbb{N}$. The subset $\{1, 2, \ldots, n\}$ of $\mathbb{N}$ is abbreviated by $[n]$. Note that $[0] = \emptyset$. Let S be a set. The power set of S is denoted as $pow(S)$, and we write S^2 as a short form for $S \times S$. The set of all finite sequences (or *strings*) over S, including the empty sequence ε, is S^*. As usual, we denote the concatenation of strings u and v by the juxtaposition uv. The cardinality of the set S and the length of the string $w \in S^*$ are denoted by $|S|$ and $|w|$, respectively.

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press

2.2 Trees

A *ranked alphabet* is a finite nonempty set $\Sigma = \bigcup_{k\in\mathbb{N}} \Sigma_{(k)}$ of symbols which is partitioned into pairwise disjoint subsets $\Sigma_{(k)}$. The symbols in $\Sigma_{(k)}$ are said to have rank k. We will write $f \in \Sigma_{(k)}$ as $f^{(k)}$ if the rank cannot easily be determined from the context. The set T_Σ of all trees over Σ consists of all mappings $t : \mathit{nodes}(t) \to \Sigma$ (called trees) with the following properties:

- The set $\mathit{nodes}(t)$ is a finite prefix-closed subset of $\mathbb{N}^*$.
- For every $v \in \mathit{nodes}(t)$, if $t(v) \in \Sigma_{(k)}$ then $\{i \in \mathbb{N} \mid vi \in \mathit{nodes}(t)\} = [k]$.

The set $\mathit{nodes}(t)$ is called the *nodes* of t. The size of t, denoted $|t|$, is $|\mathit{nodes}(t)|$.

For a tree $t \in T_\Sigma$ and a node $v \in \mathit{nodes}(t)$, the *subtree of t rooted at v* is denoted by t/v. It is defined by $\mathit{nodes}(t/v) = \{v' \in \mathbb{N}^* \mid vv' \in \mathit{nodes}(t)\}$ and, for every $v' \in \mathit{nodes}(t/v)$, $(t/v)(v') = t(vv')$. The *subtrees* of t, denoted $\mathit{subtrees}(t)$, is the set of trees $\{t/v \mid v \in \mathit{nodes}(t)\}$. When S is a set of trees, $\mathit{subtrees}(S)$ denotes $\cup_{t\in S}\mathit{subtrees}(t)$. A *leaf* of t is a node v such that $t(v)$ is of rank 0. We shall denote a tree t as $f[t_1,\ldots,t_k]$ if $t(\varepsilon) = f^{(k)}$ and $t/i = t_i$ for $1 \le i \le k$. Here, we say that the trees $t_1,\ldots,t_k$ are *direct subtrees* of t. In the special case where $k = 0$ (i.e. $\mathit{nodes}(t) = \{\varepsilon\}$), the brackets may be omitted, thus denoting t as f. A subset of T_Σ is called a *tree language*.

We also need an operation that lets us replace a direct subtree of a tree with another tree. Given a pair of trees $t = f[t_1,\ldots,t_k]$, $t' \in T_\Sigma$ and an index $i \in [k]$, we denote by $t[\![i \leftarrow t']\!]$ the tree $f[t_1,\ldots,t_{i-1},t',t_{i+1},\ldots,t_k]$.

2.3 Finite automata

A FTA is a tuple $A = (Q,\Sigma,\delta,F)$ where Q is a finite nonempty set of *states*, Σ is a ranked alphabet of *input symbols*, $\delta = (\delta_f)_{f\in\Sigma}$ is a Σ-indexed family of mappings $\delta_{f^{(k)}} : Q^k \to \mathit{pow}(Q)$, and $F \subseteq Q$ is a set of *final* states. The FTA A is *deterministic* if, for every symbol $f \in \Sigma_{(k)}$ and sequence of states $q_1,\ldots,q_k \in Q$, it holds that $|\delta_f(q_1,\ldots,q_k)| \le 1$. As usual, we extend δ_f to a mapping $\hat{\delta}_f : \mathit{pow}(Q)^k \to \mathit{pow}(Q)$ by letting $\hat{\delta}_f(Q_1,\ldots,Q_k) = \cup_{q_1\in Q_1,\ldots q_k\in Q_k}\delta_f(q_1,\ldots,q_k)$. We henceforth identify δ_f with its extension $\hat{\delta}_f$. The size of A, written $|A|$, is the sum of $|Q|$ and the size of a sparse representation of δ.

A *run* of the FTA A on a tree $t \in T_\Sigma$ is a mapping $r : \mathit{nodes}(t) \to Q$, such that $r(v)$ is in the set $\delta_{t(v)}(r(v1)\cdots r(vk))$ for every node $v \in \mathit{nodes}(t)$. The set of all runs of A on t is denoted as $\mathit{runs}_A(t)$. A run is *accepting* if $r(\varepsilon) \in F$. The automaton A *accepts* a tree t if there is an accepting run of A on t. The language *recognised* by A, denoted $\mathscr{L}(A)$, is the set of all trees that A accepts. A state $q \in Q$ is *reachable* if there is a tree $t \in T_\Sigma$, a run $r \in \mathit{runs}_A(t)$ and a node $v \in \mathit{nodes}(t)$, such that $r(v) = q$. If, for some such r, $r(\varepsilon) \in F$ then q is also *live*. A tree language L is a *regular tree language* if there is an FTA A such that $L = \mathscr{L}(A)$.

2.4 Partitions

Let S be a set, and let $\mathscr{R}$ and $\mathscr{R}'$ be equivalence relations on S. We say that $\mathscr{R}$ is a *refinement* of $\mathscr{R}'$ if $\mathscr{R} \subseteq \mathscr{R}'$, and that $\mathscr{R}$ is a *proper* refinement of $\mathscr{R}'$ if $\mathscr{R} \subset \mathscr{R}'$.

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press

The *equivalence class* (or *block*) of an element $s \in S$ with respect to $\mathscr{R}$ is the set $[s]_{\mathscr{R}} = \{s' \mid (s, s') \in \mathscr{R}\}$. It should be clear that $[s]_{\mathscr{R}}$ and $[s']_{\mathscr{R}}$ are equal if and only if s and s' are in relation $\mathscr{R}$, and disjoint otherwise, so $\mathscr{R}$ induces a partition $\left(S/\mathscr{R}\right) = \{[s]_{\mathscr{R}} \mid s \in S\}$ of S.

3 The inference algorithm

In this section, we present an algorithm that takes as input a pair of disjoint finite nonempty sets of trees $\mathscr{P}$ and $\mathscr{N}$ and outputs an FTA A such that $\mathscr{P} \subseteq \mathscr{L}(A) \subseteq (T_\Sigma \setminus \mathscr{N})$. During its execution, the algorithm iteratively refines an equivalence relation $\mathscr{R}$ on the set $\mathit{subtrees}(\mathscr{P})$. From each intermediate relation, a hypothesis automaton $A_{\mathscr{R},\mathscr{P}}$ is compiled using the construction in Definition 3.1.

Definition 3.1 (Synthesised FTA) *Let $\mathscr{P}$ be a finite subset of T_Σ and $\mathscr{R}$ be an equivalence relation on $\mathit{subtrees}(\mathscr{P})$ that refines $\{(s_1, s_2) \mid s_1 \in \mathscr{P} \iff s_2 \in \mathscr{P}\}$. The FTA* synthesised *from $\mathscr{R}$ and $\mathscr{P}$ is $A_{\mathscr{R},\mathscr{P}} = (Q, \Sigma, \delta, F)$, where $Q = \left(\mathit{subtrees}(\mathscr{P})/\mathscr{R}\right)$, $F = \{q \in Q \mid q \subseteq \mathscr{P}\}$ and*

$$\delta_f(q_1, \ldots, q_k) = \{q \in Q \mid \exists t_i \in q_i \text{ for every } i \in [k] \text{ s.t. } f[t_1, \ldots, t_k] \in q\},$$

for every symbol $f^{(k)} \in \Sigma$ and sequence of states $q_1, \ldots, q_k \in Q$. □

By the subtree-closedness of $\mathit{subtrees}(\mathscr{P})$, the state $[t]_{\mathscr{R}}$ is reachable on t, for every $t \in \mathit{subtrees}(\mathscr{P})$. From this property, which is easy to prove by induction, it follows that $A_{\mathscr{R},\mathscr{P}}$ is consistent with $\mathscr{P}$.

Lemma 3.2 *Let $\mathscr{R}$, $\mathscr{P}$ and $A_{\mathscr{R},\mathscr{P}}$ be as in Definition 3.1. For every $t \in \mathit{subtrees}(\mathscr{P})$, there is a run $r \in \mathit{runs}_{A_{\mathscr{R},\mathscr{P}}}(t)$ such that $r(\varepsilon) = [t]_{\mathscr{R}}$.*

Corollary 3.3 $\mathscr{P} \subseteq \mathscr{L}(A_{\mathscr{R},\mathscr{P}})$.

We note that if $\mathscr{R}$ is the identity relation, then $A_{\mathscr{R},\mathscr{P}}$ is a *prefix-tree acceptor* for $\mathscr{P}$, i.e. a loop-free deterministic FTA with $|\mathit{subtrees}(\mathscr{P})|$ states that accepts precisely the trees in $\mathscr{P}$. It is clear that a coarser equivalence relation will yield a smaller output automaton, but usually at the cost of a recognising a larger language.

This trade-off between size and precision makes an iterative approach appealing, that also makes use of a set $\mathscr{N}$ of negative examples. The idea is to start with a very coarse relation $\mathscr{R}$ that only distinguishes between trees in $\mathscr{P}$ and trees in $\mathit{subtrees}(\mathscr{P}) \setminus \mathscr{P}$ (as in Definition 3.1). If $A_{\mathscr{R},\mathscr{P}}$ accepts a tree s in $\mathscr{N}$, then the relation $\mathscr{R}$ is refined to reduce the number of accepting runs of $A_{\mathscr{R},\mathscr{P}}$ on s. This process is repeated until $A_{\mathscr{R},\mathscr{P}}$ rejects every tree in $\mathscr{N}$.

Assume that $A_{\mathscr{R},\mathscr{P}}$ has an accepting run r on $s \in \mathscr{N}$. To find an appropriate refinement of $\mathscr{R}$, we search r for a transition that is only available in $A_{\mathscr{R},\mathscr{P}}$ because $\mathscr{R}$ has placed a pair of distinct trees in the same equivalence class. A transition of this type is called a *generalisation*. The easiest way to find a generalisation is to traverse s bottom-up, selecting the first node v in $\mathit{nodes}(s)$ such that $s/v = f^{(k)}[t_1, \ldots, t_k]$

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press

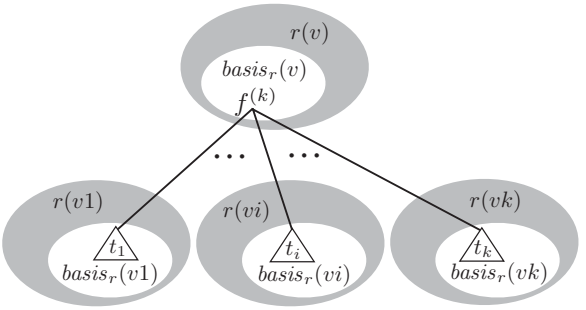


Fig. 1. For every $v \in nodes(s)$ with $s(v) = f^{(k)}$, the set $basis_r(v) \subseteq r(v)$ contains the trees that caused $r(v)$ to be in $\delta_f(r(v1), \ldots, r(vk))$.

is not in $r(v)$, but t_i is in $r(vi)$ for every $i \in [k]$. Such a v must exist, because otherwise $s \in r(\varepsilon) \subseteq \mathscr{P}$ which contradicts the disjointness of $\mathscr{P}$ and $\mathscr{N}$.

Since $A_{\mathscr{R},\mathscr{P}}$ could assign $r(v)$ to v, we have $r(v) \in \delta_f(r(v1), \ldots, r(vk))$. This implies that there are trees $t'_i \in r(vi)$, $i \in [k]$, such that $f[t'_1, \ldots, t'_k] \in r(v)$. To avoid that the transition from $r(v1), \ldots, r(vk)$ to $r(v)$ on f is used to accept s, the relation $\mathscr{R}$ is refined as follows.

We take the tree $f[t'_1, \ldots, t'_k]$ and start to replace the subtrees $t'_1, \ldots, t'_k$ with their counterparts in $t_1, \ldots, t_k$. The trees are replaced one by one, for as long as the resulting tree remains in $r(v)$. We will not manage to replace all of them because then we would eventually have $f[t_1, \ldots, t_k] \in r(v)$, which we know is not true. Let $s'' = f[t''_1, \ldots, t''_k]$ be result of this replacement process, and let $i \in [k]$ be the index of one of the subtrees that were never replaced. We now divide $r(vi)$ into two blocks, $B_1 = \{u \in r(vi) \mid s''[\![i \leftarrow u]\!] \in r(v)\}$ and $B_2 = \{u \in r(vi) \mid s''[\![i \leftarrow u]\!] \notin r(v)\}$. Clearly, $t'_i \in B_1$ whereas $t_i \in B_2$.

The run r typically contains several generalisations when $\mathscr{R}$ is coarse. To identify the nodes $gen(r) \subseteq nodes(s)$ at which they occur, we define the mapping $basis_r : nodes(s) \to subtrees(\mathscr{P})$ through structural induction on s, starting at the leaves. Let $v \in nodes(s)$ with $s/v = f[t_1, \ldots, t_k]$. Intuitively, $basis_r(v)$ contains those trees that justify a transition from $basis_r(v1), \ldots, basis_r(vk)$ to $r(v)$ on the symbol f (see Figure 1 for an illustration). If no such tree exists, then v uses a generalisation and should be added to $gen(v)$. To find further generalisations, we let $basis_r(v) = r(v)$ and continue upwards. In the definition of gen and $basis_r$ below, we use the mapping $basis'_r$ as a technical convenience.

Definition 3.4 (Basis and generalisation) *Let $\mathscr{R}$, $\mathscr{P}$ and $A_{\mathscr{R},\mathscr{P}}$ be as in Definition 3.1. Recall that the set of states of $A_{\mathscr{R},\mathscr{P}}$ is $\big(subtrees(\mathscr{P})/\mathscr{R}\big)$. In addition, let $t \in T_\Sigma$ and $r \in runs_{A_{\mathscr{R},\mathscr{P}}}(t)$. The mappings $basis_r, basis'_r : nodes(t) \to pow(T_\Sigma)$ are inductively given, for every $v \in nodes(t)$ with $t(v) = f \in \Sigma_{(k)}$, by*

$$basis'_r(v) = \{f[t_1, \ldots, t_k] \in r(v) \mid t_i \in basis_r(vi)\}$$

and

$$basis_r(v) = \begin{cases} basis'_r(v) & \text{if } basis'_r(v) \neq \emptyset, \text{ and} \\ r(v) & \text{otherwise.} \end{cases}$$

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press

Algorithm 1 Derive an FTA from given data

Require: $\mathscr{P}$ and $\mathscr{N}$ are disjoint finite nonempty sets of trees.
Ensure: $A_{\mathscr{R},\mathscr{P}}$ is an FTA such that $\mathscr{P} \subseteq \mathscr{L}(A_{\mathscr{R},\mathscr{P}}) \subseteq (T_\Sigma \setminus \mathscr{N})$

1: $\mathscr{R} \leftarrow \mathscr{P}^2 \cup (subtrees(\mathscr{P}) \setminus \mathscr{P})^2$
2: **while** $\exists s \in \mathscr{L}(A_{\mathscr{R},\mathscr{P}}) \cap \mathscr{N}$ **do**
3: select $r \in runs_{A_{\mathscr{R},\mathscr{P}}}(s)$ s.t. $r(\varepsilon) \in \mathscr{P}$
4: select $v \in gen(r)$
5: select $t = f[t_1, \ldots, t_k] \in r(v)$ s.t. $t_i \in r(vi)$, for every $i \in [k]$
6: **while** $\exists i \in [k]$ and $t'_i \in basis_r(vi)$ s.t. $t_i \notin basis_r(vi)$ and $t[\![i \leftarrow t'_i]\!] \in r(v)$ **do**
7: $t \leftarrow t[\![i \leftarrow t'_i]\!]$
8: select $i \in [k]$ s.t. $t_i \notin basis_r(vi)$
9: $\mathscr{R} \leftarrow (\mathscr{R} \setminus r(vi)^2) \cup B_1^2 \cup B_2^2$ where
10: $B_1 = \{t'_i \in r(vi) \mid t[\![i \leftarrow t'_i]\!] \in r(v)\}$
11: $B_2 = r(vi) \setminus B_1$
12: **return** $A_{\mathscr{R},\mathscr{P}}$

The set of generalisations *in the run r, denoted gen(r), is the set of nodes* $\{v \in nodes(t) \mid basis'_r(v) = \emptyset\}$.

A formal presentation of the inference algorithm is given in Algorithm 1. After constructing an initial partition of $subtrees(\mathscr{P})$, the algorithm enters into the main loop. With each new iteration of the loop, the algorithm synthesises an FTA $A_{\mathscr{R},\mathscr{P}}$ from $\mathscr{R}$ and $\mathscr{P}$. If $A_{\mathscr{R},\mathscr{P}}$ has an accepting run on a tree in the negative corpus $\mathscr{N}$, then the algorithm refines $\mathscr{R}$ to cancel this particular run. It does this by identifying a block that is used in a generalisation and then splits the block using a local condition on the transition structure, similar to that of forward bisimulation (Buchholz 2008). This process continues until $A_{\mathscr{R},\mathscr{P}}$ respects $\mathscr{N}$. In the case when $\mathscr{P}$ is subtree-closed, the initial relation $\mathscr{R}$ has a single equivalence class and the algorithm terminates after one iteration.

Before we prove that Algorithm 1 is correct, let us step through a run of the algorithm, so as to complement the theoretical presentation with a concrete example.

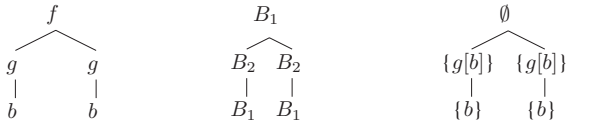
Example 3.5 *Suppose that Algorithm 1 is invoked with input* $\mathscr{P}, \mathscr{N} \subset T_\Sigma$, *where*

$$\begin{aligned} \Sigma &= \{\, a^{(0)},\ b^{(0)},\ g^{(1)},\quad f^{(2)} \,\}, \\ \mathscr{P} &= \{\, b,\ f[a,a],\ f[g[b],a],\quad f[g[b],g[f[b,a]]] \,\} \text{ and} \\ \mathscr{N} &= \{\, g[a],\ f[g[b],g[b]],\quad f[f[f[f[b,a],a],a],a] \,\}. \end{aligned}$$

With this input, the domain of $\mathscr{R}$ *is* $subtrees(\mathscr{P}) = \mathscr{P} \cup \{a,\ g[b],\ f[b,a],\ g[f[b,a]]\}$. *After the initialisation step at line 1, the relation* $\mathscr{R}$ *has the equivalence classes* $B_1 = \mathscr{P}$ *and* $B_2 = subtrees(\mathscr{P}) \setminus \mathscr{P}$.

Upon entering the main loop at line 2, the algorithm synthesises an FTA from the partition induced by $\mathscr{R}$ *on its domain (the construction is given in Definition 3.1). The result is the automaton* $A_1 = (\{B_1, B_2\}, \Sigma, \delta, \{B_1\})$ *with the transition mapping listed*

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press



(a) The tree s (b) The accepting run r (c) The mapping $basis'_r$

Fig. 2. The automaton A_1 accepts the tree $s \in \mathcal{N}$ through the run r. To cancel r, Algorithm 1 computes the mapping $basis'_r$.

below.

$$\begin{array}{llllll} \delta_a() & = & \{B_2\} & \delta_g(B_1) & = & \{B_2\} \quad \delta_f(B_1, B_2) = \{B_2\} \\ \delta_b() & = & \{B_1\} & \delta_g(B_2) & = & \{B_2\} \quad \delta_f(B_2, B_2) = \{B_1\} \end{array}$$

When applied to the trees in $\mathcal{N}$, the automaton A_1 correctly rejects $g[a]$ but accepts the tree $s = f[g[b], g[b]]$ (drawn in Figure 2a). The unique accepting run r of A_1 on s is given in Figure 2(b). To find a way to cancel r, the algorithm computes the mappings $basis'_r$ and $basis_r$. The mapping $basis'_r$ is shown in Figure 2(c); the mapping $basis_r$ is identical, except that for every node $u \in nodes(s)$ such that $basis'_r(u) = \emptyset$, we have $basis_r(u) = r(u)$. In this particular case, the only difference between $basis'_r$ and $basis_r$ is that $basis'_r(\varepsilon) = \emptyset$, whereas $basis_r(\varepsilon) = r(\varepsilon) = B_1$.

As the reader may recall, $gen(r)$ contains every node $u \in nodes(s)$ such that $basis'_r(u) = \emptyset$. According to the computations in Figure 2(c), this means that $gen(r) = \{\varepsilon\}$. Hence, Algorithm 1 can only select $v = \varepsilon$ at line 4. At line 5, the tree t can be selected in more than one way. To continue, we assume that algorithm nondeterministically chooses $t = f[a, a] \in B_1$.

While traversing the loop at lines 6–7, the algorithm nondeterministically decides to replace the left subtree (i.e. $t/1 = a$) of t. It can only replace a with $g[b]$ because $g[b]$ is the only element in $basis_r(1)$. After this operation, the right subtree must be kept as it is because if we replace it with the unique tree $g[b] \in basis_r(2)$, then the resulting tree $f[g[b], g[b]]$ is no longer in B_1. The inner loop consequently exits, and at line 8 the algorithm must choose $i = 2$.

By splitting the block $r(vi) = r(\varepsilon 2) = r(2) = B_2$ of $\mathcal{R}$ with respect to $t = f[g[b], a]$ and $i = 2$ (see lines 9–11 of Algorithm 1), a finer relation is obtained. This relation has the equivalence classes $B_{21} = \{a,\ g[f[b, a]]\}$, $B_{22} = \{g[b],\ f[b, a]\}$ and

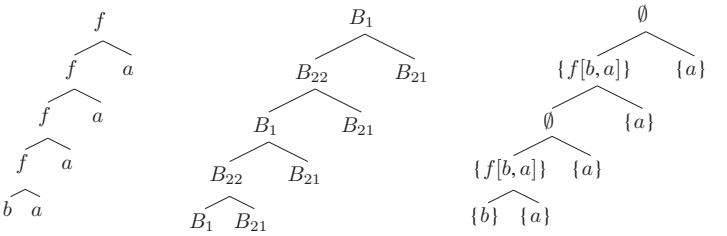
$$B_1 = \{b,\ f[a, a],\ f[g[b], a],\ f[g[b], g[f[b, a]]]\}.$$

The updated FTA $A_2 = (\{B_1, B_{21}, B_{22}\}, \Sigma, \delta, \{B_1\})$ is such that

$$\begin{array}{llllll} \delta_a() & = & \{B_{21}\} & \delta_g(B_1) & = & \{B_{22}\} \quad \delta_f(B_1, B_{21}) = \{B_{22}\} \\ \delta_b() & = & \{B_1\} & \delta_g(B_{22}) & = & \{B_{21}\} \quad \delta_f(B_{21}, B_{21}) = \{B_1\} \\ & & & & & \quad \delta_f(B_{22}, B_{21}) = \{B_1\} \end{array}$$

After the first completed iteration of the main loop, the algorithm returns to the head of the loop at line 2, were it runs A_2 on the trees in $\mathcal{N}$. This time, the automaton $A_{\mathcal{R},\mathcal{P}}$ rejects both $g[a]$ and $f[g[b], g[b]]$, but it accepts the tree $f[f[f[f[b, a], a], a], a] \in \mathcal{N}$ (drawn in Figure 3a) through the accepting run r shown in Figure 3(b). Again, the algorithm computes $basis_r$ yielding the function in Figure 3(c). At line 4, the algorithm can either choose $v = \varepsilon$ or $v = 11$ from $gen(r)$. We assume that it chooses $v = 11$ and

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press



(a) The tree s (b) The run r (c) The mapping $basis'_r$

Fig. 3. The automaton A_2 accepts the tree $s \in \mathcal{N}$ through the run r. To cancel r, Algorithm 1 computes the mappings $basis'_r$ and $basis_r$.

that it at line 5 selects $t = f[g[b], a]$. *This is a permissible option since* $t/1 = g[b] \in r(v1) = r(111) = B_{22}$ *and* $t/2 = a \in r(v2) = r(112) = B_{21}$.

Let us see what these choices mean for the WHILE *loop at line 6. It should be clear that the right subtree of* t *cannot be replaced because it is already in* $basis_r(v2)$. *The left subtree of* t *cannot be replaced either because* $f[g[b], a][\![1 \leftarrow f[b, a]]\!] = f[f[b, a], a]$ *is not in* $r(v) = r(11) = B_1$. *The* WHILE *loop thus exits immediately and the algorithm continues to split the block* $r(vi) = r(111) = B_{22}$ *of* $\mathscr{R}$ *with respect to* $t = f[g[b], a]$ *and* $i = 1$. *This refinement produces a relation with the equivalence classes* $B_{21} = \{a, g[f[b, a]]\}$, $B_{221} = \{g[b]\}$, $B_{222} = \{f[b, a]\}$ *and*

$$B_1 = \{\, b,\ f[a, a],\ f[g[b], a],\ f[g[b], g[f[b, a]]] \,\}.$$

Having completed the second iteration of the main loop, the algorithm re-computes the automaton $A_{\mathscr{R},\mathscr{P}}$. *The result is the automaton* $A_3 = (Q, \Sigma, \delta, F)$, *where* $Q = \{B_1, B_{21}, B_{221}, B_{222}\}$, $F = \{B_1\}$ *and* δ *is given by*

$$\begin{array}{llllll} \delta_a() = \{B_{21}\} & \delta_g(B_1) = \{B_{221}\} & \delta_f(B_1, B_{21}) = \{B_{222}\}\ \ddagger \\ \delta_b() = \{B_1\} & \delta_g(B_{222}) = \{B_{21}\}\ \dagger & \delta_f(B_{21}, B_{21}) = \{B_1\} \\ & & \delta_f(B_{221}, B_{21}) = \{B_1\} \end{array}$$

Returning once more to line 2 and the head of the main loop, the algorithm runs A_3 *on every tree in* $\mathcal{N}$. *As* A_3 *rejects all of them, the main loop is exited,* A_3 *is returned, and the algorithm terminates.*

The final automaton A_3 *generalises from* $\mathscr{P}$ *to an infinite language. This follows from the fact that* A_3 *contains a loop (that goes through the transitions labelled* † *and* ‡*) and that, as we shall see later, every state in* A_3 *is reachable and live.*

3.1 Correctness

In preparation for the correctness proof, we demonstrate that certain invariants hold throughout the execution of the algorithm. The first step is to verify that the selections required by the main loop of Algorithm 1 are always feasible.

Lemma 3.6 *For every tree* $t \in T_\Sigma \setminus \mathscr{P}$ *and run* $r \in runs_{A_{\mathscr{R},\mathscr{P}}}(t)$ *such that* $r(\varepsilon) \subseteq \mathscr{P}$, *the set of nodes* $gen(r)$ *is nonempty.*

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press

Proof
Assume that the conditions of Lemma 3.6 hold, but that $gen(r) = \emptyset$. It then follows that for every $v \in nodes(t)$, the set $basis_r(v) = basis'_r(v)$ contains the subtree t/v of t rooted at v. Since $basis_r(v) \subseteq r(v)$, we obtain that $t/\varepsilon = t$ is in $r(\varepsilon)$, which in turn is a subset of $\mathscr{P}$. Hence, $t \in \mathscr{P}$ which contradicts $t \in T_\Sigma \setminus \mathscr{P}$. □

Invariant 3.7 *The selections at lines 3, 4, 5 and 8 can always be made.*

Proof
Since $s \in \mathscr{L}(A_{\mathscr{R},\mathscr{P}})$, there is a run $r \in runs_{A_{\mathscr{R},\mathscr{P}}}(s)$ such that $r(\varepsilon) \subseteq \mathscr{P}$. Furthermore, since $s \in \mathscr{N}$, we know that $s \notin \mathscr{P}$, so by Lemma 3.6, $gen(r)$ is nonempty.

The tree t of line 5 must exist since $r(v) \in \delta_f(r(v1), \ldots, r(vk))$, for otherwise r would not be in $runs_{A_{\mathscr{R},\mathscr{P}}}(s)$. After the WHILE loop at lines 6 and 7, there is still at least one subtree t_i of t such that $t_i \notin basis_r(vi)$, or else v would not be in $gen(r)$, so the index sought at line 8 is guaranteed to exist. □

We continue to show that the refinement step of line 9 yields a proper refinement of the equivalence relation $\mathscr{R}$, thereby removing one source of over-generalisation. This result can then be used to prove that $A_{\mathscr{R},\mathscr{P}}$ contains no useless states.

Invariant 3.8 *The relation $\mathscr{R}$ is always an equivalence relation on subtrees$(\mathscr{P})$, and in every iteration of the main loop, $\mathscr{R}$ is properly refined.*

Proof
The proof is by induction on the number of times Algorithm 1 iterates over the main loop. After the initialisation step at line 1, $\mathscr{R} = \mathscr{P}^2 \cup (subtrees(\mathscr{P}) \setminus \mathscr{P})^2$ is clearly an equivalence relation on $subtrees(\mathscr{P})$.

For the inductive step, we show the refinement step at lines 9–11, (i) preserves reflexivity, transitivity and symmetry; and (ii) divides the block $r(vi)$ into two blocks $B_1 = \{t'_i \in r(vi) \mid t[\![i \leftarrow t'_i]\!] \in r(v)\}$ and $B_2 = r(vi) \setminus B_1$, such that $basis_r(vi) \subseteq B_2$.

The correctness of Claim (i) follows immediately from the fact that $\mathscr{R}$ is an equivalence relation and the definitions of B_1 and B_2 at lines 10 and 11.

Let us now verify Claim (ii). By definition, $basis_r(vi) \subseteq r(vi)$. If there was a tree $t'_i \in basis_r(vi)$ such that $t[\![i \leftarrow t'_i]\!] \in r(v)$, then this would contradict the satisfied termination criterion at line 6 and the choice of i at line 8 of Algorithm 1. It thus follows that $basis_r(vi) \cap B_1 = \emptyset$ and $basis_r(vi) \subseteq B_2$.

For every $u \in nodes(s)$, the set $basis_r(u)$ is nonempty (by Definition 3.4). In particular, $basis_r(vi)$ is nonempty and so is B_2 since it includes $basis_r(vi)$. Finally, we know that $r \in runs_{A_{\mathscr{R},\mathscr{P}}}(s)$, so there must be a $t'_i \in r(vi)$ such that $t[\![i \leftarrow t'_i]\!] \in r(v)$, which means that also B_1 is nonempty. □

Invariant 3.9 *Every state in $A_{\mathscr{R},\mathscr{P}}$ is reachable and live.*

Proof
By Invariant 3.8, $\mathscr{R}$ is an equivalence relation on $subtrees(\mathscr{P})$. By Definition 3.1, the states Q of $A_{\mathscr{R},\mathscr{P}}$ are the equivalence classes of $\mathscr{R}$. By Lemma 3.2, every state $[t]_{\mathscr{R}} \in Q$ is reachable on t.

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press

Let us also verify that $q = [t]_{\mathscr{R}}$ is live. As $[t]_{\mathscr{R}} \subseteq subtrees(\mathscr{P})$, there is a tree $s \in \mathscr{P}$ and a $v \in nodes(s)$ such that $s/v = t$. It follows that there is a run of $A_{\mathscr{R},\mathscr{P}}$ on s such that $r(\varepsilon) = [s]_{\mathscr{R}}$ and $r(v) = [t]_{\mathscr{R}}$, so r is accepting and $[t]_{\mathscr{R}}$ is live. □

Lemma 3.10 follows immediately from Invariant 3.8 and the finiteness of $\mathscr{P}$.

Lemma 3.10 *Algorithm 1 terminates.*

Corollary 3.3 in combination with the termination criterion at line 2 of Algorithm 1 yields Theorem 3.11.

Theorem 3.11 *Upon termination,* $\mathscr{P} \subseteq \mathscr{L}(A_{\mathscr{R},\mathscr{P}}) \subseteq (T_\Sigma \setminus \mathscr{N})$.

3.2 Complexity

In the complexity analysis, the size of a set of trees $\mathscr{X}$ is the sum of the sizes of its constituent trees. The sum $\sum_{t\in\mathscr{X}} |t|$ is denoted by $n_{\mathscr{X}}$ for $\mathscr{X} \in \{\mathscr{P}, \mathscr{N}\}$. To obtain a safe upper bound, the nondeterministic algorithm is assumed to persistently make the worst possible choices. We omit average-time calculations as they require a deeper statistical analysis and depend on the distributions that generate $\mathscr{P}$ and $\mathscr{N}$.

Theorem 3.12 *The worst-case time complexity of Algorithm 1 is in* $O\left(n_{\mathscr{P}}^2 \cdot n_{\mathscr{N}}\right)$.

Proof
An efficient implementation uses a directed acyclic graph representation of the set $\mathscr{P}$. The directed acyclic graph can be initialised in $O\left(n_{\mathscr{P}}^2\right)$ computation steps and requires no further maintenance. Each iteration of the main loop properly refines $\mathscr{R}$. Thus, after at most $n_{\mathscr{P}}$ iterations, the algorithm terminates. In each iteration, the algorithm synthesises $A_{\mathscr{R},\mathscr{P}}$ from $\mathscr{R}$ and $\mathscr{P}$, consuming $O(n_{\mathscr{P}})$ computation steps. By Theorem 1.7.3 of Comon *et al.* (1997), deciding whether there is an accepting run of t in $\mathscr{N}$ costs $O\left(|A_{\mathscr{R},\mathscr{P}}| \cdot |t|\right)$ steps, so finding any tree in $\mathscr{N}$ with an accepting run r costs $O(n_{\mathscr{P}} \cdot n_{\mathscr{N}})$ steps. Computing $gen(r)$ is almost equally expensive and is in $O\left(n_{\mathscr{P}} \cdot |t|\right)$, which is bounded by $O(n_{\mathscr{P}} \cdot n_{\mathscr{N}})$, but this includes the selection of v, t and i. Finally, $r(vi)$ can be split with respect to v, t and i in time $n_{\mathscr{P}}$ by taking advantage of the directed acyclic graph representation. The overall time complexity is thus $O\left(n_{\mathscr{P}}^2 \cdot n_{\mathscr{N}}\right)$. □

Theorem 3.12 reveals an asymmetry in the impact of $n_{\mathscr{P}}$ and $n_{\mathscr{N}}$ on the time complexity of Algorithm 1. For some applications, a complement automaton may be sufficient, i.e. an automaton that accepts every tree in $\mathscr{N}$ and rejects every tree in $\mathscr{P}$. When this is the case, we can find an automaton representation for our samples in $O\left(\min(n_{\mathscr{N}} \cdot n_{\mathscr{P}}^2, n_{\mathscr{P}} \cdot n_{\mathscr{N}}^2)\right)$ computation steps by running Algorithm 1 on input $(\mathscr{N}, \mathscr{P})$ rather than $(\mathscr{P}, \mathscr{N})$ whenever $n_{\mathscr{N}}$ is smaller than $n_{\mathscr{P}}$.

4 Experiments

To complement the theoretical results of Section 3, we implemented a prototype in Java. The prototype was executed on input samples ranging in size from 0 to

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press

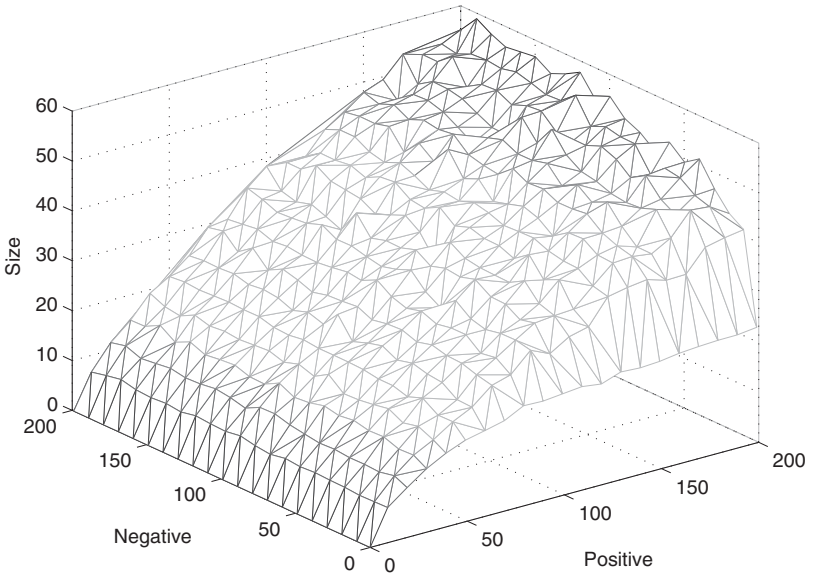


Fig. 4. The size of the output automaton as a function of $n_{\mathscr{P}}$ and $n_{\mathscr{N}}$.

200 nodes, and the size of the output automaton and the total running time were recorded. The samples were obtained as follows: A set of trees was generated at random using a regular tree grammar G, such that $\mathscr{L}(G) = T_\Sigma$, where $\Sigma = \{a^{(0)}, b^{(0)}, c^{(0)}, f^{(2)}, g^{(2)}, h^{(1)}\}$. The set was then divided, again at random, into disjoint sets $\mathscr{P}$ and $\mathscr{N}$ (not necessarily of equal size). For each measuring point, 50 test runs were made and the average values computed.

The results of the experiments are reported in Figures 4 and 5. The former shows the size of the output automaton as a function of the size of the input sample. In our tests, the output is typically smaller than the input, so for some applications the algorithm may provide a useful form of compression. Figure 5 shows the execution time of the prototype (when run on a MacBook with a 2.4 GHz Intel Core 2 Duo processor and 4 GB of memory), and the empirical results appear to support the theoretical prediction.

5 Future work

One direction for future work is to lift the family of algorithms described by de Parga *et al.* (2006) to the domain of trees and to compare their performance with that of Algorithm 1. From a theoretical point of view, it would also be nice if Algorithm 1 could be modified to allow identification in the limit. One could of course fall back on the strategy employed by Gold (1978), begin each run of the algorithm by searching for a characteristic sample and, if it exists, generate a minimal deterministic automaton, but this hardly seems satisfying, and a solution based on a canonical form for nondeterministic FTA would be more appealing.

The complexity analysis in Section 3.2 supposes a worst-case scenario, and the experiments in Section 4 are made with randomly generated input sets. To better

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press

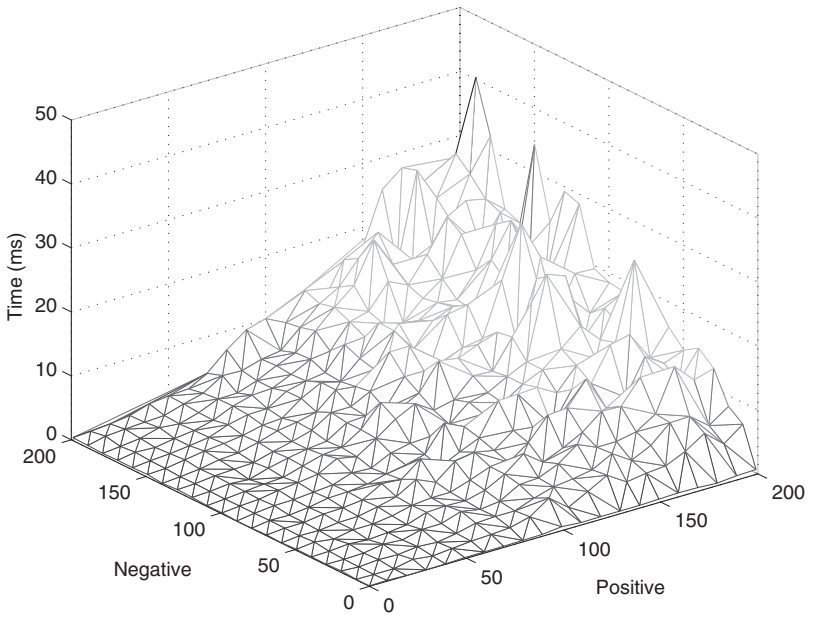


Fig. 5. The running time of the prototype (in milliseconds) as a function of $n_{\mathscr{P}}$ and $n_{\mathscr{N}}$.

understand the appropriateness of Algorithm 1 for practical natural language processing applications, it would be useful to estimate the average-case time complexity and to conduct more extensive experiments on real-world data.

Acknowledgments

We thank the referees for their helpful comments and suggestions. We are also grateful to Henrik Björklund, Frank Drewes and Brink van der Merwe for reading and discussing the manuscript.

References

Adriaans, P. 1999. Learning shallow context-free languages under simple distributions. Technical Report PP-1999-13, Institute for Logic, Language, and Computation, Amsterdam.

Adriaans, P., Trautwein, M., and Vervoort, M. 2000. Towards high speed grammar induction on large text corpora. In *Proceedings of the 27th Annual Conference on Current Trends in Theory and Practice of Informatics*, Volume 1963 of *LNCS*, Milovy, Czech Republic, pp. 173–86. 0.501961,1,1Berlin/Heidelberg, Germany: Springer-Verlag.

Angluin, D. 1981. A note on the number of queries needed to identify regular languages. *Information and Computation* **51**: 76–87.

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

Besombes, J., and Marion, J. 2007. Learning tree languages from positive examples and membership queries. *Theoretical Computer Science* **382**: 183–97.

Bod, R. 1999. Extracting stochastic grammars from treebanks. In R. Delmonte (ed.), *Proceedings of Venezia per il Trattamente Automatico delle Lingue*, Venice, Italy, pp. 229–34. Padova, Italy: Unipress.

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press

Buchholz, P. 2008. Bisimulation relations for weighted automata. *Theoretical Computer Science* **393**(1–3): 109–23.

Clark, A., and Eyraud, R. 2005. Identification in the limit of substitutable context-free languages. In S. Jain, H. Simon, and E. Tomita (eds.), *Proceedings of the 16th International Conference on Algorithmic Learning Theory*, Volume 3734 of *LNAI*, Singapore, pp. 283–96. Berlin/Heidelberg, Germany: Springer-Verlag.

Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M. 1997. Tree automata techniques and applications. Resource location. `http://www.grappa.univ-lille3.fr/tata`. Accessed October 26th, 2010.

Coste, F., and Fredouille, D. 2003. Unambiguous automata inference by means of state-merging methods. In N. Lavrac, D. Gamberger, L. Todorovski, and H. Blockeel (eds.), *Proceedings of the European Conference on Machine Learning*, Volume 2837 of *LNCS*, Cavtat-Dubrovnik, Croatia, pp. 60–71. Berlin/Heidelberg, Germany: Springer-Verlag.

Denis, F., Lemay, A., and Terlutte, A. 2000. Learning regular languages using non deterministic finite automata. In A. L. Oliveira (ed.), *Proceedings of the 5th International Conference on Grammatical Inference: Algorithms and Applications*, Volume 1891 of *LNCS*, Lisbon, Portugal, pp. 39–50. Berlin/Heidelberg, Germany: Springer-Verlag.

Denis, F., Lemay, A., and Terlutte, A. 2001. Residual finite state automata. In A. Ferreira, and H. Reichel (eds.), *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science*, Volume 2010 of *LNCS*, pp. 144–57. Berlin/Heidelberg, Germany: Springer 0.501961,1,1-Verlag.

de Parga, M. V., García, P., and Ruiz, J. 2006. A family of algorithms for non deterministic regular languages inference. In O. Ibarra, and H. Yen (eds.), *Proceedings of the 11th International Conference on Implementation and Application of Automata*, Volume 4094 of *LNCS*, pp. 265–74. Berlin/Heidelberg, Germany: Springer-Verlag.

Drewes, F. 2009. MAT learners for recognizable tree languages and tree series. *Acta Cybernetica* **19**(2): 249–74.

Drewes, F., and Högberg, J. 2007. Query learning of regular tree languages: how to avoid dead states. *Theory of Computing Systems* **40**: 163–85.

Drewes, F., and Vogler, H. 2007. Learning deterministically recognizable tree series. *Journal of Automata, Languages and Combinatorics* **12**(3): 333–54.

Edelman, S., Solan, Z., Horn, D., and Ruppin, E. 2005. Learning syntactic constructions from raw corpora. In A. Brugo, M. Clark-Cotton, and S. Ha (eds.), *Proceedings of the 29th Boston University Conference on Language Development*. MA, USA: Boston University.

García, P., de Parga, M., Álvarez, G., and Ruiz, J. 2008. Learning regular languages using nondeterministic finite automata. In O. Ibarra, and B. Ravikumar (eds.), *Conference on Implementation and Application of Automata*, Volume 5148 of *LNCS*, San Francisco, California, pp. 92–101. Berlin/Heidelberg, Germany: Springer 0.501961,1,1-Verlag.

Garcia, P., and Oncina, J. 1993. Inference of recognizable tree sets. Technical Report DSIC-II/47/93, Universidad Politecnica de Valencia, Valencia, Spain.

Geertzen, J., and van Zaanen, M. 2004. Grammatical inference using suffix trees. In G. Paliouras, and Y. Sakakibara (eds.), *Proceedings of the 7th International Conference on Grammatical Inference: Algorithms and Applications*, Volume 3264 of *LNCS*, Athens, Greece. Berlin/Heidelberg, Germany: Springer-Verlag.

Gold, E. M. 1967. Language identification in the limit. *Information and Control* **10**(5): 447–74.

Gold, E. M. 1978. Complexity of automaton identification from given data. *Information and Control* **37**(3): 302–20.

Habrard, A., and Oncina, J. 2006. Learning multiplicity tree automata. In Y. Sakakibara, S. Kobayashi, K. Sato, T. Nishino, and E. Tomita (eds.), *Proceedings of the 8th International Colloquium on Grammatical Inference: Algorithms and Applications*, Volume 4201 of *LNCS*, Tokyo, Japan, pp. 268–80. Berlin/Heidelberg, Germany: Springer-Verlag.

Harris, Z. S. 1951. *Methods in Structural Linguistics*. Chicago, IL, USA: University of Chicago Press.

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press

Knight, K., and May, J. 2009. Applications of weighted automata in natural language processing. In M. Droste, W. Kuich, and H. Vogler (eds.), *Handbook of Weighted Automata*. Berlin/Heidelberg, Germany: Springer-Verlag.

Lang, K., Pearlmutter, B., and Price, R. 1998. Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In V. Honavar, and G. Slutzki (eds.), *Proceedings of the 4th International Colloquium on Grammatical Inference: Algorithms and Applications, Ames, Iowa*, Volume 1433 of *LNCS*, pp. 1–12. Berlin/Heidelberg, Germany: Springer-Verlag.

Maletti, A. 2007. Learning deterministically recognizable tree series — revisited. In S. Bozapalidis, and G. Rahonis (eds.), *Conference on Algebraic Informatics*, Volume 4728 of LNCS, Thessaloniki, Greece, pp. 218–35. Berlin/Heidelberg, Germany: Springer-Verlag.

Oncina, J., and Garcia, P. 1991. Inferring regular languages in polynomial update time. In N. P. de la Blanca, A. Sanfeliu, and E. Vidal (eds.), *Pattern Recognition and Image Analysis*, pp. 49–61. Singapore: World Scientific.

Pitt, L., and Warmuth, M. 1989. The minimum consistent DFA problem cannot be approximated within any polynomial. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, MA, USA. ACM, pp. 421–32.

Russell, S. J., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach,* 2nd ed. NJ, USA: Prentice Hall.

Sakakibara, Y., and Muramatsu, H. 2000. Learning context-free grammars from partially structured examples. In A. L. Oliveira (ed.), *Proceedings of the 5th International Conference on Grammatical Inference: Algorithms and Applications*, Volume 1891 of *LNCS*, Lisbon, Portugal, pp. 229–40. Berlin/Heidelberg, Germany: Springer-Verlag.

Schmidt, E. M. 1978. *Succinctness of Description of Context-Free, Regular and Unambiguous Languages*. PhD thesis. Cornell University, Ithaca, New York.

Trakhtenbrot, B., and Barzdin, Y. 1973. *Finite Automata: Behavior and Synthesis*, Volume 1 of *FSCS*. Amsterdam, North-Holland: Elsevier. Translated from Russian by D. Louvish.

van Zaanen, M. 2001. *Bootstrapping Structure into Language: Alignment-Based Learning*. PhD thesis. School of Computing, University of Leeds, Leeds, UK.

https://doi.org/10.1017/S1351324911000064 Published online by Cambridge University Press