Hyper-Minimization for Deterministic Tree Automata

Artur Jeż[1,⋆] and Andreas Maletti[2,⋆⋆]

[1] Institute of Computer Science, University of Wrocław
ul. Joliot-Curie 15, 50-383 Wrocław, Poland
email: aje@cs.uni.wroc.pl
[2] Institute for Natural Language Processing, Universität Stuttgart
Pfaffenwaldring 5b, 70569 Stuttgart, Germany
email: andreas.maletti@ims.uni-stuttgart.de

Abstract. Hyper-minimization aims to reduce the size of the representation of a language beyond the limits imposed by classical minimization. To this end, the hyper-minimal representation can represent a language that has a finite difference to the original language. The first hyper-minimization algorithm is presented for (bottom-up) deterministic tree automata, which represent the recognizable tree languages. It runs in time $\mathcal{O}(\ell mn)$, where ℓ is the maximal rank of the input symbols, m is the number of transitions, and n is the number of states of the input tree automaton.

1 Introduction

Hyper-minimization for deterministic finite-state string automata (dfa) [17] allows us to reduce the size of a dfa at the expense of a finite number of errors. The original article [2] that introduced hyper-minimization and its theoretical foundations also presented the first hyper-minimization algorithm, which was subsequently improved to $\mathcal{O}(mn)$ [1] and to $\mathcal{O}(m \log n)$ [4, 8], where m is the number of transitions and n is the number of states of the input dfa. Thus, the fastest hyper-minimization algorithms have the same asymptotic time complexity as the fastest algorithms for dfa minimization [9]. Since hyper-minimization trivially reduces to minimization [8], faster hyper-minimization algorithms would imply faster minimization algorithms, which have remained elusive. Hyper-minimization was already generalized to weighted dfa [13] and to dfa over infinite strings [15]. An overview of existing hyper-minimization algorithms can be found in [12].

We generalize hyper-minimization to deterministic tree automata (dta) [5, 6], which have applications in XML processing [10] and natural language processing [11]. We faithfully generalize the existing definitions from dfa to dta.

⋆ Financial support provided by the Polish National Science Centre (NCN) grant DEC-2011/01/D/ST6/07164.
⋆⋆ Financial support provided by the German Research Foundation (DFG) grant MA 4959/1-1.

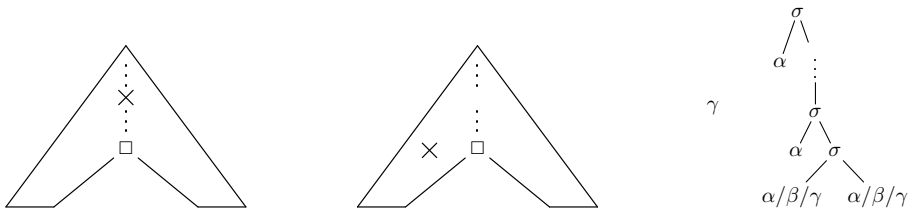

Fig. 1. Illustration of the difference locations in a context: along the path to the root (left) and off this path (middle), and illustration of the tree language $L(M_{\mathrm{ex}})$ of Ex. 2.

Thus, our hyper-minimization for dta is based on a congruence that is similar to the context-language equivalence used in dta minimization [3]. The fastest known algorithm for dta minimization [7] runs in time $\mathcal{O}(\ell m \log n)$, where ℓ is the maximal rank of the input symbols, m is the number of transitions, and n is the number of states of the input dta. The hyper-minimization algorithm that we present has the run-time complexity $\mathcal{O}(\ell m n)$, which is slightly worse than traditional minimization, but we believe that our algorithm can be improved using the standard techniques used in hyper-minimization of dfa. We sketch the improved version in Sect. 5.

Dta hyper-minimization is not a straightforward adjustment of dfa hyper-minimization. While they share the same principal structure, the actual properties used in the algorithms are different. The main reason for the differences is the location of the errors in the recognized context language. They can not only occur in the successor states (as for dfa) but can also occur in sibling states (see Fig. 1). This yields that several foundational results for dfa hyper-minimization [2] do not faithfully generalize to dta. Nevertheless, we borrow much of the surrounding infrastructure from the existing hyper-minimization algorithms [8, 14] and despite the theoretical differences, we obtain an efficient hyper-minimization algorithm following the approach of [1].

2 Preliminaries

The set of all nonnegative integers is $\mathbb{N}$, and we let $[k] = \{i \in \mathbb{N} \mid 1 \leq i \leq k\}$ for every $k \in \mathbb{N}$. The cardinality of a finite set S is denoted by $|S|$. The symmetric difference $S \ominus T$ of sets S and T is $(S - T) \cup (T - S)$. If $S \ominus T$ is finite, then S and T are almost equal. A binary relation $\cong$ on S is an equivalence if it is reflexive, symmetric, and transitive. We often present them as partitions of S.

An alphabet Σ is a finite set, and a ranked alphabet (Σ, rk) consists of an alphabet Σ and a mapping $\mathrm{rk}\colon \Sigma \to \mathbb{N}$, which assigns a rank to each symbol of Σ. For every $k \in \mathbb{N}$, we let $\Sigma_k = \mathrm{rk}^{-1}(k)$ be the set of all symbols of rank k. In the following, we typically denote the ranked alphabet (Σ, rk) by just Σ. For a set T, we let $\Sigma(T) = \{\sigma(t_1, \ldots, t_k) \mid \sigma \in \Sigma_k, t_1, \ldots, t_k \in T\}$. The set $T_\Sigma(Q)$ of Σ-trees with states Q is the smallest set T such that $Q \cup \Sigma(T) \subseteq T$. We

write T_Σ for $T_\Sigma(\emptyset)$. The height $\mathrm{ht}(t)$ of $t \in T_\Sigma(Q)$ is recursively defined as follows: $\mathrm{ht}(q) = 0$ for all $q \in Q$ and $\mathrm{ht}(\sigma(t_1, \ldots, t_k)) = 1 + \max\{\mathrm{ht}(t_i) \mid i \in [k]\}$ for all $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma(Q)$. The set $\mathrm{states}(t)$ is the minimal set Q such that $t \in T_\Sigma(Q)$. The set of positions of a tree $t \in T_\Sigma(Q)$ is denoted by $\mathrm{pos}(t)$, of which those that are labeled by $q \in Q$ form the set $\mathrm{pos}_q(t)$. Finally, for every $t \in T_\Sigma(Q)$, $q, q' \in Q$, and $w \in \mathrm{pos}_q(t)$, the tree $t[q']_w$ is obtained from t by relabeling the occurrence of q at w to q'.

A context c is a tree of $T_{\Sigma \cup \{\Box\}}(Q)$, in which the special nullary symbol $\Box$ occurs exactly once. The set of all such contexts is $C_\Sigma(Q)$, and we write C_Σ for $C_\Sigma(\emptyset)$. For every $c \in C_\Sigma(V)$ and $t \in T_{\Sigma \cup \{\Box\}}(Q)$, the tree $c[t] \in T_{\Sigma \cup \{\Box\}}(Q)$ denotes the tree obtained from c by replacing the unique occurrence of $\Box$ by t. A tree $t' \in T_\Sigma(Q)$ is a subtree of $t \in T_\Sigma(Q)$ if there exists a context $c \in C_\Sigma(Q)$ such that $t = c[t']$. The subtree is strict if $t \neq t'$. The depth of a context $c \in C_\Sigma(Q)$ is recursively defined by $\mathrm{dp}(\Box) = 0$ and

$$\mathrm{dp}(\sigma(t_1, \ldots, t_{i-1}, c, t_{i+1}, \ldots, t_k)) = 1 + \mathrm{dp}(c)$$

for every $\sigma \in \Sigma_k$, index $i \in [k]$, context $c \in C_\Sigma(Q)$, and $t_1, \ldots, t_k \in T_\Sigma(Q)$.

A deterministic tree automaton (dta) [5, 6] is a tuple $M = (Q, \Sigma, \delta, F)$ where Q is a finite set of states, Σ is a ranked alphabet of input symbols, $\delta \colon \Sigma(Q) \to Q$ is a (partial) transition function, and $F \subseteq Q$ is a set of final states. The dta M is total if δ is total. The transition function δ extends to $\delta \colon T_\Sigma(Q) \to Q$ by $\delta(q) = q$ for every $q \in Q$ and $\delta(\sigma(t_1, \ldots, t_k)) = \delta(\sigma(\delta(t_1), \ldots, \delta(t_k)))$ for every $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma(Q)$. We let $L(M)^q_{q'} = \{c \in C_\Sigma \mid \delta(c[q']) = q\}$ for every $q, q' \in Q$. Moreover, $L(M)_{q'} = \bigcup_{f \in F} L(M)^f_{q'}$ contains all contexts c such that $c[q']$ takes M into a final state and $L(M)^q = \delta^{-1}(q) \cap T_\Sigma$ contains all (stateless) trees that take M into the state q. The dta M recognizes the tree language $L(M) = \bigcup_{f \in F} L(M)^f$. In the following, we assume that every considered dta M is trim (or equivalently: has only reachable states), which means that $L(M)^q \neq \emptyset$ for every $q \in Q$.

An equivalence $\cong$ on Q is a congruence (on the dta M) if we have that $\delta(\sigma(q_1, \ldots, q_k)) \cong \delta(\sigma(q'_1, \ldots, q'_k))$ for every $\sigma \in \Sigma_k$ and $q_1 \cong q'_1, \ldots, q_k \cong q'_k$. Two states $q, q' \in Q$ are equivalent, which is denoted by $q \equiv_M q'$ (or just $q \equiv q'$), if $L(M)_q = L(M)_{q'}$. We sometimes use those notions for states from different dta over the same ranked alphabet with the obvious meaning. Note that $\equiv_M$ is a congruence, and actually, the coarsest (i.e., least refined) congruence on M that respects F, which means that a final state cannot be equivalent to a nonfinal state. The dta M is minimal if there exists no equivalent dta with strictly fewer states. It is well-known that M is minimal if and only if it does not have two different, but equivalent states. For every dta M, an equivalent minimal dta can be computed efficiently using an adaptation [7] of Hopcroft's algorithm [9], which runs in time $O(\ell m \log n)$ where $\ell = \max \mathrm{rk}(\Sigma)$ is the maximal rank of the input symbols, $m = |\mathrm{dom}(\delta)|$ is the number of transitions, and $n = |Q|$ is the number of states. From now on, let $M = (Q, \Sigma, \delta, F)$ be a minimal dta, which automatically yields that M is trim. Finally, we recall a central notion from [2] that will also be important in our setting. A state $q \in Q$ is a kernel

state if $L(M)^q$ is infinite. Otherwise q is a preamble state. The sets of kernel and preamble states are denoted by $\mathrm{Ker}(M)$ and $\mathrm{Pre}(M)$, respectively.

3 Hyper-minimal automata

The goal of hyper-minimization for a given dta M is the efficient computation of a dta that is as small as possible (measured by the number of states) and recognizes a tree language with finite difference to $L(M)$. A dta for which such a strictly smaller dta does not exist is called 'hyper-minimal', and we investigate the properties of these dta here. Before we can start, we need to introduce the main notions of this contribution and some essential properties.

For the rest of the section, we consider a minimal dta $M = (Q, \Sigma, \delta, F)$. To simplify the theoretical discussion, we assume that M is total. This can be achieved by adding a sink state $\perp$ as the target of all missing transitions of a partial dta. It should be noted that all properties of this section trivially extend to partial dta. The totality assumption made is purely a convenience.

A minimal dta is obtained by identifying and merging equivalent states. Accordingly, our goal is to obtain a hyper-minimal dta by identifying and merging almost equivalent states, where 'almost equivalent' has the usual mathematical meaning (i.e., equivalent up to a finite number of differences).

Definition 1 (cf. [2, Def. 2.2]). *The states $q, q' \in Q$ are* almost equivalent*, written $q \sim q'$, if $L(M)_q \ominus L(M)_{q'}$ is finite.*

Example 2. The running example dta M_{ex} is (Q, Σ, δ, F), where

- $Q = \{q_\alpha, q_\beta, q_\gamma, q_\sigma, \perp\}$,
- $\Sigma = \Sigma_0 \cup \Sigma_2$ with $\Sigma_0 = \{\alpha, \beta, \gamma\}$ and $\Sigma_2 = \{\sigma\}$,
- $F = \{q_\sigma, q_\gamma\}$, and
- δ returns $\perp$ except that for every $\sigma_0, \sigma_0' \in \Sigma_0$ we have

$$\delta(\sigma_0) = q_{\sigma_0} \qquad \delta(\sigma(q_{\sigma_0}, q_{\sigma_0'})) = q_\sigma \qquad \delta(\sigma(q_\alpha, q_\sigma)) = q_\sigma \ .$$

It recognizes the tree language $\{\gamma\} \cup \{c^n[\sigma(\sigma_0, \sigma_0')] \mid n \in \mathbb{N}, \sigma_0, \sigma_0' \in \Sigma_0\}$, where $c = \sigma(\alpha, \square)$, $c^0 = \square$, and $c^{n+1} = c^n[c]$ for every $n \in \mathbb{N}$ (see Fig. 1). Note that M_{ex} is minimal. However, q_β and q_γ are almost equivalent because

$$L(M_{\mathrm{ex}})_{q_\beta} = \{c^n[\sigma(\square, \sigma_0)] \mid n \in \mathbb{N}, \sigma_0 \in \Sigma_0\} \cup \{c^n[\sigma(\sigma_0, \square)] \mid n \in \mathbb{N}, \sigma_0 \in \Sigma_0\}$$
$$L(M_{\mathrm{ex}})_{q_\gamma} = \{\square\} \cup L(M_{\mathrm{ex}})_{q_\beta} \ .$$

The state q_α is neither almost equivalent to q_β nor to q_σ. ◇

We immediately observe that for all $q_1 \sim q_2$ there is an integer $k \in \mathbb{N}$ such that $\delta(c[q_1]) = \delta(c[q_2])$ for all $c \in C_\Sigma$ with $\mathrm{dp}(c) > k$. Since the difference $L(M)_{q_1} \ominus L(M)_{q_2}$ is finite, we can select k such that it is strictly larger than the depth of any context in the difference. For any context c of depth at least k we obtain that $\delta(c[q_1])$ and $\delta(c[q_2])$ are equivalent, and thus, equal by minimality.

In contrast to the string case, the converse of the previous statement is not true, which shows that the generalization is nontrivial. In a dta not only the successor, but also the sibling states determine the almost equivalence (see Fig. 1). Although q_α and q_β have the same successor states in Ex. 2, they are not almost equivalent as they expect different sibling states.

Clearly, almost equivalence is an equivalence on Q. Next, we show that it is even a congruence on M. In contrast to the context equivalence that respects F, the almost equivalence $\sim$ clearly need not respect F (see Ex. 2 where $q_\gamma \sim q_\beta$ but $q_\gamma \in F$ and $q_\beta \notin F$).

Lemma 3 (see [2, Lm. 2.10]). *For all $q \sim q'$ and contexts $c \in C_\Sigma$, we have $\delta(c[q]) \sim \delta(c[q'])$. In particular, $\sim$ is a congruence.*

Proof. For each context $c' \in L(M)_{\delta(c[q])} \ominus L(M)_{\delta(c[q'])}$ we have that the context $c'[c] \in L(M)_q \ominus L(M)_{q'}$. Clearly, different c' yield different contexts $c'[c]$, so there can only be finitely many such contexts c' because $L(M)_q \ominus L(M)_{q'}$ is finite, which proves that $\delta(c[q]) \sim \delta(c[q'])$. The latter property is a simple consequence of the former via particular contexts of depth 1 and the standard piecewise replacement. Let $\sigma \in \Sigma_k$ and $q_1 \sim q'_1, \ldots, q_k \sim q'_k$ be almost equivalent states. Moreover, for each $q \in Q$, let $t_q \in L(M)^q$ be arbitrary. Then

$$\begin{aligned}
&\delta(\sigma(q_1, \ldots, q_k)) = \delta(\sigma(\Box, t_{q_2}, \ldots, t_{q_k})[q_1]) \\
&\sim \delta(\sigma(\Box, t_{q_2}, \ldots, t_{q_k})[q'_1]) = \delta(\sigma(q'_1, q_2, \ldots, q_k)) = \delta(\sigma(t_{q'_1}, \Box, t_{q_3}, \ldots, t_{q_k})[q_2]) \\
&\sim \ldots \\
&\sim \delta(\sigma(t_{q'_1}, \ldots, t_{q'_{k-1}}, \Box)[q'_k]) = \delta(\sigma(q'_1, \ldots, q'_k)) \ .
\end{aligned}$$

□

To complete the essential definitions, two dta M and N are almost equivalent if $L(M)$ and $L(N)$ are almost equal. Naturally, this is an equivalence relation on dta. Next, we relate the states of almost equivalent dta in order to prepare our characterization of hyper-minimal dta.

Lemma 4. *Let $M = (Q, \Sigma, \delta, F)$ and $N = (P, \Sigma, \mu, G)$ be minimal dta that are almost equivalent. Then $L(M)_{\delta(t)}$ and $L(N)_{\mu(t)}$ are almost equal for all $t \in T_\Sigma$.*

Proof. For every $L \subseteq T_\Sigma$, let $t^{-1}L = \{c \in C_\Sigma \mid c[t] \in L\}$. Since $L(M)$ and $L(N)$ are almost equal, also $t^{-1}L(M)$ and $t^{-1}L(N)$ are almost equal. Together with $t^{-1}L(M) = L(M)_{\delta(t)}$ and $t^{-1}L(N) = L(N)_{\mu(t)}$, we proved the statement. □

Now we make hyper-minimality precise. The dta M is *hyper-minimal* if all almost equivalent dta are at least as large (i.e., have at least as many states). We already remarked that we want to obtain hyper-minimal dta with the help of merging. In a *merge* of $q \in Q$ into $q' \in Q$ we redirect all transitions leading to q into q'. Formally, for every two different states $q, q' \in Q$, the dta merge$(M, q \to q')$ is $(Q - \{q\}, \Sigma, \delta', F - \{q\})$ where for every $s \in \Sigma(Q - \{q\})$

$$\delta'(s) = \begin{cases} q' & \text{if } s \in \delta^{-1}(q) \\ \delta(s) & \text{otherwise.} \end{cases}$$

Lemma 5. *If $q \sim q'$ and q is a preamble state, then* $\mathrm{merge}(M, q \to q')$ *and M are almost equivalent.*

Proof. Let $\mathrm{merge}(M, q \to q') = (Q', \Sigma, \delta', F')$. The set $D = L(M)_q \ominus L(M)_{q'}$ is finite because $q \sim q'$. We select ℓ with $\ell > \mathrm{ht}(c)$ for every $c \in D$. Let $t \in T_\Sigma$ be such that $\mathrm{ht}(t) \geq \ell + |Q|$. First we replace all subtrees $t' \in L(M)^q$ in t by just q. In this way, we obtain the tree u. Note that $\delta(t) = \delta(u)$ and $\mathrm{ht}(u) \geq \ell$ because $\mathrm{ht}(t') \leq |Q|$ for all $t' \in L(M)^q$ since q is a preamble state. Let $\mathrm{pos}_q(u) = \{w_1, \ldots, w_n\}$ with $w_1 < \cdots < w_n$ be the occurrences of q in u. For each $i \in [n]$, let $c_i = (u[q']_{w_1} \cdots [q']_{w_{i-1}})[\Box]_{w_i}$ be the context obtained from u by replacing the first $i-1$ occurrences by q' and the occurrence w_i by $\Box$. Note that $\mathrm{ht}(c_i) = \mathrm{ht}(u) \geq \ell$, which allows us to obtain

$$\delta(t) = \delta(u) = \delta(c_1[q]) = \delta(c_1[q']) = \delta(c_2[q]) = \cdots = \delta(c_n[q']) \overset{\dagger}{=} \delta'(t) \ ,$$

where $\dagger$ holds because δ and δ' coincide on all transitions not involving q. Consequently, $\mathrm{merge}(M, q \to q')$ and M agree on all tall trees as desired. □

Example 6. Recall the dta $M_{\mathrm{ex}} = (Q, \Sigma, \delta, F)$ of Ex. 2. If we merge q_β into q_γ, then we obtain the dta $\mathrm{merge}(M_{\mathrm{ex}}, q_\beta \to q_\gamma)$, which is $(Q - \{q_\beta\}, \Sigma, \delta', F)$ where δ' returns $\bot$ except that for every $\sigma_0, \sigma_0' \in \Sigma_0$ we have

$$\begin{aligned} \delta'(\alpha) &= q_\alpha & \delta'(\beta) &= q_\gamma & \delta'(\gamma) &= q_\gamma \\ \delta'(\sigma(q_{\sigma_0}, q_{\sigma_0'})) &= q_\sigma & \delta'(\sigma(q_\alpha, q_\sigma)) &= q_\sigma \ . \end{aligned}$$ ◇

Now we can characterize hyper-minimality [2]. The characterization allows us to easily determine whether M is hyper-minimal. Recall that a dta is minimal if and only if it does not have two different, but equivalent states. The condition for hyper-minimality is similar, but adds a restriction to preamble states.

Theorem 7. *The minimal dta M is hyper-minimal if and only if every pair of different, but almost equivalent states consists of only kernel states.*

Proof. We start with the "only if"-direction. Suppose that there exist two different, but almost equivalent states $q, q' \in Q$ such that q is a preamble state. Then M is not hyper-minimal because $\mathrm{merge}(M, q \to q')$ is strictly smaller and almost equivalent to M by Lm. 5. For the converse, let $N = (P, \Sigma, \mu, G)$ be a hyper-minimal dta that is strictly smaller (i.e., $|P| < |Q|$) and almost equivalent to M. The product dta $M' = (Q \times P, \Sigma, \delta \times \mu, F \times G)$ is given by

$$(\delta \times \mu)(\sigma(\langle q_1, p_1\rangle, \ldots, \langle q_k, p_k\rangle)) = \langle \delta(\sigma(q_1, \ldots, q_k)), \mu(\sigma(p_1, \ldots, p_k))\rangle$$

for every $\sigma \in \Sigma_k$ and $\langle q_1, p_1\rangle, \ldots, \langle q_k, p_k\rangle \in Q \times P$. Since M is minimal, let $t_q \in L(M)^q$ for every $q \in Q$. If $q \in \mathrm{Ker}(M)$, then select t_q such that $\mathrm{ht}(t_q) \geq |Q|^2$. By the pigeon-hole principle with $|P| < |Q|$, there must exist different $q_1, q_2 \in Q$ and $p \in P$ such that $(\delta \times \mu)(t_q) = \langle q, p\rangle$ for $q \in \{q_1, q_2\}$. Consequently, $q_1 \sim q_2$ because $L(M)_{q_1}$ and $L(N)_p$ as well as $L(M)_{q_2}$ and $L(N)_p$ are almost equal by Lm. 4. This in turn yields that q_1 and q_2 are kernel states of M by assumption.

Algorithm 1 Structure of our dta hyper-minimization algorithm [2, 8].

Require: a dta M
Return: an almost equivalent hyper-minimal dta

```
    M ← Minimize(M)                                          // complexity: O(ℓm log n)
 2: K ← ComputeKernel(M)                                     // complexity: O(ℓm)
    ∼ ← {⟨q, q'⟩ ∈ Q² | L(M⊗)⟨q,q'⟩ is finite}               // see Sect. 4
 4: for all B ∈ (Q/∼) do
      select q_B ∈ B such that q_B ∈ K if possible
 6:   for all q ∈ B − K do
        M ← merge(M, q → q_B)                                // complexity: O(1)
 8: return M
```

Moreover, $\langle q_1, p\rangle$ and $\langle q_2, p\rangle$ are kernel states of M' by the selection of the access trees with $\mathrm{ht}(t_{q_1}) \geq |Q|^2 \leq \mathrm{ht}(t_{q_2})$ (because the trees t_{q_1} and t_{q_2} can be pumped [5, 6]). Now, for the sake of a contradiction, let $c \in L(M)_{q_1} \ominus L(N)_p$. Then $\{c[t] \mid t \in L(M')^{\langle q_1,p\rangle}\} \subseteq L(M) \ominus L(N)$. Since $\langle q_1, p\rangle$ is a kernel state of M', the set $L(M')^{\langle q_1,p\rangle}$ is infinite. This contradicts that M and N are almost equivalent, so consequently, $L(M)_{q_1} \ominus L(N)_p = \emptyset$, and $L(M)_{q_2} \ominus L(N)_p = \emptyset$ in the same manner. Thus, q_1 and q_2 are equivalent and $q_1 = q_2$ by the minimality of M, which shows that the dta N cannot exist. □

Example 8. The dta M_{ex} of Ex. 2 is not hyper-minimal since $q_\beta \sim q_\gamma$ and both states are preamble states (see Ex. 2). However, with a little effort we can show that the dta $\mathrm{merge}(M_{\mathrm{ex}}, q_\beta \to q_\gamma)$ is hyper-minimal (see Ex. 6). ◇

4 Hyper-minimization

The previous results suggest a hyper-minimization algorithm, which we sketch in Alg. 1. We work with the (potentially non-total) dta $M = (Q, \Sigma, \delta, F)$ now. In addition, we let $\ell = \max \mathrm{rk}(\Sigma)$, $m = |\mathrm{dom}(\delta)|$, and $n = |Q|$. Algorithm 1 simply determines the kernel states and the almost equivalence using methods that we describe later. It then merges states (by simply changing a reference) according to the conditions of Lm. 5, which guarantees that the result is almost equivalent. Finally, Thm. 7 shows that the obtained dta is hyper-minimal.

Corollary 9 (of Lm. 5 and Thm. 7). *Algorithm 1 returns a hyper-minimal dta that is almost equivalent to M.*

In Alg. 1 we use Minimize, which implements classical dta minimization [5, 6, 3] in time $O(\ell m \log n)$ using an adaptation of Hopcroft's algorithm [9, 7]. The procedure ComputeKernel computes the kernel states of M using any fast algorithm for computing strongly connected components in a graph (e.g., Tarjan [16]). The next proposition shows the trivial problem translation.

Proposition 10. $\mathrm{Ker}(M)$ *can be computed in time $O(\ell m)$.*

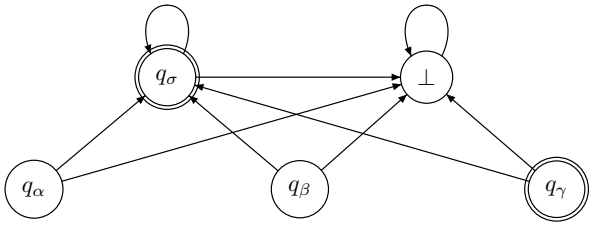

Fig. 2. The graph derived from the dta of Ex. 2.

Proof. We turn our dta M into the graph (Q, E), where

$$E = \{(q, \delta(t)) \mid t \in \mathrm{dom}(\delta), q \in \mathrm{states}(t)\} \ .$$

It is simple to observe that $q \in \mathrm{Ker}(M)$ if and only if it is reachable from a non-trivial strongly connected component of the graph (Q, E) [see [8] for details]. □

Example 11. The kernel states of the dta M_{ex} of Ex. 2 are $\{q_\sigma, \bot\}$, which is easily determined from the graph displayed in Fig. 2.

The final component is the identification of the almost equivalent states, which also determines the overall run-time of our hyper-minimization algorithm. For this final component, we use an adapted version of an algorithm from [1], which is simple but not the fastest. In the next section, we sketch how the currently fastest algorithms for dfa almost equivalence [4, 8] can be adjusted to our dta setting.

To simplify the presentation, we assume that $\delta(s) = \bot$ for the special token $\bot \notin Q$ if $\delta(s)$ is undefined. Note that we do not add $\bot$ to Q, so we do not make M total. In contrast, we just introduce a notational convenience.

Definition 12. The *exclusive-or single-point self-product* of M is the dta

$$M_\otimes = (P \cup P^2, \Sigma, \delta \cup \delta', F')$$

such that $P = Q \cup \{\bot\}$ with $\bot \notin Q$,

- $F' = \{\langle q, q' \rangle \mid$ either $q \in F$ or $q' \in F\}$, and
- for every $\sigma \in \Sigma_k$, $i \in [k]$ and $q, q', q_1, \ldots, q_k \in P$

$$\delta'(\sigma(c[\langle q, q' \rangle])) = \langle \delta(c[q]), \delta(c[q']) \rangle \ ,$$

  where $c = \sigma(q_1, \ldots, q_{i-1}, \Box, q_{i+1}, \ldots, q_k)$ and at least one of the δ-entries has to be defined (i.e., $Q \cap \{\delta(c[q]), \delta(c[q'])\} \neq \emptyset$).
- δ' is undefined otherwise. ◇

In other words, only the paired states in which one state is final and the other is nonfinal are now final states in $M_\otimes$. The transitions on pairs run componentwise with an explicit sink state component as long as at least one component is still a "normal" state. This special treatment is necessary to correctly handle partial dta.

Proposition 13. *We can construct $M_\otimes$ in time $\mathcal{O}(\ell mn)$.*

Proof. Clearly, we can create the states in time $\mathcal{O}(n^2)$. Since M is minimal, we have $n \leq m$, which yields $\mathcal{O}(n^2) \subseteq \mathcal{O}(mn)$. Clearly, for each transition in M, we construct at most $\mathcal{O}(\ell n)$ copies of that transition, which yields that we can construct all transitions in time $\mathcal{O}(\ell mn)$, where we assume that transition look-ups run in constant time. □

Example 14. Now we can handle the dta M_{ex} of Ex. 2 as a partial dta. The dta $(M_{\text{ex}})_\otimes$ is $(Q \cup Q^2, \Sigma, \delta \cup \delta', F')$, where

- $F' = \{\langle q_\sigma, q_\alpha\rangle, \langle q_\sigma, q_\beta\rangle, \langle q_\sigma, \bot\rangle, \langle q_\gamma, q_\alpha\rangle, \langle q_\gamma, q_\beta\rangle, \langle q_\gamma, \bot\rangle\}^{\text{sym}}$, where L^{sym} is the symmetric closure of L, and
- some interesting transitions of δ' include

$$\begin{aligned}
\delta'(\sigma(\langle q_\alpha, q_\beta\rangle, q_\sigma)) = \delta'(\sigma(q_\alpha, \langle q_\sigma, \bot\rangle)) &= \langle q_\sigma, \bot\rangle \\
\delta'(\sigma(\langle q_\beta, q_\gamma\rangle, q_\alpha)) = \delta'(\sigma(\langle q_\beta, q_\gamma\rangle, q_\beta)) = \delta'(\sigma(\langle q_\beta, q_\gamma\rangle, q_\gamma)) &= \langle q_\sigma, q_\sigma\rangle \\
\delta'(\sigma(q_\alpha, \langle q_\beta, q_\gamma\rangle)) = \delta'(\sigma(q_\beta, \langle q_\beta, q_\gamma\rangle)) = \delta'(\sigma(q_\gamma, \langle q_\beta, q_\gamma\rangle)) &= \langle q_\sigma, q_\sigma\rangle \ .
\end{aligned}$$

For the sake of the next theorem, we assume that $M_\otimes$ is total to avoid a distinction between $\bot$ and undefinedness. It can easily be checked that the argument also works for partial dta.

Theorem 15. *$L(M_\otimes)_{\langle q,q'\rangle}$ is finite if and only if $q \sim q'$ for every $q, q' \in Q$.*

Proof. Let $M_\otimes = (Q', \Sigma, \delta', F')$. Clearly, $\delta'(c[\langle q, q'\rangle]) = \langle \delta(c[q]), \delta(c[q'])\rangle$ for every $c \in C_\Sigma$, which can be proven using standard induction. Now

$$\begin{aligned}
& c \in L(M_\otimes)_{\langle q,q'\rangle} \iff \delta'(c[\langle q, q'\rangle]) \in F' \iff \langle \delta(c[q]), \delta(c[q'])\rangle \in F' \\
\iff & \text{either } \delta(c[q]) \in F \text{ or } \delta(c[q']) \in F \iff c \in L(M)_q \ominus L(M)_{q'} \ .
\end{aligned}$$

This strong correspondence shows the statement because the finiteness of either set ($L(M_\otimes)_{\langle q,q'\rangle}$ or $L(M)_q \ominus L(M)_{q'}$) implies the finiteness of the other and $L(M)_q \ominus L(M)_{q'}$ is finite if and only if $q \sim q'$. □

The finiteness of $L(M_\otimes)_{\langle q,q'\rangle}$ for all states $\langle q, q'\rangle$ can be determined easily (using standard algorithms) in linear time in the number of transitions of $M_\otimes$. Since the number of transitions of $M_\otimes$ is $\mathcal{O}(\ell mn)$, we can obtain $\sim$ in time $\mathcal{O}(\ell mn)$.

Example 16. In the dta M_{ex} of Ex. 2 we have $q_\alpha \not\sim q_\beta$ as demonstrated by the recursive transitions for $\langle q_\alpha, q_\beta\rangle$ in Ex. 14. Moreso, $q_\beta \sim q_\gamma$ because the language $L((M_{\text{ex}})_\otimes)_{\langle q_\beta, q_\gamma\rangle}$ is finite (it contains only □).

Since we already proved that Alg. 1 is correct and have now established the run-time, we can state our main theorem.

Theorem 17. *Hyper-minimization of M can be performed in time $\mathcal{O}(\ell mn)$.*

Proof. We run Alg. 1, which runs in time $\mathcal{O}(\ell mn)$ because Line 3 runs in this time bound as demonstrated in this section. Finally, Cor. 9 proves the algorithm's correctness. □

5 Discussion

In this section, we shortly discuss two minor issues. First, we demonstrate that dta minimization can be reduced in linear time to dta hyper-minimization. In the string case, this is achieved [8] with a new distinguished symbol that takes every state back to the initial state, thus making all states kernel states. Since we do not have a single initial state in a dta, we use a slightly different construction. Let $M = (Q, \Sigma, \delta, F)$ be a dta that is not necessarily minimal. For every $q \in \delta(\Sigma_0)$, let $\overrightarrow{q} \notin \Sigma$ be a new symbol of rank 1. Moreover, we use the two new symbols $\to$ and $\circlearrowleft$, which are of rank 0 and 1, respectively, and a new state $\imath \notin Q$. We construct the dta $M' = (Q \cup \{\imath\}, \Sigma', \delta', F)$ such that

- $\Sigma' = \Sigma \cup \{\overrightarrow{q} \mid q \in \delta(\Sigma_0)\} \cup \{\to, \circlearrowleft\}$,
- $\delta'(t) = \delta(t)$ for all $t \in \mathrm{dom}(\delta)$,
- $\delta'(\to) = \imath$ and $\delta'(\circlearrowleft(\imath)) = \imath$, and
- $\delta'(\overrightarrow{q}(\imath)) = q$ for all $q \in \delta(\Sigma_0)$.
- All remaining transitions are undefined.

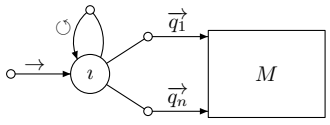

Fig. 3. Illustration.

Clearly, M' can be constructed in linear time in the size of M. The construction is illustrated in Fig. 3. Clearly, all reachable states in M' are kernel states. It is easy to see that a dta in which all reachable states are kernel states is hyper-minimal if and only if it is minimal. Consequently, we can hyper-minimize M' to obtain a minimal dta M'' for $L(M')$. From M'' we can obtain a minimal dta for $L(M)$ by dropping all transitions involving the newly introduced symbols. Thus, we have reduced minimization to hyper-minimization, which shows that the complexity of dta minimization is a lower bound on the complexity of hyper-minimization.

Second, we sketch an improved version of our hyper-minimization algorithm, which uses the structure of the fastest dfa hyper-minimization algorithms [4, 8]. First of all, we assume that M is total. We only present the computation of the almost equivalence because only this part needs to be improved to obtain the time bound $\mathcal{O}(\ell m \log n)$, which is also the time complexity of the fastest dta minimization algorithm [7]. Before we present the algorithm, we establish an auxiliary result.

Proposition 18. *Let M be minimal and $q, q' \in Q$. We have $q \sim q'$ if and only if for each context $c \in C_\Sigma(Q)$ we have*

- $\delta(c[q]) \sim \delta(c[q'])$, *and*
- $\delta(c[q]) = \delta(c[q'])$ *if* $\mathrm{states}(c) \cap \mathrm{Ker}(M) \neq \emptyset$.

Proof. The "only if" direction is a straightforward generalization of Lm. 3. For the converse, we simply take the trivial context $\Box$. $\Box$

Proposition 18 shows that we need a completely new mechanism (compared to the string case) to compute the successor states. We define the successor states, where we keep two dta: (i) the original dta M_0 to enforce the equality constraints of the second item of Prop. 18 and (ii) a dta M obtained by successive merges to capture the almost equivalence.

Algorithm 2 Algorithm computing $\sim$.

Require: minimal dta $M = (Q, \Sigma, \delta, F)$
Return: the almost equivalence $\sim$ represented as a partition

$M_0 \leftarrow M$ where $M_0 = (Q, \Sigma, \delta_0, F)$ // keep a copy of the input dta M
$\pi(q) \leftarrow \{q\}$ for all $q \in Q$ // trivial initial blocks
$h \leftarrow \emptyset$ // empty hash map of type $h\colon Q^C \to Q$
$I \leftarrow Q$; $P \leftarrow Q$ // states that need to be considered and current states
while $I \neq \emptyset$ **do**
  select $q \in I$ and remove it from I
  if $\textsc{HasValue}(h, \mathrm{succ}_q^{M,M_0})$ **then**
    $q' \leftarrow \textsc{Get}(h, \mathrm{succ}_q^{M,M_0})$ // retrieve state in bucket succ_q^{M,M_0} of h
    $\textsc{Swap}(q', q)$ if $|\pi(q')| \geq |\pi(q)|$ // exchange roles of q' and q
    $P \leftarrow P - \{q'\}$ // state q' will be merged into q
    $I \leftarrow I \cup \{r \in P \mid t \in \delta^{-1}(q'), r \in \mathrm{states}(t)\}$ // add predecessors of q' in P to I
    $M \leftarrow \mathrm{merge}'(M, q' \to q)$ // merge state q' into q (do not remove q')
    $\pi(q) \leftarrow \pi(q) \cup \pi(q')$ // q' and q are almost equivalent
  $h \leftarrow \textsc{Put}(h, \mathrm{succ}_q^{M,M_0}, q)$ // store q in h under key succ_q^{M,M_0}
return π

Definition 19. *Let $M_0 = (Q, \Sigma, \delta_0, F)$ and M be dta. For every state $q \in Q$, let $\mathrm{succ}_q^{M,M_0}\colon C \to Q$ be the mapping such that for every $c \in C$*

$$\mathrm{succ}_q^{M,M_0}(c) = \begin{cases} \delta_0(c[q]) & \text{if } \mathrm{states}(c) \cap \mathrm{Ker}(M) \neq \emptyset \\ \delta(c[q]) & \text{otherwise,} \end{cases}$$

where $C = \{\sigma(q_1, \ldots, q_{i-1}, \Box, q_{i+1}, \ldots, q_k) \mid \sigma \in \Sigma_k, i \in [k], q_1, \ldots, q_k \in Q\}$.

In other words, we compute with the original transition mapping δ_0 for all transition contexts containing a kernel state and use the current transition mapping δ for all other transition contexts. Let us attempt to explain Algorithm 2. Its overall structure is the same as in the string case [4, 8]. We only changed the details to suit the new needs in the dta case. Roughly speaking, the algorithm first copies the input dta in order to have the original transition mapping available. Then it creates a block for each state. In I it keeps a set of states that need to be processed, and in P it stores the set of states that are still useful. Both are initially Q and we also create a hash map h of type $h\colon Q^C \to Q$, which initially has no entries. Clearly, the key set of this hash map is highly complex. The algorithm iteratively extracts a state q from I and computes its successors succ_q^{M,M_0}. It then looks succ_q^{M,M_0} up in the hash-map h, and simply stores them in h if they are so far unassociated. If the successors already have an entry in h, then the algorithm extracts the state with the same successors from h, compares the sizes of their respective blocks, and merges the state q' belonging to the smaller block into the one belonging to the bigger block. We use a variant of our merging procedure here, which does not delete the state q'. It also updates the blocks to reflect the merge, and it adds all states that have transitions leading to q' to I

for processing because their successors have changed. The algorithm terminates when the set I is empty. The time complexity of this algorithm can be analyzed as in the string case [8]. Finally, its correctness still needs to be established.

References

1. Badr, A.: Hyper-minimization in $O(n^2)$. Int. J. Found. Comput. Sci. 20(4), 735–746 (2009)
2. Badr, A., Geffert, V., Shipman, I.: Hyper-minimizing minimized deterministic finite state automata. RAIRO Theor. Inf. Appl. 43(1), 69–94 (2009)
3. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata: Techniques and applications. Available on: `http://tata.gforge.inria.fr/` (2007)
4. Gawrychowski, P., Jeż, A.: Hyper-minimisation made efficient. In: Proc. 34th Int. Symp. Mathematical Foundations of Computer Science. LNCS, vol. 5734, pp. 356–368. Springer (2009)
5. Gécseg, F., Steinby, M.: Tree Automata. Akadémiai Kiadó, Budapest (1984)
6. Gécseg, F., Steinby, M.: Tree languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, chap. 1, pp. 1–68. Springer (1997)
7. Högberg, J., Maletti, A., May, J.: Backward and forward bisimulation minimization of tree automata. Theoret. Comput. Sci. 410(37), 3539–3552 (2009)
8. Holzer, M., Maletti, A.: An $n \log n$ algorithm for hyper-minimizing a (minimized) deterministic automaton. Theoret. Comput. Sci. 411(38–39), 3404–3413 (2010)
9. Hopcroft, J.E.: An $n \log n$ algorithm for minimizing states in a finite automaton. In: Kohavi, Z., Paz, A. (eds.) Theory of Machines and Computations, pp. 189–196. Academic Press (1971)
10. Hosoya, H.: Foundations of XML Processing: The Tree-Automata Approach. Cambridge University Press (2011)
11. Knight, K.: Capturing practical natural language transformations. Machine Translation 21(2), 121–133 (2007)
12. Maletti, A.: Notes on hyper-minimization. In: Proc. 13th Int. Conf. Automata and Formal Languages. pp. 34–49. Nyíregyháza College (2011)
13. Maletti, A., Quernheim, D.: Hyper-minimisation of deterministic weighted finite automata over semifields. In: Proc. 13th Int. Conf. Automata and Formal Languages. pp. 285–299. Nyíregyháza College (2011)
14. Maletti, A., Quernheim, D.: Optimal hyper-minimization. Int. J. Found. Comput. Sci. 22(8), 1877–1891 (2011)
15. Schewe, S.: Beyond hyper-minimisation — minimising DBAs and DPAs is NP-complete. In: Proc. 30th Int. Conf. Foundations of Software Technology and Theoretical Computer Science. LIPIcs, vol. 8, pp. 400–411. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2010)
16. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. Comput. 1(2), 146–160 (1972)
17. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 1, chap. 2, pp. 41–110. Springer (1997)