Hyper-minimisation of deterministic weighted finite automata over semifields*

Andreas Maletti** and Daniel Quernheim**

Universität Stuttgart, Institute for Natural Language Processing
Azenbergstraße 12, 70174 Stuttgart, Germany
{Andreas.Maletti & Daniel.Quernheim}@ims.uni-stuttgart.de

Abstract Hyper-minimisation of deterministic finite automata is a recently introduced state reduction technique that allows a finite change in the recognised language. A generalisation of this lossy compression method to the weighted setting over semifields is presented, which allows the recognised formal power series to differ for finitely many input strings. First, the structure of hyper-minimal deterministic weighted finite automata is characterised in a similar way as in classical weighted minimisation and unweighted hyper-minimisation. Second, an efficient minimisation algorithm, which runs in time $\mathcal{O}(n \log n)$, is derived from this characterisation.

1 Introduction

Deterministic finite automata (DFA) [23] are one of the simplest, but most useful devices in computer science. Their simplicity and the availability of efficient manipulation software [16,1] makes them attractive in many application areas such as speech processing [19], image compression [6], morphology [4], natural language semantics [8], and pattern matching [5]. 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 minimisation. The asymptotically most efficient algorithm [14,11] for general DFA minimisation 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.

Recently, hyper-minimisation 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

* This article is based on Daniel Quernheim: *Hyper-minimisation of weighted finite automata.* Master's thesis, Universität Potsdam, 2010.

** Both authors were supported by the German Research Foundation (DFG) grant MA / 4959 / 1–1.

finite change in the recognised language. The asymptotically fastest hyper-minimisation algorithms [9,13] compute the "almost-equivalence" relation and merge states with finite left language according to it in time $\mathcal{O}(n \log n)$.

Here we consider weighted hyper-minimisation. Our weight structures will be commutative semifields, which are commutative semirings [12,10] 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) [21] over semifields, similar results are known. They can be efficiently minimised, but the minimal equivalent WDFA is no longer unique due to ability to "push" weights [19,7]. The asymptotically fastest minimisation algorithms [19,7] nevertheless still run in time $\mathcal{O}(n \log n)$. Essentially, they normalise 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 minimisation can then perform the computation of the equivalence and the merges.

In weighted hyper-minimisation we focus on the notion that allows the recognised series to differ for finitely many input strings, which we call 'strict almost-equivalence'. More sophisticated notions that are based on differences between the weights of strings are conceivable [20], but we will mostly discuss strict almost-equivalence. This notion has the benefit that it is simple, but realistic enough. We will join unweighted hyper-minimisation and weighted minimisation to weighted hyper-minimisation. Our algorithm (see Algs. 1 and 2) contains features of both of its predecessors and is asymptotically as efficient as them because it also runs in time $\mathcal{O}(n \log n)$. In contrast to [20], we introduce standardised signatures to avoid the explicit pushing of weights. This adjustment allows us to mold our weighted hyper-minimisation algorithm into the structure of the unweighted algorithm [13].

2 Preliminaries

An alphabet Σ is simply a finite set, and Σ^* is the set of all strings over it including ε, which is the empty string. The length of a string $x = x_1 \cdots x_\ell$ with $x_1, \ldots, x_\ell \in \Sigma$ is $|x| = \ell$. Concatenation of strings is simply denoted by juxtaposition. A language L over Σ is a subset $L \subseteq \Sigma^*$.

A commutative semifield is a tuple $\langle \mathbb{K}, +, \cdot, 0, 1 \rangle$ such that (i) $\langle \mathbb{K}, +, 0 \rangle$ and $\langle \mathbb{K}, \cdot, 1 \rangle$ are commutative monoids, (ii) the multiplication $\cdot$ distributes over the addition $+$ [i.e., $(k_1 + k_2) \cdot k_3 = (k_1 \cdot k_3) + (k_2 \cdot k_3)$ for all $k_1, k_2, k_3 \in \mathbb{K}$], (iii) 0 is an annihilator [i.e., $k \cdot 0 = 0$ for all $k \in \mathbb{K}$], and (iv) for every $k \in \mathbb{K} - \{0\}$ there exists $k^{-1} \in \mathbb{K}$ such that $k \cdot k^{-1} = 1$. In other words, a

commutative semifield is a commutative semiring [12,10] with multiplicative inverses. Useful semifields include (i) the real numbers $\langle \mathbb{R}, +, \cdot, 0, 1 \rangle$, (ii) the tropical semifield $\langle \mathbb{R} \cup \{\infty\}, \min, +, \infty, 0 \rangle$, (iii) 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 (iv) the BOOLEAN semifield $\mathbb{B} = \langle \{0,1\}, \max, \min, 0, 1 \rangle$.

Let $\langle \mathbb{K}, +, \cdot, 0, 1 \rangle$ be a commutative semifield. A (power) series is a mapping $\alpha \colon \Sigma^* \to \mathbb{K}$. Its support $\mathrm{supp}(\alpha) \subseteq \Sigma^*$ is $\{x \mid \alpha(x) \neq 0\}$. Given $k \in \mathbb{K}$, we let $(k \cdot \alpha) \colon \Sigma^* \to \mathbb{K}$ be the series such that $(k \cdot \alpha)(x) = k \cdot \alpha(x)$ for every $x \in \Sigma^*$. A weighted deterministic finite automaton (WDFA) is a tuple $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, c, F \rangle$, where (i) Q is a finite set of states, (ii) Σ is an alphabet, (iii) $q_0 \in Q$ is an initial state, (iv) $\delta \colon Q \times \Sigma \to Q$ is a transition mapping, (v) $c \colon Q \times \Sigma \to \mathbb{K} - \{0\}$ is a weight (or cost) assignment, and $F \subseteq Q$ is a set of final states. We can extend 'δ' and 'c' to mappings $\delta \colon Q \times \Sigma^* \to Q$ and $c \colon Q \times \Sigma^* \to \mathbb{K} - \{0\}$ by $\delta(q, \varepsilon) = q$ and $\delta(q, ax) = \delta(\delta(q,a), x)$ and $c(q, \varepsilon) = 1$ and $c(q, ax) = c(q,a) \cdot c(\delta(q,a), x)$ for every $q \in Q$, $a \in \Sigma$, and $x \in \Sigma^*$. For every $q \in Q$, its right series $[\![q]\!]_{\mathcal{A}} \colon \Sigma^* \to \mathbb{K}$ is given for every $x \in \Sigma^*$ by $[\![q]\!]_{\mathcal{A}}(x) = c(q,x)$ if $\delta(q,x) \in F$ and $[\![q]\!]_{\mathcal{A}}(x) = 0$ otherwise. If $\mathcal{A}$ is obvious from the context, then we just write $[\![q]\!]$ instead of $[\![q]\!]_{\mathcal{A}}$. The WDFA $\mathcal{A}$ recognises the series $[\![\mathcal{A}]\!] = [\![q_0]\!]$.

> *In the following, let $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, c, F \rangle$ be a* WDFA *over the commutative semifield $\langle \mathbb{K}, +, \cdot, 0, 1 \rangle$. We assume a fixed alphabet Σ.*

Let $\mathcal{B} = \langle Q', \Sigma, q_0', \delta', c', F' \rangle$ be a WDFA. The product $\mathcal{A} \cdot \mathcal{B}$ [22,15] is the WDFA $\langle Q \times Q', \Sigma, \langle q_0, q_0' \rangle, \delta'', c'', F \times F' \rangle$ with $\delta''(\langle q, q' \rangle, a) = \langle \delta(q,a), \delta(q',a) \rangle$ and $c''(\langle q, q' \rangle, a) = c(q,a) \cdot c'(q',a)$ for every $q \in Q$, $q' \in Q'$, and $a \in \Sigma$. Let $\alpha, \beta \colon \Sigma^* \to \mathbb{K}$. If there exists $k \in \mathbb{K} - \{0\}$ such that $\alpha = k \cdot \beta$, then α and β are equivalent, which is written $\alpha \equiv \beta$ or $\alpha \equiv \beta$ (k) if the factor is relevant. For $q \in Q$ and $q' \in Q'$ we write $q \equiv q'$ and $q \equiv q'$ (k) if $[\![q]\!]_{\mathcal{A}} \equiv [\![q']\!]_{\mathcal{B}}$ and $[\![q]\!]_{\mathcal{A}} \equiv [\![q']\!]_{\mathcal{B}}$ (k), respectively. The WDFA $\mathcal{A}$ and $\mathcal{B}$ are equivalent if $[\![q_0]\!]_{\mathcal{A}} = [\![q_0']\!]_{\mathcal{B}}$. An equivalence relation $\cong \subseteq Q \times Q$ is a congruence if $\delta(q,a) \cong \delta(p,a)$ for all $a \in \Sigma$ and $q \cong p$. Note that $\equiv \subseteq Q \times Q$ is a congruence. If $\mathbb{K} = \mathbb{B}$, then the notion of WDFA coincides with the classical (unweighted) notion of the 'deterministic finite automaton' (DFA) [23].

3 State of research

A WDFA is *minimal* if no WDFA with (strictly) fewer states recognises the same series. We can obtain a minimal equivalent WDFA from a given WDFA by merging *equivalent states* until no pair of different, but equivalent states

remains. In the unweighted case, the resulting minimal DFA is unique and can be constructed in time $\mathcal{O}(n \log n)$ [14,11], where n is the number of states of the input DFA. In the weighted case, the resulting WDFA is not unique, but it can also be obtained in time $\mathcal{O}(n \log n)$ [19]. In order to achieve a further reduction, errors (i.e., changes in the recognised series) have to be allowed. A recent technique is *hyper-minimisation* [3], which simply allows any finite number of errors.

Definition 1 (see [3, Def. 2.2]). *The languages $L, L' \subseteq \Sigma^*$ are* almost-equivalent*, which is denoted by $L \sim L'$, if their symmetric difference is finite. In addition, the states $q, p \in Q$ are* almost-equivalent*, which is again denoted by $q \sim p$, if* $\operatorname{supp}([\![q]\!]) \sim \operatorname{supp}([\![p]\!])$.

A state $q \in Q$ is a preamble state if $\{x \in \Sigma^* \mid \delta(q_0, x) = q\}$ is finite. Otherwise, it is a kernel state. The sets of all preamble states and all kernel states are P and $K = Q - P$, respectively. The notion of almost-equivalence on states can trivially be extended to pairs of states belonging to different WDFA $\mathcal{A}$ and $\mathcal{B}$ that share the same input alphabet Σ. We say that the WDFA $\mathcal{A}$ and $\mathcal{B}$ are almost-equivalent, which is also denoted by $\mathcal{A} \sim \mathcal{B}$, if their initial states are almost-equivalent (i.e., $q_0 \sim q_0'$). Finally, the WDFA $\mathcal{A}$ is *hyper-minimal* if there is no almost-equivalent WDFA with (strictly) fewer states. Recall that $\mathcal{A}$ is minimal if it has no pair of different, but equivalent states. A similar characterisation was obtained in [3] for hyper-minimality.

Theorem 2 (see [3, Thm. 3.4]). *A minimal* DFA *is hyper-minimal if and only if it has no pair of different, but almost-equivalent states such that at least one of them is a preamble state.*

A hyper-minimal DFA can be obtained by merging states [3]. A merge of state p into state q reroutes all transitions entering p into q. If p was the initial state, then q is the new initial state. However, finality is not adjusted. In Fig. 1 we show the merge of the state B into A, which yields that the states B, D, and F can be deleted since they are inaccessible. Note that a hyper-minimal DFA can have almost-equivalent kernel states (for example: G and I in Fig. 1). Such states cannot be merged unless they are equivalent.

Recall that the equivalent minimal DFA for a given DFA $\mathcal{A}$ is unique (up to isomorphism). Although there is no unique almost-equivalent hyper-minimal DFA for $\mathcal{A}$, all hyper-minimal DFA that are almost-equivalent to $\mathcal{A}$ share structural similarities, which were already studied in [3].

Theorem 3 (see [3, Thm. 3.9]). *Let $\mathcal{A}$ and $\mathcal{B}$ be hyper-minimal* DFA *with states Q and Q', respectively. If they are almost-equivalent, then there*

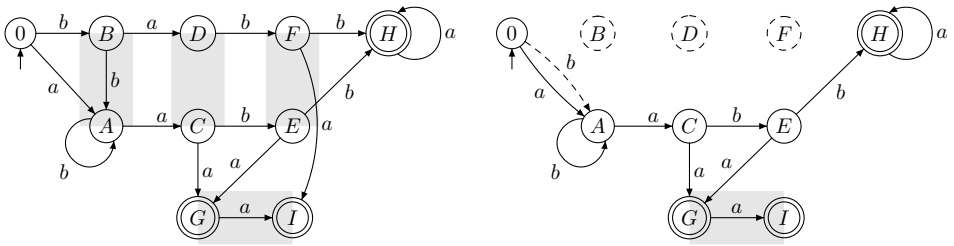

Figure 1. Minimal DFA (left) and hyper-minimal DFA (right). Almost-equivalent states have been indicated by shading. The preamble states are $\{0, B, D, F\}$, and the hyper-minimal DFA was obtained by merging the state B into A. Note that the almost-equivalent kernel states G and I cannot be merged.

is a mapping $h\colon Q \to Q'$ *such that (i)* $q \sim h(q)$ *and (ii)* $q \equiv h(q)$ *and* $h(q)$ *is a kernel state provided that* q *is a kernel state for every* $q \in Q$.

The difficult part of the hyper-minimisation algorithm [3] is the computation of the almost-equivalent states. Several approaches for this problem have been proposed. The original algorithm [3] runs in time $\mathcal{O}(n^3)$ where n is the number of states of the input DFA. It was improved to $\mathcal{O}(n^2)$ in [2]. A further improvement to $\mathcal{O}(n \log n)$ was achieved independently in [9,13], of which [9] also discusses how the length of the longest error string can be constrained. A conceptually simple quadratic-time hyper-minimisation algorithm [17] also keeps track of the number of errors and returns the optimal DFA among all hyper-minimal and almost-equivalent DFA. The recent contribution [18] contains a detailed account of this algorithm and an empirical evaluation of hyper-minimisation.

4 Hyper-minimisation

In this section, we will investigate hyper-minimisation for WDFA.

Proposition 4. *The language* $\operatorname{supp}(\llbracket \mathcal{A} \rrbracket)$ *can be recognised by a* DFA *with* $|Q|$ *states.*

Proof. We simply remove the weight component to obtain a DFA. Formally, the DFA $\mathcal{B} = \langle Q, \Sigma, q_0, \delta, F \rangle$ trivially recognises $\operatorname{supp}(\llbracket \mathcal{A} \rrbracket)$. □

From Prop. 4 it follows immediately that WDFA are almost-equivalent if and only if the corresponding DFA are almost-equivalent. This yields that we can reduce hyper-minimisation for WDFA to the unweighted case. We present this situation graphically in Fig. 2.

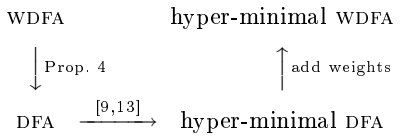

Figure 2. Constructing a hyper-minimal WDFA.

Almost-equivalence is a rather weak property for WDFA because it only requires that there are at most finitely many differences in the support. Essentially, this disregards the weights completely, which allowed us to reduce hyper-minimisation to the unweighted case. Next, we consider a stricter notion, which we will investigate in the rest of the paper.

Definition 5. *The series* $\alpha, \beta \colon \Sigma^* \to \mathbb{K}$ *are* strictly almost-equivalent *if there exists* $k \in \mathbb{K} - \{0\}$ *such that* $\alpha(x) = k \cdot \beta(x)$ *for almost all* $x \in \Sigma^*$. *We write* $\alpha \approx \beta$ *or* $\alpha \approx \beta$ (k) *if* α *and* β *are strictly almost-equivalent (with factor* k*). The states* $q, p \in Q$ *are strictly almost-equivalent, which is again denoted by* $q \approx p$ *or* $q \approx p$ (k), *if* $[\![q]\!] \approx [\![p]\!]$ (k) *for some* $k \in \mathbb{K} - \{0\}$.

Lemma 6. *Strict almost-equivalence is an equivalence relation on series and a congruence on states.*

Proof. We first prove the statement for series. Trivially, $\approx$ is reflexive and symmetric. Let $\alpha, \beta, \gamma \colon \Sigma^* \to \mathbb{K}$ be a series such that $\alpha \approx \beta$ (k) and $\beta \approx \gamma$ (k'). Then there exist finite sets A and B such that $\alpha(x) = k \cdot \beta(x)$ and $\beta(y) = k' \cdot \gamma(y)$ for all $x \in \Sigma^* - A$ and $y \in \Sigma^* - B$. Consequently, $\alpha(z) = k \cdot k' \cdot \gamma(z)$ for all $z \in \Sigma^* - (A \cup B)$, which proves $\alpha \approx \gamma$ $(k \cdot k')$. The same argumentation can be used for $\approx$ on states.

It remains to prove that $\approx$ on states is a congruence. Let $q \approx p$ (k) and $a \in \Sigma$. Consequently, there exists a finite set $A \subseteq \Sigma^*$ such that $[\![q]\!](x) = k \cdot [\![p]\!](x)$ for all $x \in \Sigma^* - A$. Moreover,

$$[\![\delta(q,a)]\!](y) = \frac{[\![q]\!](ay)}{c(q,a)} \qquad \text{and} \qquad [\![\delta(p,a)]\!](y) = \frac{[\![p]\!](ay)}{c(p,a)}$$

for all $y \in \Sigma^*$. Since $q \approx p$ (k), we obtain that

$$[\![\delta(q,a)]\!](y) = \frac{c(p,a)}{c(q,a)} \cdot k \cdot [\![\delta(p,a)]\!](y) \tag{1}$$

for all $y \in \Sigma^*$ such that $ay \notin A$. This proves that $\delta(q,a) \approx \delta(p,a)$, which in turn proves that $\approx$ is a congruence. □

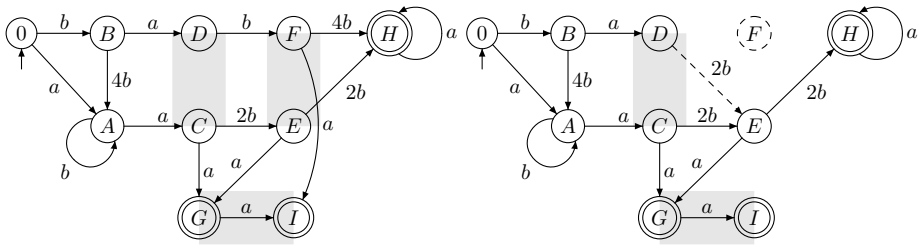

Figure 3. WDFA over the real numbers [left] and the WDFA obtained by merge$(F, 2, E)$ [right]. Where no weight is indicated, the multiplicative identity 1 is implicit. Omitted transitions lead to the sink state $\perp$. The dashed line indicates the rerouted transition.

For the WDFA displayed in Fig. 3 the strict almost-equivalence is determined by the partition $\{\{0\}, \{A\}, \{B\}, \{C, D\}, \{E, F\}, \{G, I, \perp\}, \{H\}\}$. Whenever two series are strictly almost-equivalent, then they are almost-equivalent. In general, the converse is not true. As usual, two WDFA $\mathcal{A}$ and $\mathcal{B}$ are strictly almost-equivalent if their initial states are. It remains to show how to find a smallest WDFA that is strictly almost-equivalent to $\mathcal{A}$. Such a WDFA is called strictly hyper-minimal in the following. It turns out that the existing hyper-minimisation algorithms can also be extended to this setting. Suppose that $\mathcal{A}$ is a minimal WDFA, which can be ensured using the algorithms of [7]. We can use weighted merges of preamble states into strictly almost-equivalent states because such merges will only change the weights associated to a finite number of strings. Intuitively, a weighted merge consists of local pushing [7], followed by edge redirection.

Definition 7. *Let $q, p \in Q$ and $k \in \mathbb{K}$. The* k-weighted merge *of p into q is the* WDFA $\text{merge}(p, k, q) = \langle Q, \Sigma, q_0', \delta', c', F\rangle$ *with for all $r \in Q$ and $a \in \Sigma$*

$$q_0' = \begin{cases} q & \text{if } q_0 = p \\ q_0 & \text{otherwise} \end{cases}$$

$$\delta'(r,a) = \begin{cases} q & \text{if } \delta(r,a) = p \\ \delta(r,a) & \text{otherwise} \end{cases} \qquad c'(r,a) = \begin{cases} k \cdot c(r,a) & \text{if } \delta(r,a) = p \\ c(r,a) & \text{otherwise.} \end{cases}$$

Going back to the WDFA of Fig. 3 (left), we can merge F into E using the scaling factor 2. The WDFA $\text{merge}(F, 2, E)$ is displayed in Fig. 3 (right).

Lemma 8. *Let $p \approx q$ (k), and let $\mathcal{B} = \text{merge}(p, k, q)$. Then $\mathcal{B}$ is strictly almost-equivalent to $\mathcal{A}$ if $p \in P$.*

Proof. Let $\mathcal{B} = \langle Q, \Sigma, q_0', \delta', c', F\rangle$ be the merged WDFA. Clearly, we have $[\![\mathcal{B}]\!](x) = [\![\mathcal{A}]\!](x)$ for all $x \in \Sigma^*$ such that $\delta(q_0, y) \neq p$ for all prefixes y of x (i.e., $x = yz$ for some $z \in \Sigma^*$). This is simply due to the fact that $\mathcal{B}$ faithfully replicates the behaviour of $\mathcal{A}$ in this case. Now let $y \in \Sigma^*$ be such that $\delta(q_0, y) = p$. By assumption, $[\![p]\!](z) = k \cdot [\![q]\!](z)$ for almost all $z \in \Sigma^*$, which yields that $\delta(p, z) \in F$ if and only if $\delta(q, z) \in F$ for almost all $z \in \Sigma^*$. In the case $\delta(p, z) \notin F$ and $\delta(q, z) \notin F$, we obtain that $[\![\mathcal{B}]\!](yz) = 0 = [\![\mathcal{A}]\!](yz)$. In the remaining case we have $\delta(p, z) \in F$ and $\delta(q, z) \in F$. Since $[\![\mathcal{A}]\!](yz) = c(q_0, y) \cdot [\![p]\!](z)$ and $c'(q, z) = c(q, z)$ because the state p is not reachable from q, we obtain

$$\begin{aligned}[\![\mathcal{B}]\!](yz) = c'(q_0', y) \cdot c'(q, z) \quad &= c(q_0, y) \cdot k \cdot c(q, z) \\ = c(q_0, y) \cdot k \cdot [\![q]\!](z) &= c(q_0, y) \cdot [\![p]\!](z) = [\![\mathcal{A}]\!](yz) \ .\end{aligned}$$

Thus, the semantics $[\![\mathcal{B}]\!]$ and $[\![\mathcal{A}]\!]$ coincide for almost all strings with prefix y. Since p is a preamble state, there are only finitely many such strings y, which yields that $\mathcal{B}$ and $\mathcal{A}$ are strictly almost-equivalent. □

Corollary 9 (of Lm. 8). *If $\mathcal{A}$ has a pair of different, but strictly almost-equivalent states such that at least one of them is a preamble state, then $\mathcal{A}$ is not strictly hyper-minimal.*

Proof. Let $p \approx q$ (k) with $p \neq q$ and $p \in P$. Then the WDFA merge(p, k, q) is strictly almost-equivalent to $\mathcal{A}$ by Lm. 8, and it is smaller because the state p can be deleted. □

Consequently, the WDFA of Fig. 3 (right) is not strictly hyper-minimal because the states C and D are strictly almost-equivalent and D is a preamble state. In addition, Lm. 8 shows that the WDFA of Fig. 3 are strictly almost-equivalent. This was the simple part of the merging strategy. To show that the merging process indeed yields a strictly hyper-minimal WDFA, we still have to show the converse of Crl. 9.

Lemma 10. *Let $\mathcal{A}$ be minimal. The* WDFA *$\mathcal{A}$ is strictly hyper-minimal if it has no pair of different, but strictly almost-equivalent states such that at least one of them is a preamble state.*

Proof. Let $\mathcal{B} = \langle Q', \Sigma, q_0', \delta', c', F'\rangle$ be a WDFA with $|Q'| < |Q|$ that is strictly almost-equivalent to $\mathcal{A}$. Clearly, $q_0 \approx q_0'$ and due to the congruence property (which also holds between different WDFA) we obtain that $\delta(q_0, x) \approx \delta'(q_0', x)$ for all $x \in \Sigma^*$. Since $|Q'| < |Q|$, there are $x, y \in \Sigma^*$ with $q_1 = \delta(q_0, x) \neq \delta(q_0, y) = q_2$ but $\delta'(q_0', x) = q' = \delta'(q_0', y)$. Consequently,

$q_1 = \delta(q_0, x) \approx \delta'(q'_0, x) = q' = \delta'(q'_0, y) \approx \delta(q_0, y) = q_2$, which yields $q_1 \approx q_2$. Clearly, q_1 and q_2 are kernel states because otherwise the WDFA $\mathcal{A}$ would have a pair of strictly almost-equivalent states such that at least one of them is a preamble state, which contradicts the assumption.

Since q_1 and q_2 are kernel states, the strings x and y can be chosen such that $|x| \geq |Q|^2 \leq |y|$. We will now use an argument that is similar to the one used in the proof of [3, Thm. 3.1]. Let $x = x_1 \cdots x_\ell$ with $x_1, \ldots, x_\ell \in \Sigma$. If we run $\mathcal{A}$ and $\mathcal{B}$ on the prefixes of x, then we obtain pairs of states $\langle p_i, p'_i \rangle = \langle \delta(q_0, x_1 \cdots x_i), \delta'(q'_0, x_1 \cdots x_i) \rangle$ for every $1 \leq i \leq \ell$. By assumption $\ell \geq |Q|^2 > |Q| \cdot |Q'|$. Consequently, there are indices $1 \leq i < j \leq \ell$ such that $(p_i, p'_i) = (p_j, p'_j)$. It follows that $\langle q_1, q' \rangle$ is a kernel state in the product WDFA $\mathcal{A} \cdot \mathcal{B}$. In the same way, we can prove that also $\langle q_2, q' \rangle$ is a kernel state of $\mathcal{A} \cdot \mathcal{B}$. Let $L_{\langle q_1, q' \rangle}$ and $L_{\langle q_2, q' \rangle}$ be the infinite sets of strings that take the WDFA $\mathcal{A} \cdot \mathcal{B}$ into $\langle q_1, q' \rangle$ and $\langle q_2, q' \rangle$, respectively. Since $\mathcal{B}$ and $\mathcal{A}$ are strictly almost-equivalent, there exist $k_1, k_2 \in \mathbb{K} - \{0\}$ such that

$$c(q_0, w_1) \cdot [\![q_1]\!](z) = [\![\mathcal{A}]\!](w_1 z) = k_1 \cdot [\![\mathcal{B}]\!](w_1 z) = k_1 \cdot c'(q'_0, w_1) \cdot [\![q']\!](z)$$
$$c(q_0, w_2) \cdot [\![q_2]\!](z) = [\![\mathcal{A}]\!](w_2 z) = k_2 \cdot [\![\mathcal{B}]\!](w_2 z) = k_2 \cdot c'(q'_0, w_2) \cdot [\![q']\!](z)$$

for almost all $w_1 z, w_2 z \in \Sigma^*$ with $w_1 \in L_{\langle q_1, q' \rangle}$ and $w_2 \in L_{\langle q_2, q' \rangle}$. We can select $w_1 \in L_{\langle q_1, q' \rangle}$ and $w_2 \in L_{\langle q_2, q' \rangle}$ such that the previous two equations hold for all $z \in \Sigma^*$ because $L_{\langle q_1, q' \rangle}$ and $L_{\langle q_2, q' \rangle}$ are infinite. Consequently,

$$\frac{c(q_0, w_1) \cdot [\![q_1]\!](z)}{k_1 \cdot c'(q'_0, w_1)} = \frac{c(q_0, w_2) \cdot [\![q_2]\!](z)}{k_2 \cdot c'(q'_0, w_2)} \qquad \text{and} \qquad [\![q_1]\!](z) = k \cdot [\![q_2]\!](z)$$

for all $z \in \Sigma^*$, where $k = \frac{k_1 \cdot c'(q'_0, w_1) \cdot c(q_0, w_2)}{k_2 \cdot c'(q'_0, w_2) \cdot c(q_0, w_1)}$. This yields $q_1 \equiv q_2$ (k). However, we have $q_1 \neq q_2$, which due to the minimality of $\mathcal{A}$ yields $q_1 \not\equiv q_2$. This is a contradiction, which shows that such a WDFA $\mathcal{B}$ cannot exist. □

The WDFA of Fig. 4 (right) is strictly hyper-minimal, which can be verified with the help of Lm. 10. Now we combine Cor. 9 and Lm. 10 to obtain a characterisation that is similar to Thm. 2.

Theorem 11. *Let $\mathcal{A}$ be minimal. Then $\mathcal{A}$ is strictly hyper-minimal if and only if it has no pair of different, but strictly almost-equivalent states such that at least one of them is a preamble state.*

5 Minimisation algorithms

The general structure of the strict hyper-minimisation algorithm is presented in Alg. 1. Minimisation is well-described in [7], and computing the

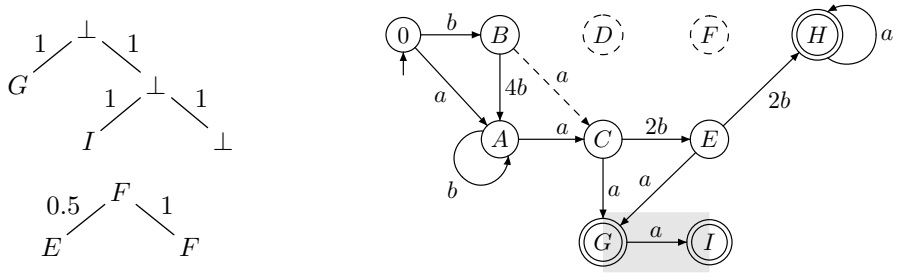

Figure 4. Trees representing scaling factors for the non-trivial blocks of $\approx$ (left) and strictly hyper-minimal WDFA (right). The dashed line indicates the rerouted transition.

Algorithm 1 Strict hyper-minimisation in semifields.

Require: a WDFA $\mathcal{A}$ over a semifield
Return: a strictly hyper-minimal WDFA strictly almost-equivalent to $\mathcal{A}$

$\mathcal{A} \leftarrow \textsc{EisnerMinimise}(\mathcal{A})$
$\langle K, \overline{K} \rangle \leftarrow \textsc{ComputeKernelCoKernel}(\mathcal{A})$
$\langle \approx, f \rangle \leftarrow \textsc{ComputeStrictAlmostEquivalence}(\mathcal{A}, \overline{K})$
return $\textsc{MergeWeighted}(\mathcal{A}, \approx, K, f)$

kernel is explained in [13]. We also need to compute the *co-kernel* (see Def. 12), which is essentially the same problem (the co-kernel is the kernel of the reversed automaton). After we explored the structure of strictly hyper-minimal WDFA in the previous section, we still need to compute the strict almost-equivalence and the factors needed for the merges. Let us consider the minimal WDFA $\mathcal{A}$ in Fig. 3 (left). How do we efficiently determine whether a pair of almost-equivalent states like C and D is also strictly almost-equivalent? Obviously, $[\![C]\!]$ and $[\![D]\!]$ are equal for all strings with prefix 'bb'. However, they differ for a finite number of strings beginning with 'a' or 'ba', which yields $C \not\equiv D$. To identify such finite subsets, we introduce an additional notion.

Definition 12. *A state $q \in Q$ is a* co-preamble state *if* $\operatorname{supp}([\![q]\!])$ *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.*

On our example WDFA of Fig. 3 (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 strict 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 strictly almost-equivalent. In addition, a co-preamble state can-

not be strictly almost-equivalent to a co-kernel state. The interesting part of the strict almost-equivalence is thus completely determined by the series of the co-kernel states. This special role of the co-preamble states has already been pointed out in [9] in the context of DFA. In the following, we assume an arbitrary, but fixed total order on the alphabet symbols of Σ. The efficiency of the computation of the almost-equivalence in the existing hyper-minimisation algorithms relies on hashing the signature of a state. The hash can efficiently return a state with the same signature, which avoids the pairwise comparisons. To make this idea applicable in our setting, we need to standardise the signature. To this end, we ignore transitions into co-preamble states and standardise the transition weights.

Definition 13. *The* standardised signature $\mathrm{sig}_q \colon \Sigma \to Q \times \mathbb{K}$ *of* $q \in Q$ *is such that for every* $a \in \Sigma$*:*

- *If* $\delta(q,a) \in \overline{P}$*, then* $\mathrm{sig}_q(a) = \langle \bot, 1 \rangle$.
- *Otherwise, let* $a_0 \in \Sigma$ *be the smallest symbol such that* $\delta(q,a_0) \in \overline{K}$*. Then* $\mathrm{sig}_q(a) = \langle \delta(q,a), \frac{c(q,a)}{c(q,a_0)} \rangle$.

For the example WDFA of Fig. 3 (left) we obtain $\mathrm{sig}_E(a) = \langle \bot, 1 \rangle$ and $\mathrm{sig}_E(b) = \langle H, 0 \rangle$, which coincides with sig_F. Next, we show that states with equal standardised signature are indeed strictly almost-equivalent.

Lemma 14. *Let* $q, p \in Q$ *be such that* $\mathrm{sig}_q = \mathrm{sig}_p$*. Then* $q \approx p$.

Proof. If q or p is a co-preamble state, then both q and p are co-preamble states and thus $q \approx p$. Now, let $q, p \in \overline{K}$, and let $k = \frac{c(q,a_0)}{c(p,a_0)}$, where a_0 is the smallest symbol such that $\delta(q,a_0) \in \overline{K}$. For every $a \in \Sigma$ and $x \in \Sigma^*$,

$$[\![q]\!](ax) = c(q,a) \cdot [\![\delta(q,a)]\!](x) \quad \text{and} \quad [\![p]\!](ax) = c(p,a) \cdot [\![\delta(p,a)]\!](x) \ .$$

Further, let $\mathrm{sig}_q(a) = \langle q_a, k_a \rangle = \mathrm{sig}_p(a)$. If $q_a = \bot$, then $[\![q]\!](ax) = k \cdot [\![p]\!](ax)$ for almost all $x \in \Sigma^*$. Otherwise, we obviously have $\delta(q,a) = q_a = \delta(p,a)$, and we obtain

$$[\![q]\!](ax) = k_a \cdot c(q,a_0) \cdot [\![q_a]\!](x) = \frac{c(p,a)}{c(p,a_0)} \cdot c(q,a_0) \cdot [\![q_a]\!](x) = k \cdot [\![p]\!](ax)$$

for every $x \in \Sigma^*$, which shows that $q \approx p$ (k) because 'k' does not depend on the symbol 'a'. □

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

Lemma 15. *Let $\mathcal{A}$ be minimal, and let $q \approx p$ be such that $\mathrm{sig}_q \neq \mathrm{sig}_p$. Then there exist $q', p' \in Q$ such that $q' \neq p'$ and $\mathrm{sig}_{q'} = \mathrm{sig}_{p'}$.*

Proof. Since $q \approx p$, there exists an integer ℓ such that $[\![\delta(q,x)]\!] \equiv [\![\delta(p,x)]\!]$ for all $x \in \Sigma^*$ with $|x| \geq \ell$. The minimality of $\mathcal{A}$ yields that $\delta(q,x) = \delta(p,x)$ for all such x. Let $y \in \Sigma^*$ be maximal with $q' = \delta(q,y) \neq \delta(p,y) = p'$. Since y is maximal, we have $\delta(q,ya) = q_a = \delta(p,ya)$ for all $a \in \Sigma$. If q_a is a co-preamble state, then $\mathrm{sig}_{q'}(a) = \langle \bot, 1 \rangle = \mathrm{sig}_{p'}$. Now, let $b \in \Sigma$ be such that q_b is a co-kernel state, and let $a_0 \in \Sigma$ be the smallest symbol such that $\delta(q', a_0) \in \overline{K}$. Since $q \approx p$ and $\approx$ is a congruence by Lm. 6, we have $q' \approx p'$ (k) for some $k \in \mathbb{K} - \{0\}$, which means that $[\![q']\!](x) = k \cdot [\![p']\!](x)$ for almost all $x \in \Sigma^*$. Consequently,

$$c(q', b) \cdot [\![q_b]\!](z) = k \cdot c(p', b) \cdot [\![q_b]\!](z)$$
$$c(q', a_0) \cdot [\![q_{a_0}]\!](z) = k \cdot c(p', a_0) \cdot [\![q_{a_0}]\!](z)$$

for almost all $z \in \Sigma^*$. Since both q_b and q_{a_0} are co-kernel states, we conclude that $c(q', b) = k \cdot c(p', b)$ and $c(q', a_0) = k \cdot c(p', a_0)$, which yields

$$\frac{c(q', b)}{c(q', a_0)} = \frac{k \cdot c(p', b)}{k \cdot c(p', a_0)} = \frac{c(p', b)}{c(p', a_0)} \quad .$$

This proves $\mathrm{sig}_{q'}(b) = \mathrm{sig}_{p'}(b)$, which yields $\mathrm{sig}_{q'} = \mathrm{sig}_{p'}$ as required. □

Lemmata 14 and 15 suggest the algorithm in Alg. 2 for computing the strict almost-equivalence and a tree representing some of the scaling factors. It is a straightforward modification of an algorithm by [13] using our standardised signatures. We start with singleton sets of states (since every state is strictly almost-equivalent to itself) and merge states (representing blocks) with equal standardised signature until no such states exist anymore. Then the obtained partition is a representation of the strict almost-equivalence.

Essentially, the only changes to the original algorithm are the use of standardised signatures and the computation of the scaling factors. This is achieved by constructing a tree for each block, representing the merges. Sample trees are given in Fig. 4 (left). The function JoinTrees(t_1, t_2, w) builds a binary tree t with labeled edges from two subtrees t_1 and t_2. Assume that p is the root node of t_1 and q is the root node of t_2. Then $t = q((t_1 : w), (t_2 : 1))$. This does not increase the asymptotic runtime of Alg. 2, which is $\mathcal{O}(n \log n)$ where n is the number of states.

We can then reconstruct scaling factors for an arbitrary pair of states p and q in the same block represented by state r by evaluating the paths starting at the root node and leading to the leaves labeled p and q, respectively.

Algorithm 2 Computing the strict almost-equivalence.

Require: a minimal WDFA $\mathcal{A}$ and co-kernel $\overline{K}$
Return: strict almost-equivalence $\approx$ and scale trees t

for $q \in Q$ **do**
  $\pi(q) \leftarrow \{q\}$ // singleton classes initially
  $t(q) \leftarrow q$
$h \leftarrow \emptyset$ // hash map for signatures
$I \leftarrow Q$; $J \leftarrow Q$ // states that need to be considered and current states
while $I \neq \emptyset$ **do**
  $q \leftarrow \textsc{RemoveHead}(I)$; $s \leftarrow \mathrm{sig}(q)$
  if $\textsc{HasValue}(h, s)$ **then**
    $p \leftarrow \textsc{Get}(h, s)$ // equal signature found
    if $|\pi(p)| \geq |\pi(q)|$ **then**
      $\textsc{Swap}(p, q)$ // for efficiency reasons
    $J \leftarrow J - \{p\}$ // p is merged into q's class
    $I \leftarrow (I - \{p\}) \cup \{r \in J \mid (\exists a : \delta(r, a) = p\}$
    $f(p, q) \leftarrow \frac{c(p,a_0)}{c(q,a_0)}$ // a_0 is as in Def. 13
    $\mathcal{A} \leftarrow \mathrm{merge}(p, f(p, q), q)$ // perform the merge
    $\pi(q) \leftarrow \pi(q) \cup \pi(p)$
    $t(q) \leftarrow \textsc{JoinTrees}(t(p), t(q), f(p, q))$
  $h \leftarrow \textsc{Put}(h, s, q)$ // q is the new representative of $[q]$
return $\langle \pi, t \rangle$

Algorithm 3 Merging almost-equivalent states.

Require: a minimal WDFA $\mathcal{A}$, its kernel states K, scale trees t, and strict almost-equivalence $\approx$

for all $B \in (Q/\!\approx)$ **do**
  select $q \in B$ such that $q \in K$ if possible
  for all $p \in B - K$ **do**
    $f(p, q) \leftarrow \textsc{ComputeScalingFactor}(t, p, q)$
    $\mathcal{A} \leftarrow \mathrm{merge}(p, f(p, q), q)$ // complexity: $\mathcal{O}(\log n)$

A path is evaluated by multiplying the edge label weights. Since these trees have the property that any path from a leaf labeled p to a node labeled r evaluates to $f(p, r)$, we obtain $f(p, q) = \frac{f(q,r)}{f(p,r)}$. These lookups take at most $\mathcal{O}(\log n)$ time for a given pair of states since the height of the trees is at most $\log n$. Therefore, merging (Alg. 3), and as a consequence, also strict hyper-minimisation (Alg. 1) can be implemented in time $\mathcal{O}(n \log n)$.

For the correctness of Alg. 2, we still need to prove a technical property.

Lemma 16. *Let $p \neq q$ be states such that $\mathrm{sig}(p) = \mathrm{sig}(q)$. Moreover, let $\mathcal{B} = \mathrm{merge}(p, f(p, q), q)$, and let $\cong$ be its strict almost-equivalence (restricted to Q'). Then $\cong \; = \; \approx \cap (Q' \times Q')$ where $Q' = Q - \{p\}$.*

Proof. Let $p' \approx q'$ be such that $p' \neq p \neq q'$, and let $\mathcal{B} = \langle Q, \Sigma, q'_0, \delta', c', F \rangle$. For simplicity's sake, we assume that $p \neq q_0$, but this missing case can be handled in the same manner. Let $x = x_1 \cdots x_\ell$ with $x_1, \ldots, x_\ell \in \Sigma$. Then we obtain the runs

$$\begin{aligned} R_{p'} &= \langle \delta(p', x_1), \delta(p', x_1x_2), \cdots, \delta(p', x) \rangle \quad \text{with weight } c(p', x) \\ R_{q'} &= \langle \delta(q', x_1), \delta(q', x_1x_2), \cdots, \delta(q', x) \rangle \quad \text{with weight } c(q', x). \end{aligned}$$

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

$$c'(p', x) = \begin{cases} c(p', x) & \text{if } \delta(p', x) \neq p \\ c(p', x) \cdot f(p, q) & \text{otherwise} \end{cases}$$

$$c'(q', x) = \begin{cases} c(q', x) & \text{if } \delta(q', x) \neq p \\ c(q', x) \cdot f(p, q) & \text{otherwise.} \end{cases}$$

Since $\delta(p', y) = \delta(q', y)$ for suitably long strings y and $p' \approx q'$, we obtain that $p' \cong q'$. The same reasoning can be used to prove the converse. □

Theorem 17. *Algorithm 2 computes $\approx$ and an incomplete scaling map.*

Proof (Sketch). If there exist different, but strictly almost-equivalent states, then there exist different states with the same standardised signature by Lm. 15. Lemma 14 shows that such states are strictly almost-equivalent. Finally, Lm. 16 shows that we can continue the computation of the strict almost-equivalence after a weighted merge of such states. The correctness of the scaling map is shown implicitly in the proof of Lm. 14. □

Acknowledgements

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

References

1. Allauzen, C., Riley, M., Schalkwyk, J., Skut, W., Mohri, M.: OpenFst: A general and efficient weighted finite-state transducer library. In: Proc. 12th Int. Conf. Implementation and Application of Automata (CIAA). LNCS, vol. 4783, pp. 11–23. Springer (2007)
2. Badr, A.: Hyper-minimization in $O(n^2)$. Int. J. Found. Comput. Sci. 20(4), 735–746 (2009)

3. Badr, A., Geffert, V., Shipman, I.: Hyper-minimizing minimized deterministic finite state automata. RAIRO Theor. Inf. Appl. 43(1), 69–94 (2009)
4. Beesley, K.R., Karttunen, L.: Finite State Morphology. CSLI Studies in Computational Linguistics, CSLI Publications, Stanford, CA (2003)
5. Crochemore, M., Rytter, W.: Jewels of Stringology. World Scientific (2003)
6. Culik II, K., Kari, J.: Image compression using weighted finite automata. Computer and Graphics 17(3), 305–313 (1993)
7. Eisner, J.: Simpler and more general minimization for weighted finite-state automata. In: Proc. Joint Meeting Human Language Technology and the North American Chapter of the ACL (HLT-NAACL). pp. 64–71. ACL (2003)
8. Fernando, T.: A finite-state approach to events in natural language semantics. J. Logic Computat. 14(1), 79–92 (2004)
9. Gawrychowski, P., Jeż, A.: Hyper-minimisation made efficient. In: Proc. 34th Int. Symp. Mathematical Foundations of Computer Science (MFCS). LNCS, vol. 5734, pp. 356–368. Springer (2009)
10. Golan, J.S.: Semirings and their Applications. Kluwer Academic, Dordrecht (1999)
11. Gries, D.: Describing an algorithm by Hopcroft. Acta Inform. 2(2), 97–109 (1973)
12. Hebisch, U., Weinert, H.J.: Semirings — Algebraic Theory and Applications in Computer Science. World Scientific (1998)
13. Holzer, M., Maletti, A.: An $n \log n$ algorithm for hyper-minimizing a (minimized) deterministic automaton. Theor. Comput. Sci. 411(38–39), 3404–3413 (2010)
14. Hopcroft, J.E.: An $n \log n$ Algorithm for Minimizing the States in a Finite Automaton. In: Kohavi, Z. (ed.) The Theory of Machines and Computations, pp. 189–196. Academic Press (1971)
15. Kuich, W., Salomaa, A.: Semirings, Automata, Languages, EATCS Monographs on Theoretical Computer Science, vol. 5. Springer (1986)
16. Lombardy, S., Régis-Gianas, Y., Sakarovitch, J.: Introducing Vaucanson. Theor. Comput. Sci. 328(1–2), 77–96 (2004)
17. Maletti, A.: Better hyper-minimization — not as fast, but fewer errors. In: Proc. 15th Int. Conf. Implementation and Application of Automata (CIAA). LNCS, vol. 6482, pp. 201–210. Springer (2011)
18. Maletti, A., Quernheim, D.: Optimal hyper-minimization (2011), Manuscript available at `http://arxiv.org/abs/1104.3007`
19. Mohri, M.: Finite-state transducers in language and speech processing. Comput. Linguist. 23(2), 269–311 (1997)
20. Quernheim, D.: Hyper-minimisation of weighted finite automata. Master's thesis, Institut für Linguistik, Universität Potsdam (2010)
21. Sakarovitch, J.: Rational and recognisable power series. In: Droste, M., Kuich, W., Vogler, H. (eds.) Handbook of Weighted Automata, chap. 4, pp. 105–174. EATCS Monographs on Theoretical Computer Science, Springer (2009)
22. Salomaa, A., Soittola, M.: Automata-Theoretic Aspects of Formal Power Series. Texts and Monographs in Computer Science, Springer (1978)
23. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 1, chap. 2, pp. 41–110. Springer (1997)