Theoretical Computer Science 578 (2015) 72–87

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Hyper-optimization for deterministic tree automata[☆]

Andreas Maletti[1]

Institute for Natural Language Processing, Universität Stuttgart, Pfaffenwaldring 5b, 70569 Stuttgart, Germany

ARTICLE INFO

Article history:
Received 28 October 2013
Received in revised form 20 May 2014
Accepted 14 January 2015
Available online 19 January 2015

Keywords:
Hyper-minimization
Deterministic tree automaton
Lossy compression
Error analysis
Error optimization

ABSTRACT

Hyper-minimization is a lossy minimization technique that allows a finite number of errors. It was already demonstrated that hyper-minimization can be performed efficiently for deterministic string automata and (bottom-up) deterministic tree automata (DTAs). The asymptotically fastest DTA hyper-minimization algorithms run in time $\mathcal{O}(m \cdot \log n)$, where m is the size of the DTA and n is the number of its states. In this contribution, the committed errors are investigated. First, the structure of all hyper-minimal DTAs for a given tree language is characterized, which also yields a formula for the number of such hyper-minimal DTAs. Second, an algorithm is developed that computes the number of errors that a given hyper-minimal DTA commits when compared to a given reference DTA. Third, it is shown that optimal hyper-minimization (i.e., computing a hyper-minimal DTA that commits the least number of errors of all hyper-minimal DTAs) can be achieved in time $\mathcal{O}(m \cdot n)$. Finally, a discussion of various other error measures (besides only their number) is provided.

1. Introduction

In many application areas, large finite-state models are approximated automatically from data. Classical examples in the area of natural language processing include the estimation of probabilistic tree automata [1–3] for parsing [4], probabilistic finite-state automata [5] for speech recognition [6], and probabilistic finite-state transducers [7] for machine translation [8]. Since those models tend to become huge, various efficient minimization techniques (such as the merge-steps in the Berkeley parser training [4] or bisimulation minimization [9,10]) are used. Unfortunately, already the computation of an equivalent minimal nondeterministic finite-state automaton (NFA) [11] given an input NFA is PSPACE-complete [12] and thus inefficient; this remains true even if the input NFA is deterministic. However, given a deterministic finite-state automaton (DFA) the computation of an equivalent minimal DFA is very efficient [13]. Consequently, we restrict our focus to deterministic finite-state devices. Exactly, the same situation exhibits itself for tree automata [14,15], which are the finite-state models used in this contribution. In addition, (bottom-up) deterministic tree automata are as expressive as (nondeterministic) tree automata, which recognize exactly the regular tree languages.

In several applications (e.g., digital audio compression) it is beneficial to reduce the size at the expense of errors. This approach is generally called 'lossy compression', and naturally, there are application-specific constraints on the permitted errors. In hyper-minimization [16] we simply allow any finite number of errors; i.e., the obtained representation might recognize a language that has a finite symmetric difference to the language recognized by the original representation.

☆ This is an extended and revised version of [25].
E-mail address: andreas.maletti@ims.uni-stuttgart.de.
1 The author was financially supported by the German Research Foundation (DFG) grant MA 4959/1-1.

http://dx.doi.org/10.1016/j.tcs.2015.01.020
0304-3975/© 2015 Elsevier B.V. All rights reserved.

This constraint is rather simplistic, but it allows a convenient theoretical treatment [16] and efficient minimization algorithms [17–20]. Besides there are models for which finitely many errors are absolutely inconsequential [21]. Moreover, refined constraints often yield NP-hard minimization problems [22] and thus inefficient minimization procedures. Recently, an efficient hyper-minimization algorithm [23] for (bottom-up) deterministic tree automata (DTAs) was developed. It runs in time $\mathcal{O}(m \cdot \log n)$, where m is the size of the input DTA and n is the number of its states. Thus, it is asymptotically as efficient as the fastest classical minimization algorithms [10] for DTAs.

All existing hyper-minimization algorithms for DTAs are purely qualitative in the sense that they guarantee that the returned DTA meets the error constraint and is hyper-minimal, which means that all DTAs with strictly fewer states[2] than the returned DTA violate the error constraint (i.e., recognize tree languages that have infinite difference to the tree language recognized by the input DTA). Consequently, the returned DTA is as small as possible subject to our error constraint. However, there is no (non-trivial) bound on the number of errors it might commit. In general, there are many (non-isomorphic) hyper-minimal DTAs for a given tree language. Our first result characterizes the differences between such alternatives in the spirit of [16, Theorems 3.8 and 3.9]. It shows that two such hyper-minimal DTAs permit a bijection between their states such that the distinction into preamble (i.e., those states that can only be reached by finitely many trees) and non-preamble (or kernel) states is preserved. Moreover, the two DTAs behave equivalently on their preambles except for their acceptance decisions and isomorphically on their kernels. In summary, such DTAs can only differ in two aspects:

- the finality (i.e., whether the state is final or not) of preamble states, and
- transition targets for transitions from exclusively preamble states to a kernel state.

Since we can easily count the number of permissible options, our characterization allows us to predict how many such hyper-minimal DTAs exist for the given input DTA.

The main application of the characterization is the calculation of the number of errors committed by a given hyper-minimal DTA inspired by the approach of [24]. In this sense we deliver a much more refined, quantitative analysis of hyper-minimization. To calculate the errors we first associate the errors to the states in which they occur. For preamble states the choice of their finality directly impacts which errors are caused. Due to the close connection between such DTAs we can easily compute the tree language recognized in each preamble state. Consequently, all errors associated to preamble states can be easily explained by the choice of finality. The analysis is slightly more complicated for errors associated to kernel states. Clearly, such error trees must have subtrees in which they took a special transition, which is a transition from exclusively preamble states to a kernel state, at the root. All kernel errors are then attributed to the special transitions. Since an error tree might have several such subtrees, we need to select a unique special transition for each error tree. Otherwise we would count the same error tree several times. We chose to select the left-most special transition, but other selections are possible. Fortunately, the restriction to left-most special transitions can be incorporated easily into the approach of [24]. Overall, we obtain an algorithm that computes the number of errors committed by a hyper-minimal DTA N in comparison to a reference DTA M (that recognizes a finitely different tree language) in time $\mathcal{O}(m \cdot n)$, where m is the size of M and n is the number of states of M. Since our association and attribution of errors is such that the choices do not impact another, we can simply optimize each choice such that it causes the least number of errors locally. In this way we obtain a DTA that also globally commits the least number of errors. Thus, we can also compute an optimal hyper-minimal DTA N' in time $\mathcal{O}(m \cdot n)$, where 'optimal' means that it commits the least number of errors among all hyper-minimal DTAs that recognize a finitely different tree language. Naturally, we can also compute the exact number of errors committed by this optimal DTA. Finally, we quickly discuss alternative optimality criteria since the number of errors is only one natural criterion. We focus on criteria that score the error trees since this approach meets our requirements. We discuss more realistic scoring functions might add up the error tree sizes or calculate the maximal height of an error tree. Unfortunately, not all such scoring functions can easily be incorporated into our algorithm.

2. Preliminaries

The set $\mathbb{N}$ contains all nonnegative integers, and we let $[k] = \{i \in \mathbb{N} \mid 1 \leq i \leq k\}$ for all $k \in \mathbb{N}$. The symmetric difference $S \ominus T$ of sets S and T is $(S - T) \cup (T - S)$. If a set S is finite, then we write $|S|$ for its cardinality. A binary relation $\cong \subseteq S \times S$ is an equivalence relation if it is reflexive (i.e., $s \cong s$ for all $s \in S$), symmetric (i.e., $s \cong s'$ implies $s' \cong s$ for all $s, s' \in S$), and transitive (i.e., $s \cong s'$ and $s' \cong s''$ imply $s \cong s''$ for all $s, s', s'' \in S$). Given such an equivalence relation $\cong$, the equivalence class $[s]_\cong$ of $s \in S$ is $[s]_\cong = \{s' \in S \mid s \cong s'\}$. Moreover, $(S/\cong) = \{[s]_\cong \mid s \in S\}$.

An alphabet Σ is a finite set, of which the elements are usually called symbols. A *ranked alphabet* (Σ, rk) consists of an alphabet Σ and a mapping $\mathrm{rk}\colon \Sigma \to \mathbb{N}$ that assigns a rank to each symbol of Σ. The set of all symbols of rank k is $\Sigma_k = \{\sigma \in \Sigma \mid \mathrm{rk}(\sigma) = k\}$ for every $k \in \mathbb{N}$. We typically denote a ranked alphabet (Σ, rk) by just Σ and assume that the ranks are clear from the context (or irrelevant). Similarly, we often omit universal quantifications like "for all $k \in \mathbb{N}$" especially for ranks. Whenever we want to explicitly indicate the rank of a symbol, we write $\sigma^{(k)}$ for a k-ary symbol σ. For every set T, we let $\Sigma(T) = \{\sigma(t_1, \ldots, t_k) \mid \sigma \in \Sigma_k, t_1, \ldots, t_k \in T\}$ contain all trees with a root node labeled by a symbol

[2] Since we consider only deterministic devices, we can as well use the number of transitions as a size measure.