# UNWEIGHTED AND WEIGHTED HYPER-MINIMIZATION*

ANDREAS MALETTI† and DANIEL QUERNHEIM†

*Universität Stuttgart, Institute for Natural Language Processing*
*Pfaffenwaldring 5b, 70569 Stuttgart, Germany*
{andreas.maletti & daniel.quernheim}@ims.uni-stuttgart.de

Received (15 December 2011)
Revised (20 June 2012)
Accepted (30 June 2012)
Communicated by (Pál Dömösi and Zoltán Ésik)

Hyper-minimization of deterministic finite automata (DFA) is a recently introduced state reduction technique that allows a finite change in the recognized language. A generalization of this lossy compression method to the weighted setting over semifields is presented, which allows the recognized weighted language to differ for finitely many input strings. First, the structure of hyper-minimal deterministic weighted finite automata is characterized in a similar way as in classical weighted minimization and unweighted hyper-minimization. Second, an efficient hyper-minimization algorithm, which runs in time $\mathcal{O}(n \log n)$, is derived from this characterization. Third, the closure properties of canonical regular languages, which are languages recognized by hyper-minimal DFA, are investigated. Finally, some recent results in the area of hyper-minimization are recalled.

## 1. Introduction

Deterministic finite automata (DFA) [44] are one of the simplest, but most useful devices in computer science. Their simplicity and the availability of efficient manipulation software [32, 1] makes them attractive in many application areas such as speech processing [37], image compression [11], morphology [4] and linguistic analysis [28], natural language semantics [15], and pattern matching [10]. Often huge DFA consisting of several million states are required. Fortunately, every DFA admits an efficiently computable and unique (up to isomorphism) equivalent minimal DFA. Virtually every finite-state toolkit implements minimization, which is the process of computing such an equivalent minimal DFA. The asymptotically most efficient algorithm [24, 19, 42] for general DFA minimization computes the equivalent states and merges them in time $\mathcal{O}(n \log n)$, where $n$ is the number of states of the input DFA.

*This article is an extended and revised version of [Maletti, Quernheim: *Hyper-minimisation of deterministic weighted finite automata over semifields.* pp. 285–299] and [Maletti: *Notes on hyper-minimization.* pp. 34–49], which both appeared in the proceedings of the *13th International Conference on Automata and Formal Languages.* Nyíregyháza College, 2011.
†Both authors were supported by the German Research Foundation (DFG) grant MA / 4959 / 1–1.

Recently, hyper-minimization of DFA [3] has been proposed, which is a means of state reduction beyond the usual notion of the minimal DFA. Thus, it can potentially compress DFA even further at the expense of a finite change in the recognized language. The asymptotically fastest hyper-minimization algorithms [16, 23] compute the "almost-equivalence" relation and merge states with finite left language according to it in time $\mathcal{O}(n \log n)$. Variations such as cover automata minimization [8], which has been explored before hyper-minimization due to its usefulness in compressing finite languages, or $k$-minimization [16] restrict the length of the error strings instead of their number.

We will generalize hyper-minimization to the weighted setting. Our weight structures will be commutative semifields, which are commutative semirings [21, 18] with multiplicative inverses. As before, we will restrict our attention to deterministic automata. Actually, the mentioned applications of DFA often use the weighted version to compute a quantitative answer. For weighted deterministic finite automata (WDFA) [39] over semifields, similar results are known. They can be efficiently minimized, but the minimal equivalent WDFA is no longer unique due to ability to "push" weights [37, 14]. The asymptotically fastest minimization algorithms [37, 14] nevertheless still run in time $\mathcal{O}(n \log n)$. Essentially, they normalize the input WDFA by "pushing" weights towards the initial state. In the process, the signatures of equivalent states become equivalent, so that a classical unweighted minimization can then perform the computation of the equivalence and the merges.

We focus on the notion that allows the recognized weighted languages to differ for finitely many input strings. More sophisticated notions that are based on differences between the weights of strings are conceivable [38], yet the notion we consider has the benefit that it is simple, but realistic enough. We will join unweighted hyper-minimization and weighted minimization to weighted hyper-minimization. Our algorithms (see Algorithms 1 and 2) contain features of both of their predecessors and are asymptotically as efficient as them because they also run in time $\mathcal{O}(n \log n)$. In contrast to [38], we introduce standardized signatures to avoid the explicit pushing of weights. This adjustment allows us to mold our weighted hyper-minimization algorithm into the structure of the unweighted algorithm [23].

Finally, we provide an extensive review of recent progress in the area of lossy compression for DFA, which includes the generalization from a finite difference to a regular difference [22]. In particular, we study canonical languages [3], which are the languages accepted by hyper-minimal DFA, and answer an open question of [3]. We will also review how to optimize the obtained hyper-minimal DFA with respect to particular secondary criteria. Such an optimization is possible because there is no unique hyper-minimal DFA for a given input language. We recall from the literature that we can optimize the number of errors or the length of the longest error string easily, but that if we want to optimize ratios (for example, saved states vs. errors made), then the problem becomes intractable. We close the review with an application of cover automata minimization and $k$-minimization, which can be combined to compute a 'finite-factored DFA' in [2]. Such a DFA can exactly represent certain languages much more succinctly. We show that the critical step of the length bound selection can be done automatically without any (asymptotic) overhead.

## 2. Preliminaries

The set of all nonnegative integers is $\mathbb{N}$. The symmetric difference $S \ominus T$ of two sets $S$ and $T$ is $S \ominus T = (S - T) \cup (T - S)$. An alphabet $\Sigma$ is simply a finite set, and $\Sigma^*$ is the set of all strings over it including $\varepsilon$, which is the empty string. The length $|s|$ of a string $s = \sigma_1 \cdots \sigma_\ell$ with $\sigma_1, \dots, \sigma_\ell \in \Sigma$ is $|s| = \ell$. The sets $\Sigma^{\leq \ell}$ and $\Sigma^{\geq \ell}$ contain all strings over $\Sigma$ of length at most $\ell$ and at least $\ell$, respectively. Concatenation of strings is simply denoted by juxtaposition. A language $L$ over $\Sigma$ is a subset $L \subseteq \Sigma^*$.

Our weights are taken from a commutative semifield $\langle \mathbb{K}, +, \cdot, 0, 1 \rangle$, which means that $\langle \mathbb{K}, +, 0 \rangle$ and $\langle \mathbb{K}, \cdot, 1 \rangle$ are commutative monoids, the multiplication $\cdot$ distributes over finite sums (including the empty sum), and for every $k \in \mathbb{K} - \{0\}$ there exists $k^{-1} \in \mathbb{K}$ such that $k \cdot k^{-1} = 1$. In other words, commutative semifields are commutative semirings [21, 18] with multiplicative inverses. Useful commutative semifields include the real numbers $\langle \mathbb{R}, +, \cdot, 0, 1 \rangle$, the tropical semifield $\langle \mathbb{R} \cup \{\infty\}, \min, +, \infty, 0 \rangle$, the probabilistic semifield $\langle [0, 1], \max, \cdot, 0, 1 \rangle$ with $[0, 1] = \{k \in \mathbb{R} \mid 0 \leq k \leq 1\}$, and the BOOLEAN semifield $\mathbb{B} = \langle \{0, 1\}, \max, \min, 0, 1 \rangle$. From now on, let $\langle \mathbb{K}, +, \cdot, 0, 1 \rangle$ be a commutative semifield with $0 \neq 1$, and let $\underline{\mathbb{K}} = \mathbb{K} - \{0\}$. We will sometimes write $\frac{k_1}{k_2}$ instead of $k_1 \cdot k_2^{-1}$.

A weighted language is a mapping $\varphi \colon \Sigma^* \to \mathbb{K}$, whose support $\operatorname{supp}(\varphi) \subseteq \Sigma^*$ is $\operatorname{supp}(\varphi) = \varphi^{-1}(\underline{\mathbb{K}})$. Given $k \in \mathbb{K}$, we let $(k \cdot \varphi) \colon \Sigma^* \to \mathbb{K}$ be the weighted language such that $(k \cdot \varphi)(s) = k \cdot \varphi(s)$ for every $s \in \Sigma^*$. A weighted deterministic finite automaton (WDFA) [40, 31] is a tuple $\mathcal{A} = (Q, \Sigma, q_0, k_0, \delta, \mathrm{wt}, F)$, in which $Q$ is a finite set of states, $\Sigma$ is an alphabet of input symbols, $q_0 \in Q$ is an initial state, $k_0 \in \underline{\mathbb{K}}$ is an initial weight, $\delta \colon Q \times \Sigma \to Q$ is a transition mapping, $\mathrm{wt} \colon Q \times \Sigma \to \underline{\mathbb{K}}$ a transition weight assignment, and $F \subseteq Q$ is a set of final states. The transition and transition weight mappings '$\delta$' and 'wt' extend to mappings $\delta \colon Q \times \Sigma^* \to Q$ and $\mathrm{wt} \colon Q \times \Sigma^* \to \underline{\mathbb{K}}$ by $\delta(q, \varepsilon) = q$ and $\mathrm{wt}(q, \varepsilon) = 1$, and $\delta(q, \sigma s) = \delta(\delta(q, \sigma), s)$ and $\mathrm{wt}(q, \sigma s) = \mathrm{wt}(q, \sigma) \cdot \mathrm{wt}(\delta(q, \sigma), s)$ for every $q \in Q$, $\sigma \in \Sigma$, and $s \in \Sigma^*$. We simply write $\delta(s)$ and $\mathrm{wt}(s)$ for $\delta(q_0, s)$ and $\mathrm{wt}(q_0, s)$, respectively. Next, we define the $q$-semantics $[\![q]\!]_{\mathcal{A}} \colon \Sigma^* \to \mathbb{K}$ of $\mathcal{A}$ for every $q \in Q$. Formally, for every $q \in Q$ and $s \in \Sigma^*$, let $[\![q]\!]_{\mathcal{A}}(s) = \mathrm{wt}(q, s)$ if $\delta(q, s) \in F$ and $[\![q]\!]_{\mathcal{A}}(s) = 0$ otherwise. Intuitively, $[\![q]\!]_{\mathcal{A}}$ is the weighted language recognized by $\mathcal{A}$ starting in state $q$ (with initial weight 1). The WDFA $\mathcal{A}$ recognizes the weighted language $[\![\mathcal{A}]\!] = k_0 \cdot [\![q_0]\!]_{\mathcal{A}}$. Two WDFA are equivalent if their recognized weighted languages coincide. A WDFA over the BOOLEAN semifield $\mathbb{B}$ is also called DFA [44] and written $(Q, \Sigma, q_0, \delta, F)$ because the components '$k_0$' and 'wt' are uniquely determined. Moreover, we identify each BOOLEAN weighted language $\varphi \colon \Sigma^* \to \{0, 1\}$ with its support.

Two states $q, q' \in Q$ are equivalent [5], written $q \equiv q'$, if there exists $k \in \underline{\mathbb{K}}$ such that $[\![q]\!]_{\mathcal{A}} = k \cdot [\![q']\!]_{\mathcal{A}}$. An equivalence relation $\cong \subseteq Q \times Q$ is a congruence relation if $\delta(q,\sigma) \cong \delta(q',\sigma)$ for all $\sigma \in \Sigma$ and $q \cong q'$. Note that $\equiv$ is a congruence relation. The WDFA $\mathcal{A}$ is minimal if there is no equivalent WDFA with strictly fewer states. We can compute a minimal WDFA efficiently using a variant of HOPCROFT's algorithm [24, 37, 14] that computes $\equiv$ and runs in time $\mathcal{O}(n \log n)$ where $n = |Q|$.

## 3. A Characterization of Hyper-Minimality

Hyper-minimization [3] of WDFA is a form of lossy compression that allows any finite number of errors. It has been investigated in [3] for the BOOLEAN semifield. We generalize their results to our weighted setting.

*Let $\mathcal{A} = (Q, \Sigma, q_0, k_0, \delta, \mathrm{wt}, F)$ and $\mathcal{B} = (P, \Sigma, p_0, k'_0, \mu, \mathrm{wt}', G)$ be* WDFA *over the commutative semifield $\langle \mathbb{K}, +, \cdot, 0, 1\rangle$ with $0 \neq 1$.*

**Definition 1.** *Two weighted languages $\varphi_1, \varphi_2 \colon \Sigma^* \to \mathbb{K}$ are* almost-equivalent, *written $\varphi_1 \approx \varphi_2$, if there exists $k \in \underline{\mathbb{K}}$ such that $\varphi_1(s) = k \cdot \varphi_2(s)$ for almost all $s \in \Sigma^*$. We write $\varphi_1 \approx \varphi_2$ $(k)$ to indicate the factor $k$. The* WDFA *$\mathcal{A}$ and $\mathcal{B}$ are almost-equivalent if $[\![\mathcal{A}]\!] \approx [\![\mathcal{B}]\!]$ $(1)$. Finally, the states $q \in Q$ and $p \in P$ are almost-equivalent if $[\![q]\!]_{\mathcal{A}} \approx [\![p]\!]_{\mathcal{B}}$.*

First, we show some basic properties of $\approx$. In particular, we show that $\approx \subseteq Q \times Q$ is a congruence relation in a way similar to Lemma 2.10 in [3].

**Lemma 2.** *Almost-equivalence is an equivalence relation such that $\delta(q,s) \approx \mu(p,s)$ for all $s \in \Sigma^*$, $q \in Q$, and $p \in P$ with $q \approx p$.*

**Proof.** Trivially, $\approx$ is reflexive and symmetric. Let $\varphi_1, \varphi_2, \varphi_3 \colon \Sigma^* \to \mathbb{K}$ be weighted languages such that $\varphi_1 \approx \varphi_2$ $(k)$ and $\varphi_2 \approx \varphi_3$ $(k')$. Then there exist finite sets $L$ and $L'$ such that $\varphi_1(s) = k \cdot \varphi_2(s)$ and $\varphi_2(s') = k' \cdot \varphi_3(s')$ for all $s \in \Sigma^* - L$ and $s' \in \Sigma^* - L'$. Consequently, $\varphi_1(s'') = k \cdot k' \cdot \varphi_3(s'')$ for all $s'' \in \Sigma^* - (L \cup L')$, which proves $\varphi_1 \approx \varphi_3$ $(k \cdot k')$. The same arguments can be used for $\approx$ on WDFA and states.

For the second property, induction allows us to easily prove

$$[\![q]\!]_{\mathcal{A}}(ss') = \mathrm{wt}(q,s) \cdot [\![\delta(q,s)]\!]_{\mathcal{A}}(s') \qquad \text{and} \qquad (\dagger)$$
$$[\![p]\!]_{\mathcal{B}}(ss') = \mathrm{wt}'(p,s) \cdot [\![\mu(p,s)]\!]_{\mathcal{B}}(s')$$

for all $s, s' \in \Sigma^*$. Since $q \approx p$ $(k)$, there exists a finite set $L \subseteq \Sigma^*$ such that $[\![q]\!]_{\mathcal{A}}(s'') = k \cdot [\![p]\!]_{\mathcal{B}}(s'')$ for all $s'' \in \Sigma^* - L$. Consequently,

$$[\![\delta(q,s)]\!]_{\mathcal{A}}(s') = \frac{[\![q]\!]_{\mathcal{A}}(ss')}{\mathrm{wt}(q,s)} = k \cdot \frac{[\![p]\!]_{\mathcal{B}}(ss')}{\mathrm{wt}(q,s)} = k \cdot \frac{\mathrm{wt}'(p,s)}{\mathrm{wt}(q,s)} \cdot [\![\mu(p,s)]\!]_{\mathcal{B}}(s')$$

for all $s' \in \Sigma^*$ such that $ss' \notin L$, which proves that $\delta(q,s) \approx \mu(p,s)$. □

Given the WDFA $\mathcal{A}$, the goal of hyper-minimization is to construct an almost-equivalent WDFA $\mathcal{B}$ such that no WDFA is smaller than $\mathcal{B}$ and almost-equivalent

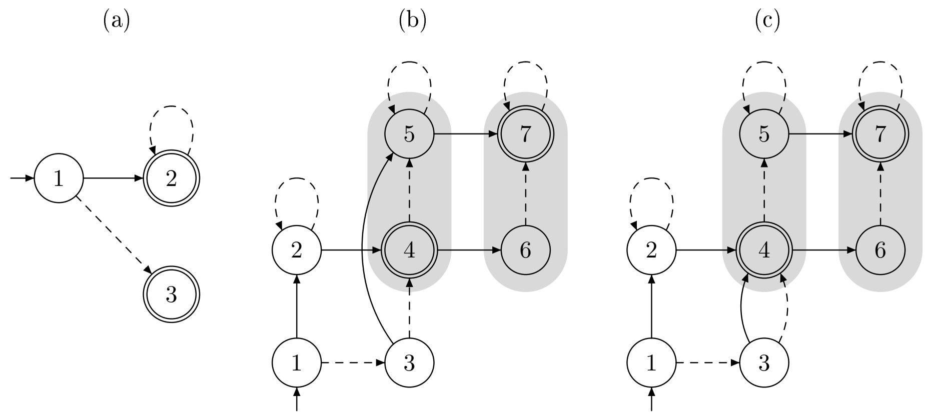

Fig. 1. Three hyper-minimal DFA with indicated almost-equivalence.

to $\mathcal{A}$. Since almost-equivalence is an equivalence relation, we can replace the requirement "almost-equivalent to $\mathcal{A}$" by "almost-equivalent to $\mathcal{B}$" and call a WDFA $\mathcal{B}$ *hyper-minimal* if no (strictly) smaller WDFA is almost-equivalent to it. Then hyper-minimization is the computation of a hyper-minimal WDFA $\mathcal{B}$ that is almost-equivalent to $\mathcal{A}$. Let us first investigate hyper-minimality, which was characterized in [3] for the BOOLEAN semifield using the additional notion of a *preamble* state.

**Definition 3 (Definition 2.11 in [3])** *A state $q \in Q$ of $\mathcal{A}$ is a* preamble state *if $\delta^{-1}(q) \subseteq \Sigma^*$ is finite. Otherwise, it is a* kernel state.

**Example 4.** *Figure 1 shows three* DFA*, in which we marked almost-equivalence of states. States 4 and 5 in the* DFA *(b) and (c) are almost-equivalent because*

$$\llbracket 4 \rrbracket = \{\varepsilon\} \cup \{\dashrightarrow^m \longrightarrow \dashrightarrow^n \mid m + n \geq 1\}$$
$$\llbracket 5 \rrbracket = \{\dashrightarrow^m \longrightarrow \dashrightarrow^n \mid m, n \in \mathbb{N}\} \ ,$$

*which yields that $\llbracket 4 \rrbracket \ominus \llbracket 5 \rrbracket = \{\varepsilon, \longrightarrow\}$. In fact, the* DFA *(b) and (c) are almost-equivalent. The preamble states of all three* DFA *are $\{1, 3\}$.*

Recall that a WDFA (without unreachable states; i.e., $\delta^{-1}(q) \cap \Sigma^* \neq \emptyset$ for every $q \in Q$) is minimal if and only if it does not have a pair of different, but equivalent states [25]. The "only-if" part of this statement is shown by merging two equivalent states to obtain a smaller, but equivalent WDFA.

**Definition 5.** *Let $q, q' \in Q$ and $k \in \underline{\mathbb{K}}$ with $q \neq q'$. The $k$-*weighted merge *of $q$ into $q'$ is the* WDFA $\text{merge}_{\mathcal{A}}(q \xrightarrow{k} q') = (Q - \{q\}, \Sigma, q_0', k_0', \delta', \text{wt}', F - \{q\})$ *such that for all $r \in Q$ and $\sigma \in \Sigma$*

$$q_0' = \begin{cases} q' & \text{if } q_0 = q \\ q_0 & \text{otherwise} \end{cases} \qquad k_0' = \begin{cases} k \cdot k_0 & \text{if } q_0 = q \\ k_0 & \text{otherwise} \end{cases}$$

$$\delta'(r,\sigma) = \begin{cases} q' & \textit{if } \delta(r,\sigma) = q \\ \delta(r,\sigma) & \textit{otherwise} \end{cases} \qquad \mathrm{wt}'(r,\sigma) = \begin{cases} k \cdot \mathrm{wt}(r,\sigma) & \textit{if } \delta(r,\sigma) = q \\ \mathrm{wt}(r,\sigma) & \textit{otherwise.} \end{cases}$$

The 2-weighted merge of $F$ into $E$ in the WDFA $\mathcal{C}$ of Figure 2 (left) yields the WDFA $\mathrm{merge}_{\mathcal{C}}(F \xrightarrow{2} E)$ that is displayed in Figure 2 (right).

**Lemma 6.** *Let $q, q' \in Q$ be different states, of which $q$ is a preamble state, and $k \in \underline{\mathbb{K}}$ be such that $q \approx q'$ $(k)$. Then $\mathrm{merge}_{\mathcal{A}}(q \xrightarrow{k} q')$ is almost-equivalent to $\mathcal{A}$.*

**Proof.** Let $\mathcal{B} = \mathrm{merge}(q \xrightarrow{k} q')$. Clearly, we have $[\![\mathcal{B}]\!](s) = [\![\mathcal{A}]\!](s)$ for all $s \in \Sigma^*$ such that $\delta(s') \neq q$ for all prefixes $s'$ of $s$ (i.e., $s = s's''$ for some $s'' \in \Sigma^*$). This is simply due to the fact that $\mathcal{B}$ faithfully replicates the behavior of $\mathcal{A}$ in this case. Moreover, $\delta(q', s) \neq q$ for all $s \in \Sigma^*$ by a simple variation of Lemma 2.14 in [3], which proves that $[\![q']\!]_{\mathcal{B}} = [\![q']\!]_{\mathcal{A}}$. Now let $s \in \delta^{-1}(q)$. By assumption, $[\![q]\!]_{\mathcal{A}}(s') = k \cdot [\![q']\!]_{\mathcal{A}}(s')$ for almost all $s' \in \Sigma^*$. We obtain

$$\begin{aligned} [\![\mathcal{B}]\!](ss') &\overset{\dagger}{=} \big(k'_0 \cdot \mathrm{wt}'(s)\big) \cdot [\![q']\!]_{\mathcal{B}}(s') = \big(k_0 \cdot \mathrm{wt}(s) \cdot k\big) \cdot \big(k^{-1} \cdot [\![q]\!]_{\mathcal{A}}(s')\big) \\ &= k_0 \cdot \mathrm{wt}(s) \cdot [\![q]\!]_{\mathcal{A}}(s') \quad \overset{\dagger}{=} [\![\mathcal{A}]\!](ss') \ . \end{aligned}$$

Thus, $[\![\mathcal{B}]\!]$ and $[\![\mathcal{A}]\!]$ coincide for almost all strings with prefix $s$. Since $q$ is a preamble state, $\delta^{-1}(q)$ is finite, which yields that $\mathcal{B}$ and $\mathcal{A}$ are almost-equivalent. □

Lemma 6 shows that the WDFA of Figure 2 (right) is not hyper-minimal because the states $C$ and $D$ are almost-equivalent and $D$ is a preamble state. Moreover, it also shows that the WDFA of Figure 2 are almost-equivalent. Finally, we still need to show that the merging process indeed yields a hyper-minimal WDFA to obtain a characterization of hyper-minimal WDFA in the spirit of Theorem 3.4 in [3]. For example, Figure 3 shows a hyper-minimal WDFA that is almost-equivalent to the WDFA of Figure 2 and can be obtained by a 1-weighted merge of $D$ into $C$.

**Theorem 7.** *A minimal* WDFA *is hyper-minimal if and only if it has no pair of different, but almost-equivalent states, of which at least one is a preamble state.*

**Proof.** Let $\mathcal{A}$ be the minimal WDFA. For the "only if" part, we know by Lemma 6 that the smaller WDFA $\mathrm{merge}(q \xrightarrow{k} q')$ is almost-equivalent to $\mathcal{A}$ if $q \approx q'$ $(k)$ and $q$ is a preamble state. For the "if" direction, suppose that $\mathcal{B}$ is almost-equivalent to $\mathcal{A}$ (i.e., $q_0 \approx p_0$) and $|P| < |Q|$. For all $s \in \Sigma^*$ we have $\delta(s) \approx \mu(s)$ by Lemma 2. Since $|P| < |Q|$, there exist $s_1, s_2 \in \Sigma^*$ with $q_1 = \delta(s_1) \neq \delta(s_2) = q_2$ but $\mu(s_1) = p = \mu(s_2)$. Consequently, $q_1 = \delta(s_1) \approx \mu(s_1) = p = \mu(s_2) \approx \delta(s_2) = q_2$, which yields $q_1 \approx q_2$. By assumption, $q_1$ and $q_2$ are kernel states. Using a variation of the above argument (see Theorem 3.3 in [3]) we can obtain $s_1$ and $s_2$ with the above properties such that $|s_1|, |s_2| \geq |Q|^2$.

Let $s_1 = \sigma_1 \cdots \sigma_\ell$ with $\sigma_1, \ldots, \sigma_\ell \in \Sigma$. If we run $\mathcal{A}$ and $\mathcal{B}$ on the prefixes of $s_1$, then we obtain pairs of states $\langle q_i, p_i \rangle = \langle \delta(\sigma_1 \cdots \sigma_i), \mu(\sigma_1 \cdots \sigma_i) \rangle$ for every $1 \leq i \leq \ell$.

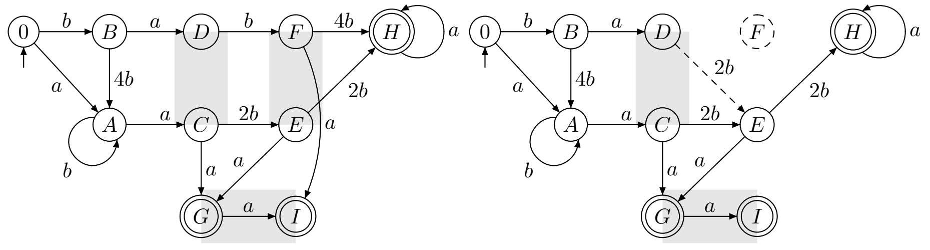

Fig. 2. WDFA over the real numbers [left] and the WDFA obtained by merge($F \xrightarrow{2} E$) [right]. Where no weight is indicated, the multiplicative identity 1 is implicit. Omitted transitions lead to the sink state $\bot$. The dashed line indicates the rerouted transition.

By assumption $\ell \geq |Q|^2 > |Q| \cdot |P|$. Consequently, there are indices $1 \leq i < j \leq \ell$ such that $\langle q_i, p_i \rangle = \langle q_j, p_j \rangle$. Thus, we obtain that there are infinitely many strings in both $L(q_1, p) = \delta^{-1}(q_1) \cap \mu^{-1}(p)$ and $L(q_2, p) = \delta^{-1}(q_2) \cap \mu^{-1}(p)$, where we used the same argument for $q_2$ to obtain the latter statement. Since $\mathcal{A}$ and $\mathcal{B}$ are almost-equivalent, we have

$$k_0 \cdot \mathrm{wt}(s'_1) \cdot [\![q_1]\!]_{\mathcal{A}}(s') = [\![\mathcal{A}]\!](s'_1 s') = [\![\mathcal{B}]\!](s'_1 s') = k'_0 \cdot \mathrm{wt}'(s'_1) \cdot [\![p]\!]_{\mathcal{B}}(s')$$
$$k_0 \cdot \mathrm{wt}(s'_2) \cdot [\![q_2]\!]_{\mathcal{A}}(s') = [\![\mathcal{A}]\!](s'_2 s') = [\![\mathcal{B}]\!](s'_2 s') = k'_0 \cdot \mathrm{wt}'(s'_2) \cdot [\![p]\!]_{\mathcal{B}}(s')$$

for almost all $s'_1 s', s'_2 s' \in \Sigma^*$ with $s'_1 \in L(q_1, p)$ and $s'_2 \in L(q_2, p)$. In fact, we can select $s_1 \in L(q_1, p)$ and $s_2 \in L(q_2, p)$ such that the previous two equations hold for all $s' \in \Sigma^*$ because $L(q_1, p)$ and $L(q_2, p)$ are infinite. Consequently,

$$\frac{k_0 \cdot \mathrm{wt}(s_1) \cdot [\![q_1]\!]_{\mathcal{A}}(s')}{k'_0 \cdot \mathrm{wt}'(s_1)} = \frac{k_0 \cdot \mathrm{wt}(s_2) \cdot [\![q_2]\!]_{\mathcal{A}}(s')}{k'_0 \cdot \mathrm{wt}'(s_2)} \quad \text{and} \quad [\![q_1]\!]_{\mathcal{A}}(s') = k \cdot [\![q_2]\!]_{\mathcal{A}}(s')$$

for all $s' \in \Sigma^*$ and $k = \frac{\mathrm{wt}'(s_1) \cdot \mathrm{wt}(s_2)}{\mathrm{wt}'(s_2) \cdot \mathrm{wt}(s_1)}$, which yields $q_1 \equiv q_2$. This contradicts minimality since $q_1 \neq q_2$, which shows that such a WDFA $\mathcal{B}$ cannot exist. $\square$

## 4. Hyper-Minimization

*Let $P$ and $K$ be the sets of preamble and kernel states of $\mathcal{A}$. We assume an arbitrary, but fixed total order on $\Sigma$.*

In this section, we consider hyper-minimization of unweighted and weighted DFA. Since the unweighted case is already well-described in the literature [3, 2, 16, 23, 35] we focus on weighted hyper-minimization, for which we need co-preamble states, which are the preamble states of the reversed automaton. Since the reversed automaton is not necessarily deterministic, we give an equivalent formal definition.

**Definition 8.** *A state $q \in Q$ is a* co-preamble state *if* $\mathrm{supp}([\![q]\!]_{\mathcal{A}})$ *is finite. Otherwise it is a* co-kernel state. *The sets of all co-preamble states and all co-kernel states are $\overline{P}$ and $\overline{K} = Q - \overline{P}$, respectively.*

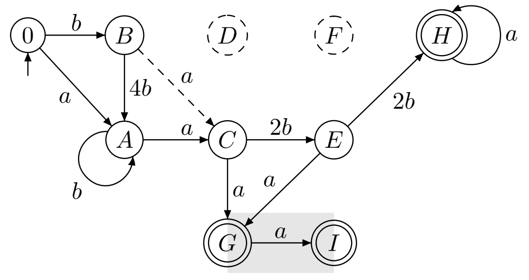

Fig. 3. Hyper-minimal WDFA. The dashed line indicates the rerouted transition.

On our example WDFA of Figure 2 (left), we observe that $\{G, I\}$ are co-preamble states and all remaining states are co-kernel states. Transitions entering a co-preamble state can be ignored while checking almost-equivalence because (up to a finite number of weight differences) the reached states behave like the sink state $\perp$. Trivially, all co-preamble states are almost-equivalent. In addition, a co-preamble state cannot be almost-equivalent to a co-kernel state. The interesting part of the almost-equivalence is thus completely determined by the weighted languages of the co-kernel states. This special role of the co-preamble states has already been pointed out in [16] in the context of DFA.

All hyper-minimization algorithms [3, 2, 16, 23] share the same overall structure (Algorithm 1). In the final step we perform state merges (see Definition 5). Merging only preamble states into almost-equivalent states makes sure that the resulting WDFA is almost-equivalent to the input WDFA by Lemma 6.

Algorithm 1 first minimizes the input WDFA using, for example, EISNER's algorithm [14]. With the help of a weight redistribution along the transitions (i.e., pushing), it reduces the problem to DFA minimization, for which we can use HOPCROFT's algorithm [24]. In the next step, we compute the set $K$ of kernel states of $\mathcal{A}$ using any algorithm that computes strongly connected components (for example, TARJAN's algorithm [41]). By [16, 23] a state is a kernel state if and only if it is reachable from (i) a nontrivial strongly connected component or (ii) a state with a self-loop. Since the co-kernel is the kernel of the reversed WDFA, we can use the same algorithm to compute the co-kernel states. In line 4 we compute the almost-equivalence on the states $Q$, which is the part where the algorithms [3, 2, 16, 23] differ. Finally, we merge almost-equivalent states according to Lemma 6 until the obtained WDFA is hyper-minimal (see Theorem 7).

We generally assume a RAM [43] (random access machine) as our computational model and a fixed alphabet $\Sigma$. computing the almost-equivalence on DFA runs in time $\mathcal{O}(n^3)$, where $n = |Q|$. It was improved in [2] to run in time $\mathcal{O}(n^2)$. Finally, [16, 23] independently improved the bound to $\mathcal{O}(n \log n)$, which coincides with the well-known bound for classical DFA minimization [24]. All those algorithms use the following observation (see Definition 2.2 in [3]), which we immediately present in

**Algorithm 1** Structure of the hyper-minimization algorithm.

**Require:** a WDFA $\mathcal{A}$ with $n$ states

**Return:** an almost-equivalent hyper-minimal WDFA

```
   $\mathcal{A} \leftarrow$ MINIMIZE($\mathcal{A}$)          // HOPCROFT's or EISNER's algorithm; $\mathcal{O}(n \log n)$
2: $K \leftarrow$ COMPUTEKERNEL($\mathcal{A}$)                          // TARJAN's algorithm; $\mathcal{O}(n)$
   $\overline{K} \leftarrow$ COMPUTECOKERNEL($\mathcal{A}$)                        // TARJAN's algorithm; $\mathcal{O}(n)$
4: $(\sim, t) \leftarrow$ COMPUTEALMOSTEQUIVALENCE($\mathcal{A}, \overline{K}$)      // Algorithm 2; $\mathcal{O}(n \log n)$
   return MERGESTATES($\mathcal{A}, K, \sim, t$)                          // Algorithm 3; $\mathcal{O}(n)$
```

the weighted setting.

**Lemma 9.** *Let $\mathcal{A}$ be a minimal* WDFA*. The states $q, q' \in Q$ are almost-equivalent if and only if there is $k \in \mathbb{N}$ such that $\delta(q, s) = \delta(q', s)$ for all $s \in \Sigma^*$ with $|s| \geq k$.*

Our algorithm for computing the almost-equivalence is a modification of the algorithm of [23]. However, we need to handle the scaling factors, for which we introduce another notion. Roughly speaking, the algorithm of [23] just compared the signatures of states. In the weighted setting, we first need to standardize the signature to account for the scaling factor. To this end, we ignore transitions into co-preamble states and normalize the transition weights.

**Definition 10.** *The* standardized signature $\mathrm{sig}_q \colon \Sigma \to Q \times \underline{\mathbb{K}}$ *of $q \in Q$ is such that for every $\sigma \in \Sigma$:*

- *If $\delta(q, \sigma) \in \overline{P}$, then $\mathrm{sig}_q(\sigma) = \langle \bot, 1 \rangle$.*
- *Otherwise, let $\sigma_0 \in \Sigma$ be the smallest symbol such that $\delta(q, \sigma_0) \in \overline{K}$. Then $\mathrm{sig}_q(\sigma) = \langle \delta(q, \sigma), \frac{\mathrm{wt}(q,\sigma)}{\mathrm{wt}(q,\sigma_0)} \rangle$.*

For the example WDFA of Figure 2 (left) we obtain $\mathrm{sig}_E(a) = \langle \bot, 1 \rangle$ and $\mathrm{sig}_E(b) = \langle H, 0 \rangle$, which coincides with $\mathrm{sig}_F$. Next, we show that states with equal standardized signature are indeed almost-equivalent.

**Lemma 11.** *Let $q, q' \in Q$. If $\mathrm{sig}_q = \mathrm{sig}_{q'}$, then $q \approx q'$.*

**Proof.** If $q$ or $q'$ is a co-preamble state, then both $q$ and $q'$ are co-preamble states and thus $q \approx q'$. Now, let $q, q' \in \overline{K}$, and let $k = \frac{\mathrm{wt}(q,\sigma_0)}{\mathrm{wt}(q',\sigma_0)}$, where $\sigma_0$ is the smallest symbol such that $\delta(q, \sigma_0) \in \overline{K}$. For every $\sigma \in \Sigma$ and $s \in \Sigma^*$,

$$[\![q]\!]_{\mathcal{A}}(\sigma s) = \mathrm{wt}(q, \sigma) \cdot [\![\delta(q, \sigma)]\!]_{\mathcal{A}}(s) \quad \text{and} \quad [\![q']\!]_{\mathcal{A}}(\sigma s) = \mathrm{wt}(q', \sigma) \cdot [\![\delta(q', \sigma)]\!]_{\mathcal{A}}(s) \ .$$

Further, let $\mathrm{sig}_q(\sigma) = \langle q_\sigma, k_\sigma \rangle = \mathrm{sig}_{q'}(\sigma)$. If $q_\sigma = \bot$, then $[\![q]\!]_{\mathcal{A}}(\sigma s) = k \cdot [\![q']\!]_{\mathcal{A}}(\sigma s)$ for almost all $s \in \Sigma^*$. Otherwise, we obviously have $\delta(q, \sigma) = q_\sigma = \delta(q', \sigma)$, and we obtain

$$\begin{aligned}[\![q]\!]_{\mathcal{A}}(\sigma s) &= k_\sigma \cdot \mathrm{wt}(q, \sigma_0) \cdot [\![q_\sigma]\!]_{\mathcal{A}}(s) \\ &= \frac{\mathrm{wt}(q', \sigma)}{\mathrm{wt}(q', \sigma_0)} \cdot \mathrm{wt}(q, \sigma_0) \cdot [\![q_a]\!]_{\mathcal{A}}(s) = k \cdot [\![q']\!]_{\mathcal{A}}(\sigma s)\end{aligned}$$

**Algorithm 2** Algorithm computing the almost-equivalence $\sim$.

**Require:** minimal WDFA $\mathcal{A}$ and its co-kernel states $\overline{K}$
**Return:** almost-equivalence $\sim$ as a partition and scaling map $f\colon Q \to \underline{\mathbb{K}}$

```
    for all $q \in Q$ do
 2:     $\pi(q) \leftarrow \{q\}$; $f(q) \leftarrow 1$                                   // trivial initial blocks
    $h \leftarrow \emptyset$; $I \leftarrow Q$                                  // hash map of type $h\colon Q^{\Sigma} \to Q$
 4: for all $q \in I$ do
        succ $\leftarrow \mathrm{sig}_q$        // compute standardized signature using current $\delta$ and $\overline{K}$
 6:     if HasValue$(h, \mathrm{succ})$ then
            $q' \leftarrow$ Get$(h, \mathrm{succ})$                  // retrieve state in bucket 'succ' of $h$
 8:         if $|\pi(q')| \geq |\pi(q)|$ then
                Swap$(q, q')$                                   // exchange roles of $q$ and $q'$
10:         $I \leftarrow I \cup \{r \in Q - \{q'\} \mid \exists \sigma\colon \delta(r, \sigma) = q'\}$     // add predecessors of $q'$
            $f(q') \leftarrow \frac{\mathrm{wt}(q', \sigma_0)}{\mathrm{wt}(q, \sigma_0)}$                         // $\sigma_0$ is as in Definition 10
12:         $\mathcal{A} \leftarrow \mathrm{merge}_{\mathcal{A}}(q' \overset{f(q')}{\to} q)$                              // merge $q'$ into $q$
            $\pi(q) \leftarrow \pi(q) \cup \pi(q')$                     // $q$ and $q'$ are almost-equivalent
14:         for all $r \in \pi(q')$ do
                $f(r) \leftarrow f(r) \cdot f(q')$                        // recompute scaling factors
16:     $h \leftarrow$ Put$(h, \mathrm{succ}, q)$                          // store $q$ in $h$ under key 'succ'
    return $(\pi, f)$
```

for every $s \in \Sigma^*$, which shows that $q \approx q'$ $(k)$ because the scaling factor $k$ does not depend on the symbol $\sigma$. □

In fact, the previous proof also shows that at most the empty string yields a difference in $[\![q]\!]_{\mathcal{A}}$ and $[\![q']\!]_{\mathcal{A}}$ (up to the common factor). For the completeness, we also need a restricted converse for minimal WDFA.

**Lemma 12.** *Let $\mathcal{A}$ be minimal, and let $q \approx q'$ be such that $\mathrm{sig}_q \neq \mathrm{sig}_{q'}$. Then there exist $r, r' \in Q$ such that $r \neq r'$ and $\mathrm{sig}_r = \mathrm{sig}_{r'}$.*

**Proof.** Since $q \approx q'$, there exists an integer $\ell$ such that $\delta(q, s) = \delta(q', s)$ for all $s \in \Sigma^*$ with $|s| \geq \ell$ by Lemma 9. Let $s' \in \Sigma^*$ be a maximal string such that $r = \delta(q, s') \neq \delta(q', s') = r'$. Since $s'$ is maximal, we have $\delta(q, s'\sigma) = q_\sigma = \delta(q', s'\sigma)$ for all $\sigma \in \Sigma$. If $q_\sigma$ is a co-preamble state, then $\mathrm{sig}_r(\sigma) = \langle \bot, 1 \rangle = \mathrm{sig}_{r'}(\sigma)$. Now, let $\sigma \in \Sigma$ be such that $q_\sigma$ is a co-kernel state, and let $\sigma_0 \in \Sigma$ be the smallest symbol such that $\delta(r, \sigma_0) \in \overline{K}$. Since $q \approx q'$ and $\approx$ is a congruence relation by Lemma 2, we have $r \approx r'$ $(k)$ for some $k \in \underline{\mathbb{K}}$, which means that $[\![r]\!]_{\mathcal{A}}(s) = k \cdot [\![r']\!]_{\mathcal{A}}(s)$ for almost all $s \in \Sigma^*$. Consequently,

$$\mathrm{wt}(r, \sigma) \cdot [\![q_\sigma]\!]_{\mathcal{A}}(s) = k \cdot \mathrm{wt}(r', \sigma) \cdot [\![q_\sigma]\!]_{\mathcal{A}}(s)$$
$$\mathrm{wt}(r, \sigma_0) \cdot [\![q_{\sigma_0}]\!]_{\mathcal{A}}(s) = k \cdot \mathrm{wt}(r', \sigma_0) \cdot [\![q_{\sigma_0}]\!]_{\mathcal{A}}(s)$$

for almost all $s \in \Sigma^*$. Since both $q_\sigma$ and $q_{\sigma_0}$ are co-kernel states, we conclude that $\mathrm{wt}(r, \sigma) = k \cdot \mathrm{wt}(r', \sigma)$ and $\mathrm{wt}(r, \sigma_0) = k \cdot \mathrm{wt}(r', \sigma_0)$, which yields

$$\frac{\mathrm{wt}(r, \sigma)}{\mathrm{wt}(r, \sigma_0)} = \frac{k \cdot \mathrm{wt}(r', \sigma)}{k \cdot \mathrm{wt}(r', \sigma_0)} = \frac{\mathrm{wt}(r', \sigma)}{\mathrm{wt}(r', \sigma_0)} \enspace .$$

This proves $\mathrm{sig}_r(\sigma) = \mathrm{sig}_{r'}(\sigma)$, and consequently, $\mathrm{sig}_r = \mathrm{sig}_{r'}$ as required. □

Lemmata 11 and 12 suggest Algorithm 2 for computing the almost-equivalence and a map representing the scaling factors. This map contains a scaling factor for each state with respect to a representative state of its block. Algorithm 2 is a straightforward modification of an algorithm by [23] using our standardized signatures. We first compute the standardized signature for each state and store it into a (perfect) hash map [13] to avoid pairwise comparisons. If we find a collision (i.e., a pair of states with the same signature), then we merge them such that the state representing the bigger block survives (see Lines 9 and 12). Each state is considered at most $\log n$ times because the size of the "losing" block containing it at least doubles. After each merge, scaling factors of the "losing" block are computed with respect to the new representative. Again, we only recompute the scaling factor of each state at most $\log n$ times. Hence the small modifications compared to [23] do not increase the asymptotic run-time of Algorithm 2, which is $\mathcal{O}(n \log n)$ where $n$ is the number of states (see Theorem 9 in [23]).

**Proposition 13.** *Algorithm 2 can be implemented to run in time $\mathcal{O}(n \log n)$.*

Finally, we need an adjusted merging process that takes the scaling factors into account. When merging one state into another, their mutual scaling factor can be computed from the scaling map by multiplicaton of one scaling factor with the inverse of the other. Therefore, merging (see Algorithm 3) can be implemented in time $\mathcal{O}(n)$, and hyper-minimization (Algorithm 1) can be implemented in time $\mathcal{O}(n \log n)$ in the weighted setting.

**Proposition 14.** *Our hyper-minimization algorithm can be implemented to run in time $\mathcal{O}(n \log n)$.*

It remains to prove the correctness of our algorithm. To prove the correctness of Algorithm 2, we still need a technical property.

**Lemma 15.** *Let $q, q' \in Q$ be states such that $q \neq q'$ but $\mathrm{sig}_q = \mathrm{sig}_{q'}$. Moreover, let $\mathcal{B} = \mathrm{merge}(q' \xrightarrow{k} q)$ with $k = \frac{f(q')}{f(q)}$, and let $\cong$ be its almost-equivalence (restricted to $P$). Then $\cong \, = \, \approx \cap \, (P \times P)$ where $P = Q - \{q'\}$.*

**Algorithm 3** Merging almost-equivalent states.

**Require:** a minimal WDFA $\mathcal{A}$, its kernel states $K$, its almost-equivalence $\approx$, and a scaling map $f\colon Q \to \underline{\mathbb{K}}$

**Return:** hyper-minimal WDFA $\mathcal{A}$ that is almost-equivalent to the input WDFA

```
   for all $B \in (Q/{\approx})$ do
2:     select $q \in B$ such that $q \in K$ if possible
       for all $q' \in B - K$ do
4:         $\mathcal{A} \leftarrow \text{merge}_{\mathcal{A}}(q' \xrightarrow{\frac{f(q')}{f(q)}} q)$
```

**Proof.** Let $p_1 \approx p_2$ with $p_1, p_2 \in P$. For simplicity's sake, we assume that $q' \neq q_0$, but this missing case can be handled in the same manner. Let $s = \sigma_1 \cdots \sigma_\ell$ with $\sigma_1, \ldots, \sigma_\ell \in \Sigma$. Then we obtain the runs

$$R_{p_1} = \langle \delta(p_1, \sigma_1), \delta(p_1, \sigma_1\sigma_2), \cdots, \delta(p_1, s) \rangle \quad \text{with weight } \text{wt}(p_1, s)$$
$$R_{p_2} = \langle \delta(p_2, \sigma_1), \delta(p_2, \sigma_1\sigma_2), \cdots, \delta(p_2, s) \rangle \quad \text{with weight } \text{wt}(p_2, s).$$

The corresponding runs $R'_{p_1}$ and $R'_{p_2}$ in $\mathcal{B}$ replace every occurrence of $q'$ in both $R_{p_1}$ and $R_{p_2}$ by $q$. Their weights are

$$\text{wt}'(p_1, s) = \begin{cases} \frac{k'_0}{k_0} \cdot \text{wt}(p_1, s) & \text{if } \delta(p_1, s) \neq q' \\ \frac{k'_0}{k_0} \cdot \text{wt}(p_1, s) \cdot k & \text{otherwise} \end{cases}$$
$$\text{wt}'(p_2, s) = \begin{cases} \frac{k'_0}{k_0} \cdot \text{wt}(p_2, s) & \text{if } \delta(p_2, s) \neq q' \\ \frac{k'_0}{k_0} \cdot \text{wt}(p_2, s) \cdot k & \text{otherwise.} \end{cases}$$

Since $\delta(p_1, s') = \delta(p_2, s')$ for suitably long strings $s' \in \Sigma^*$ and $p_1 \approx p_2$, we obtain that $p_1 \cong p_2$. The same reasoning can be used to prove the converse. □

**Theorem 16.** *Algorithm 2 computes $\approx$ and a scaling map.*

**Proof sketch.** If there exist different, but almost-equivalent states, then there exist different states with the same standardized signature by Lemma 12. Lemma 11 shows that such states are almost-equivalent. Finally, Lemma 15 shows that we can continue the computation of the almost-equivalence after a weighted merge of such states. The correctness of the scaling map is shown implicitly in the proof of Lemma 11. □

**Theorem 17.** *We can hyper-minimize* WDFA *in time $\mathcal{O}(n \log n)$.*

## 5. Canonical Regular Languages and Hyper-Optimization

In the remaining sections, we will only work with DFA; i.e., WDFA over the BOOLEAN semifield. An open question in [3] suggests to call a regular language $L \subseteq \Sigma^*$ *canonical* if it is recognized by a hyper-minimal DFA. In other words, the regular language $L$ is canonical if and only if the minimal DFA for $L$ is hyper-minimal. The

| class | complement | $*$ | reversal | $h$ | $h^{-1}$ | $\cup$ | $\cap$ | $-$ | concatenation |
|---|---|---|---|---|---|---|---|---|---|
| regular | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| canonical | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |

Table 1. Closure properties for regular and canonical regular languages ($h$ means homomorphism).

authors of [3] already remark that the canonical regular languages are a proper subset of the regular languages (because the finite languages are not canonical) and closed under complement (because the complement of a hyper-minimal DFA is still hyper-minimal by Theorem 7). However, it remained open which other closure properties canonical regular languages have (see Table 1).

**Theorem 18.** *Canonical regular languages are not closed under any of the following operations: star, reversal, homomorphism, inverse homomorphism, union, intersection, set difference, and concatenation.*

**Proof.** We present all the relevant counterexamples in Figure 4. It can easily be verified that the input DFA are hyper-minimal and the output DFA are not hyper-minimal using Theorem 7. To facilitate this check, we indicated the almost-equivalence for all displayed DFA. □

Another question raised in [3] was whether we can optimize another criterion such as the number of errors or the length of the longest error. This process of optimizing with respect to a secondary criterion is called hyper-optimization. We have already remarked that the DFA (b) and (c) of Figure 1 are hyper-minimal and almost-equivalent. In addition, they are almost-equivalent to the union of (b) and (c) displayed in Figure 4, so both are potential results of a hyper-minimization. In this case, both hyper-minimal DFA commit exactly one error relative to the union DFA, but an example in [36] shows that the gap can be significant. Moreover, the DFA (c) of Figure 1 commits an error of length 3, whereas the error of the DFA (b) is of length 2. Thus, there are qualitative differences between hyper-minimal and almost-equivalent DFA.

The characterization in Theorem 3.9 of [3] establishes the exact relation between almost-equivalent hyper-minimal DFA. Such DFA can only differ in three aspects:

- the finality of preamble states,
- the target of transitions from preamble to kernel states, and
- the initial state.

To formalize these differences, we need some additional notions. A mapping $h\colon Q \to P$ is a transition homomorphism if $h(\delta(q, \sigma)) = \mu(h(q), \sigma)$ for every $q \in Q$ and $\sigma \in \Sigma$. If additionally $q \in F$ if and only if $h(q) \in G$ for every $q \in Q$, then $h$ is a DFA homomorphism. In both cases, $h$ is an isomorphism if it is bijective. It

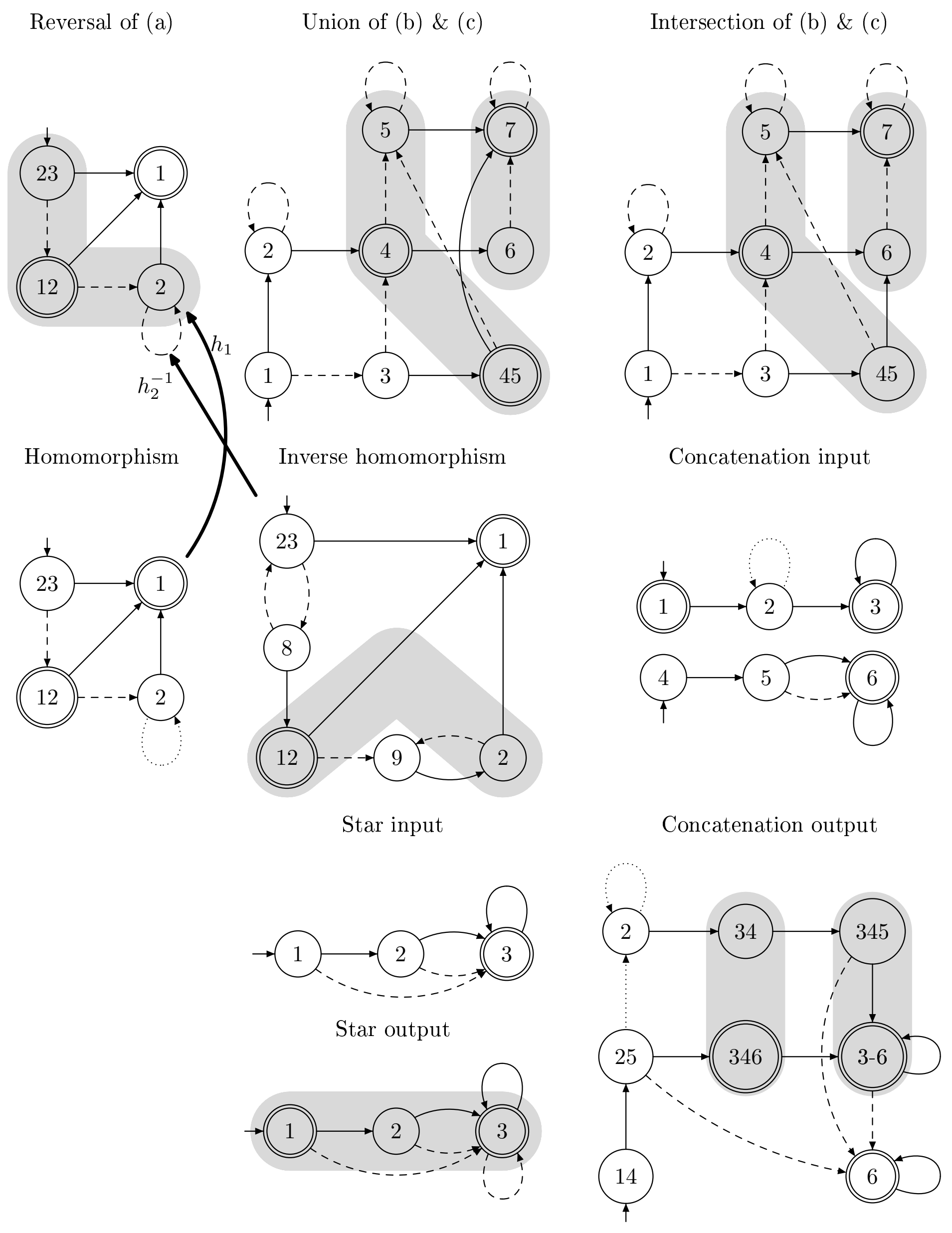

Fig. 4. Hyper-minimal DFA used to prove non-closure under homomorphic image, inverse homomorphic image, and concatenation. The DFA (a), (b), (c) are shown in Figure 1. The DFA below the heading "Homomorphism" yields the "Reversal of (a)" via the homomorphism $h_1$ that keeps all letters but sends the dotted arrow to the dashed arrow. The DFA below "Inverse homomorphism" yields the same DFA via the inverse of the homomorphism $h_2$ that preserves $\longrightarrow$, but sends $\dashrightarrow$ to $\dashrightarrow\longrightarrow$. The DFA below "Concatenation input" concatenated yield the DFA "Concatenation output", which is not hyper-minimal. Similarly, we demonstrate the nonclosure under star.

**Algorithm 4** $\textsc{CompE}(q, q')$: Compute the size of $[\![q]\!]_{\mathcal{A}} \ominus [\![q']\!]_{\mathcal{A}}$.

**Require:** minimal DFA $\mathcal{A}$ and almost-equivalent states $q \sim q'$
**Global:** $Q \times Q$ error matrix $E$ over $\mathbb{N} \cup \{\infty\}$ initially 0 on diagonal and $\infty$ elsewhere

```
   if $E_{q,q'} = \infty$ then
2:     $c \leftarrow (q \in F)$ xor $(q' \in F)$            // 1 error if $q$ and $q'$ differ on finality
       $E_{q,q'} \leftarrow c + \sum_{\sigma \in \Sigma} \textsc{CompE}(\delta(q,\sigma), \delta(q',\sigma))$      // recursive calls
4: return $E_{q,q'}$                                  // return the computed value
```

is well-known that DFA are equivalent if and only if there exists a DFA isomorphism between them.

**Theorem 19 (Theorem 3.9 in [3])** *Let $\mathcal{A}$ and $\mathcal{B}$ be almost-equivalent hyper-minimal* DFA. *Then there exists a bijection $h \colon Q \to P$ such that*

- *$q \sim h(q)$ for every $q \in Q$,*
- *$h$ is a transition isomorphism between preamble states of $\mathcal{A}$ and $\mathcal{B}$, and*
- *$h$ is a* DFA *isomorphism between kernel states of $\mathcal{A}$ and $\mathcal{B}$.*

**Lemma 20 (Lemma 2 in [36])** *Let $\mathcal{B} = \mathrm{merge}_{\mathcal{A}}(q' \to q)$ for some $q, q' \in Q$ with $q \neq q'$. Then*

$$[\![\mathcal{A}]\!] \ominus [\![\mathcal{B}]\!] = \{ss' \mid \textit{shortest } s\textit{: } \delta(q_0, s) = q, s' \in [\![q]\!]_{\mathcal{A}} \ominus [\![q']\!]_{\mathcal{A}}\} \ .$$

It is shown in [34] that (i) all three aspects are responsible for different errors and (ii) the number of errors introduced in a single merge can easily be computed using Lemma 20. Whether a preamble state $q$ is final can be decided based on the finitely many strings leading to $q$. We simply check which option (final or nonfinal) yields fewer errors. To decide the target of a transition from a preamble state $q'$ to a kernel state, we compute the number of errors for each potential target $q$. This is achieved with the help of Lemma 20, for which we compute (i) the number of strings that lead into state $q'$ and (ii) the size of $[\![q]\!]_{\mathcal{A}} \ominus [\![q']\!]_{\mathcal{A}}$ using Algorithm 4.

**Theorem 21 (Corollary 9 in [34])** *A hyper-minimal* DFA *that commits the least number of errors (among all hyper-minimal* DFA*) can be computed in time $\mathcal{O}(n^2)$.*

So we can optimize a secondary criterion, but optimizing a ratio (e.g., balance the number of saved states and the number of errors) is difficult. For example, deciding whether a hyper-minimal DFA with at most $m$ states exists that commits at most $\ell$ errors (relative to $\mathcal{A}$) is NP-hard by Corollary 1 in [17]. Finally, hyper-minimization and hyper-optimization have been evaluated on random DFA [36]. They are effective (good reduction and almost all errors avoidable) on DFA that are also easy to minimize. Surprisingly, DFA that are hard to minimize are also hard to hyper-minimize and hyper-optimize in the sense that only very few states are saved at the expense of a large number of unavoidable errors (see Section 6 in [36]).

## 6. Cover Automata and $k$-Minimization

For all input DFA that recognize a finite language, hyper-minimization simply returns the trivial DFA that recognizes no string, which is undesirable in most applications [37, 4, 33]. A finite language is best represented by a *cover automaton* [8], which is a DFA $\mathcal{A}$ together with a length limit $k \in \mathbb{N}$. It accepts a string $s \in \Sigma^*$ if and only if (i) $|s| \leq k$ and (ii) $s \in [\![\mathcal{A}]\!]$. Thus the DFA $\mathcal{A}$ can make errors on strings longer than $k$. In cover automata minimization we construct a minimal DFA $\mathcal{B}$ such that $[\![\mathcal{B}]\!] \cap \Sigma^{\leq k} = [\![\mathcal{A}]\!] \cap \Sigma^{\leq k}$ for a given cover automaton $(\mathcal{A}, k)$. Similarly, in $k$-minimization [16] we construct a minimal DFA $\mathcal{B}$ such that $[\![\mathcal{B}]\!] \cap \Sigma^{\geq k} = [\![\mathcal{A}]\!] \cap \Sigma^{\geq k}$. A uniform framework is provided by [22], in which the finite difference that we consider for almost-equivalence is generalized to a regular difference.

Cover automata minimization [8, 7, 30, 6, 9, 27] has been studied well, so let us consider $k$-minimization [16]. It is based on the equivalence relation $\sim_k$, which is defined for every $q \in Q$ and $p \in P$ by $q \sim_k p$ if and only if $[\![q]\!]_{\mathcal{A}} \ominus [\![p]\!]_{\mathcal{B}} \subseteq \Sigma^{\leq k}$. The smallest such $k$ (which is $-\infty$ if $q$ and $p$ are equivalent) is called the *gap* $\mathrm{gap}(q, p)$ between $q$ and $p$, and it is the length of a longest string on which $q$ and $p$ disagree. To limit the length of error strings, we also need to consider the lengths $\mathrm{level}_{\mathcal{A}}(q)$ and $\mathrm{level}_{\mathcal{B}}(p)$ of longest strings that lead to $q$ and $p$, respectively. Formally, $\mathrm{level}_{\mathcal{A}}(q) = \sup \{|s| \mid s \in \delta^{-1}(q)\}$.

**Definition 22 (Section 4.1 in [16])** *Let $q \in Q$ and $p \in P$. Then $q$ and $p$ are* $k$-similar *if and only if* $\mathrm{gap}(q, p) + \min(k, \mathrm{level}_{\mathcal{A}}(q), \mathrm{level}_{\mathcal{B}}(p)) \leq k$.

Unfortunately, $k$-similarity is only a compatibility relation (reflexive and symmetric). However, 'gap' behaves like an ultrametric [12], which allows us to construct an ultrametric tree [20, 26, 29] for it in time $\mathcal{O}(n \log n)$ by Theorem 5 of [17]. Using the ultrametric tree we can compute $k$-similarity easily [16, 17]. Overall, $k$-minimization can be performed in time $\mathcal{O}(n \log n)$ [30, 16, 17].

**Theorem 23.** *Cover automata and $k$-minimization run in time $\mathcal{O}(n \log n)$.*

Thus, restricting the length of the errors instead of their number is feasible [3]. However, if we combine the restrictions, then minimization becomes intractable. Formally, given $k, \ell \in \mathbb{N}$ and a DFA $\mathcal{A}$, it is NP-hard to decide whether there exists a $k$-minimal DFA $\mathcal{B}$ committing at most $\ell$ errors by Corollary 2 of [17].

Finally, let us consider an application that combines cover automata and $k$-minimization. A *finite-factored* DFA [2] is a triple $\mathcal{F} = (\mathcal{B}, k, \mathcal{C})$ of two DFA $\mathcal{B}$ and $\mathcal{C}$ over the same alphabet $\Sigma$ and $k \in \mathbb{N}$. It accepts the language

$$[\![\mathcal{F}]\!] = \{s \in \Sigma^* \mid s \in [\![\mathcal{B}]\!] \cap \Sigma^{\leq k} \text{ or } s \in [\![\mathcal{C}]\!] \cap \Sigma^{>k}\} \ .$$

In other words, the authoritative DFA is selected based on the length of string $s$, which is slightly different in Section 3 of [2]. As an example [2] let us consider the language $L = \{0, 1\}^{\leq 5} \cup \{a, b\}^*$ over $\Sigma = \{0, 1, a, b\}$, for which a minimal DFA and two finite-factored DFA are shown in Figure 5.

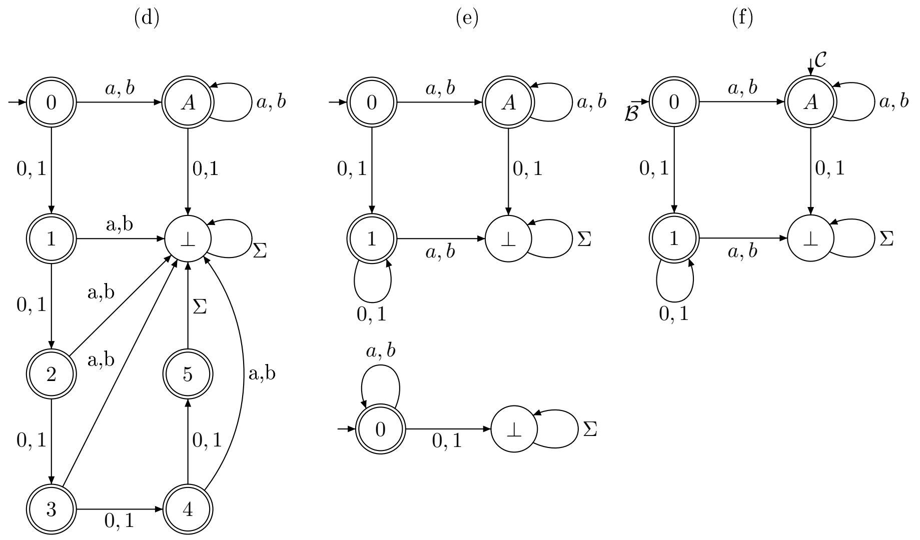

Fig. 5. (d) Minimal DFA $\mathcal{A}$ and (e) and (f) equivalent finite-factored DFA $(\mathcal{B}, 5, \mathcal{C})$ without and with sharing, respectively.

In the example, we selected $k = 5$ and then just minimized the cover automaton $(\mathcal{A}, k)$ and $k$-minimized $\mathcal{A}$ to obtain $\mathcal{B}$ and $\mathcal{C}$, respectively, in time $\mathcal{O}(n \log n)$. The optimal value for $k$ can be computed using Theorem 3 of [17] and Theorem 11 in [27] by computing the sizes of the relevant minimal cover automata and $k$-minimal DFA in time $\mathcal{O}(n \log n)$ for all sensible values of $k$. The size of the finite-factored DFA is the sum of the sizes of two constituting DFA. In the example, we see that the representation using a finite-factored DFA can be smaller than the equivalent minimal DFA. Using all the results previously mentioned we can conclude that even minimization to a finite-factored DFA can be achieved efficiently.

**Theorem 24.** *We can minimize using finite-factored* DFA *in time* $\mathcal{O}(n \log n)$.

## Acknowledgments

The authors would like to express their gratitude towards the reviewers, who helped to improve the presentation of the material.

## References

[1] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut and M. Mohri, "OpenFst: A general and efficient weighted finite-state transducer library," in *Proc. 12th Int. Conf. Implementation and Application of Automata* (Springer, 2007), vol. 4783 of LNCS, pp. 11–23.
[2] A. Badr, "Hyper-minimization in $O(n^2)$," *Int. J. Found. Comput. Sci.* **20** (2009) 735–746.

[3] A. Badr, V. Geffert and I. Shipman, "Hyper-minimizing minimized deterministic finite state automata," *RAIRO Theor. Inf. Appl.* **43** (2009) 69–94.
[4] K. R. Beesley and L. Karttunen, *Finite State Morphology*, CSLI Studies in Computational Linguistics (CSLI Publications, Stanford, CA, 2003).
[5] J. Berstel and C. Reutenauer, *Rational Series and Their Languages*, vol. 12 of *EATCS Monographs on Theoret. Comput. Sci.* (Springer, 1988).
[6] C. Câmpeanu, A. Paun and J. R. Smith, "An incremental algorithm for constructing minimal deterministic finite cover automata," in *Proc. 10th Int. Conf. Implementation and Application of Automata* (Springer, 2005), vol. 3845 of LNCS, pp. 90–103.
[7] C. Câmpeanu, A. Paun and S. Yu, "An efficient algorithm for constructing minimal cover automata for finite languages," *Int. J. Found. Comput. Sci.* **13** (2002) 83–97.
[8] C. Câmpeanu, N. Santean and S. Yu, "Minimal cover-automata for finite languages," *Theor. Comput. Sci.* **267** (2001) 3–16.
[9] J.-M. Champarnaud, F. Guingne and G. Hansel, "Similarity relations and cover automata," *RAIRO Theor. Inf. Appl.* **39** (2005) 115–123.
[10] M. Crochemore and W. Rytter, *Jewels of Stringology* (World Scientific, 2003).
[11] K. Culik II and J. Kari, "Image compression using weighted finite automata," *Computer and Graphics* **17** (1993) 305–313.
[12] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order* (Cambridge University Press, 2002), 2nd edn.
[13] M. Dietzfelbinger, A. R. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert and R. E. Tarjan, "Dynamic perfect hashing: Upper and lower bounds," *SIAM J. Comput.* **23** (1994) 738–761.
[14] J. Eisner, "Simpler and more general minimization for weighted finite-state automata," in *Human Language Technology Conf. of the North American Chapter of the ACL* (2003), pp. 64–71.
[15] T. Fernando, "A finite-state approach to events in natural language semantics," *J. Logic Computat.* **14** (2004) 79–92.
[16] P. Gawrychowski and A. Jeż, "Hyper-minimisation made efficient," in *Proc. 34th Int. Symp. Mathematical Foundations of Computer Science* (Springer, 2009), vol. 5734 of LNCS, pp. 356–368.
[17] P. Gawrychowski, A. Jeż and A. Maletti, "On minimising automata with errors," in *Proc. 36th Int. Symp. Mathematical Foundations of Computer Science* (Springer, 2011), vol. 6907 of LNCS, pp. 327–338.
[18] J. S. Golan, *Semirings and their Applications* (Kluwer Academic, Dordrecht, 1999).
[19] D. Gries, "Describing an algorithm by Hopcroft," *Acta Inform.* **2** (1973) 97–109.
[20] J. A. Hartigan, "Representation of similarity matrices by trees," *J. Amer. Statist. Assoc.* **62** (1967) 1140–1158.
[21] U. Hebisch and H. J. Weinert, *Semirings — Algebraic Theory and Applications in Computer Science* (World Scientific, 1998).
[22] M. Holzer and S. Jakobi, "From equivalence to almost-equivalence, and beyond — minimizing automata with errors," in *Proc. 16th Int. Conf. Developments in Language Theory* (Springer, 2012), vol. 7410 of LNCS.
[23] M. Holzer and A. Maletti, "An $n \log n$ algorithm for hyper-minimizing a (minimized) deterministic automaton," *Theor. Comput. Sci.* **411** (2010) 3404–3413.
[24] J. E. Hopcroft, "An $n \log n$ algorithm for minimizing states in a finite automaton," in *Theory of Machines and Computations* (Academic Press, 1971), pp. 189–196.
[25] J. E. Hopcroft, R. Motwani and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison Wesley, 2007), 3rd edn.
[26] C. J. Jardine, N. Jardine and R. Sibson, "The structure and construction of taxonomic

hierarchies," *Math. Biosci.* **1** (1967) 173–179.

[27] A. Jeż and A. Maletti, "Computing all $l$-cover automata fast," in *Proc. 16th Int. Conf. Implementation and Application of Automata* (Springer, 2011), vol. 6807 of LNCS, pp. 203–214.

[28] C. D. Johnson, *Formal Aspects of Phonological Description*, no. 3 in Monographs on Linguistic Analysis (Mouton, The Hague, 1972).

[29] S. C. Johnson, "Hierarchical clustering schemes," *Psychometrika* **32** (1967) 241–254.

[30] H. Körner, "A time and space efficient algorithm for minimizing cover automata for finite languages," *Int. J. Found. Comput. Sci.* **14** (2003) 1071–1086.

[31] W. Kuich and A. Salomaa, *Semirings, Automata, Languages*, vol. 5 of *EATCS Monographs on Theoretical Computer Science* (Springer, 1986).

[32] S. Lombardy, Y. Régis-Gianas and J. Sakarovitch, "Introducing Vaucanson," *Theor. Comput. Sci.* **328** (2004) 77–96.

[33] C. Mahlow and M. Piotrowski (eds.), *State of the Art in Computational Morphology*, vol. 41 of *Communications in Computer and Information Science* (Springer, 2009).

[34] A. Maletti, "Better hyper-minimization — not as fast, but fewer errors," in *Proc. 15th Int. Conf. Implementation and Application of Automata* (Springer, 2011), vol. 6482 of LNCS, pp. 201–210.

[35] A. Maletti, "Notes on hyper-minimization," in *Proc. 13th Int. Conf. Automata and Formal Languages* (Nyíregyháza College, 2011), pp. 34–49.

[36] A. Maletti and D. Quernheim, "Optimal hyper-minimization," *Int. J. Found. Comput. Sci.* **22** (2011) 1877–1891.

[37] M. Mohri, "Finite-state transducers in language and speech processing," *Comput. Linguist.* **23** (1997) 269–311.

[38] D. Quernheim, "Hyper-minimisation of weighted finite automata," Master's Thesis, Institut für Linguistik, Universität Potsdam, 2010.

[39] J. Sakarovitch, "Rational and recognisable power series," in *Handbook of Weighted Automata*, EATCS Monographs on Theoret. Comput. Sci. (Springer, 2009), chap. 4, pp. 105–174.

[40] A. Salomaa and M. Soittola, *Automata-Theoretic Aspects of Formal Power Series*, Texts and Monographs in Computer Science (Springer, 1978).

[41] R. E. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.* **1** (1972) 146–160.

[42] A. Valmari and P. Lehtinen, "Efficient minimization of DFAs with partial transition functions," in *Proc. 25th Ann. Symp. Theoretical Aspects of Computer Science* (Schloss Dagstuhl — Leibniz-Zentrum für Informatik, Germany, 2008), vol. 1 of *LIPIcs*, pp. 645–656.

[43] P. van Emde Boas, "Machine models and simulation," in *Algorithms and Complexity* (Elsevier and MIT Press, 1990), vol. A of *Handbook of Theoretical Computer Science*, pp. 1–66.

[44] S. Yu, "Regular languages," in *Handbook of Formal Languages* (Springer, 1997), vol. 1, chap. 2, pp. 41–110.