

International Journal of Foundations of Computer Science
 © World Scientific Publishing Company

OPTIMAL HYPER-MINIMIZATION*

ANDREAS MALETTI[†] and DANIEL QUERNHEIM

*Institute for Natural Language Processing, Universität Stuttgart
 Azenbergstraße 12, 70174 Stuttgart, Germany
 {andreas.maletti, daniel.quernheim}@ims.uni-stuttgart.de*

Received (31 January 2011)

Accepted (Day Month Year)

Communicated by (xxxxxxxxxx)

Minimal deterministic finite automata (DFAs) can be reduced further at the expense of a finite number of errors. Recently, such minimization algorithms have been improved to run in time $O(n \log n)$, where n is the number of states of the input DFA, by [GAWRYCHOWSKI and JEŹ: Hyper-minimisation made efficient. Proc. MFCS, LNCS 5734, 2009] and [HOLZER and MALETTI: An $n \log n$ algorithm for hyper-minimizing a (minimized) deterministic automaton. *Theor. Comput. Sci.* 411, 2010]. Both algorithms return a DFA that is as small as possible, while only committing a finite number of errors. These algorithms are further improved to return a DFA that commits the least number of errors at the expense of an increased (quadratic) run-time. This solves an open problem of [BADR, GEFFERT, and SHIPMAN: Hyper-minimizing minimized deterministic finite state automata. *RAIRO Theor. Inf. Appl.* 43, 2009]. In addition, an experimental study on random automata is performed and the effects of the existing algorithms and the new algorithm are reported.

Keywords: deterministic finite automaton; minimization; error analysis.

2010 Mathematics Subject Classification: 68Q45, 68Q25, 68W40

1. Introduction

Deterministic finite automata (DFAs) [14] are used in a vast number of applications that require huge automata like speech processing [11] or linguistic analysis [10]. To keep the operations efficient, minimal DFA are typically used in applications. A minimal DFA is such that all equivalent DFAs are larger, where the size is measured by the number of states. The asymptotically fastest minimization algorithm runs in time $O(n \log n)$ and is due to HOPCROFT [9], where n is the size of the input DFA.

*This is an extended and revised version of [A. Maletti: *Better hyper-minimization — not as fast, but fewer errors.* In Proc. CIAA, volume 6482 of LNCS, pages 201-210. Springer-Verlag, 2011].

[†]The work was carried out while the author was at the *Departament de Filologies Romàniques, Universitat Rovira i Virgili* (Tarragona, Spain) and was supported by the *Ministerio de Educación y Ciencia* (MEC) grants JDCI-2007-760 and MTM-2007-63422.

Recently, stronger minimization procedures, called hyper-minimization, have been investigated [2, 1, 5, 7, 12]. They can efficiently compress minimal DFAs even further at the expense of a finite number of errors. The fastest hyper-minimization algorithms [5, 7] run in time $O(n \log n)$. More specifically, given an input DFA M , a hyper-minimization algorithm returns a *hyper-minimal* DFA for M , which

- recognizes the same language as M up to a finite number of errors, and
- is minimal among all DFAs with the former property (hyper-minimal).

In this contribution, we extend a known hyper-minimization algorithm to return a *hyper-optimal* DFA for M , which is a hyper-minimal DFA for M that commits the least number of errors among all hyper-minimal DFAs for M . Moreover, the algorithm returns the number of committed errors, which allows a user to disregard the returned DFA if the number is unacceptably large. Our algorithm is based essentially on a syntactic characterization of hyper-minimal DFAs for M (see Theorems 3.8 and 3.9 of [2]). Roughly speaking, two hyper-minimal DFAs for M differ in exactly three aspects [2]: (i) the finality of the states P that are reachable by only finitely many strings, (ii) the transitions from states of P to states not in P , and (iii) the initial state. The characterization has two main uses: It allows us to compute the exact number of errors for each hyper-minimal DFA for M , and it allows us to easily consider all hyper-minimal DFAs for M in order to find a hyper-optimal DFA for M . We thus solve a remaining open problem of [2]. Unfortunately, the time complexity of the obtained algorithm is $O(n^2)$, and it remains an open problem whether the algorithm can be improved to run in time $O(n \log n)$.

Finally, we demonstrate hyper-minimization and the new algorithm on test DFAs, which we generated from random non-deterministic finite automata [14, 13]. The difficult cases for minimization that were identified in [13] also prove to be difficult for hyper-minimization in the sense that only a small reduction is possible at the expense of a significant amount of errors. The new algorithm alleviates this problem by avoiding a large number of mistakes. Outside the hard instances of [13], already hyper-minimization reduces the size nicely at the expense of only a few errors.

2. Preliminaries

The set of integers is \mathbb{Z} , and the subset of nonnegative integers is \mathbb{N} . If the symmetric difference $S \triangle T = (S \setminus T) \cup (T \setminus S)$ of two sets S and T is finite, then S and T are almost-equal. Each finite set Σ is an alphabet, and the set of all strings over Σ is Σ^* . The empty string is ε , and the concatenation of two strings $u, v \in \Sigma^*$ is denoted by the juxtaposition uv . The length of the string $w = \sigma_1 \cdots \sigma_k$ with $\sigma_1, \dots, \sigma_k \in \Sigma$ is $|w| = k$. A string $u \in \Sigma^*$ is a prefix of w if there exists a string $v \in \Sigma^*$ such that $w = uv$. Any subset $L \subseteq \Sigma^*$ is a language over Σ .

A deterministic finite automaton (for short: DFA) is a tuple $M = (Q, \Sigma, q_0, \delta, F)$, in which Q is a finite set of states, Σ is an alphabet of input symbols, $q_0 \in Q$ is an initial state, $\delta: Q \times \Sigma \rightarrow Q$ is a transition mapping, and $F \subseteq Q$ is a set of

final states. The transition mapping δ extends to a mapping $\underline{\delta}: Q \times \Sigma^* \rightarrow Q$ by $\underline{\delta}(q, \varepsilon) = q$ and $\underline{\delta}(q, \sigma w) = \underline{\delta}(\delta(q, \sigma), w)$ for every $q \in Q$, $\sigma \in \Sigma$, and $w \in \Sigma^*$. For every $q \in Q$, let

$$L(M, q) = \{w \in \Sigma^* \mid \underline{\delta}(q_0, w) = q\} \quad \text{and} \quad L(q, M) = \{w \in \Sigma^* \mid \underline{\delta}(q, w) \in F\} .$$

Intuitively, $L(M, q)$ contains all strings that take M (from the initial state q_0) into the state q , and $L(q, M)$ contains all strings that take M from q into a final state. Moreover, $\text{Ker}(M) = \{q \in Q \mid L(M, q) \text{ infinite}\}$ is the set of kernel states of M , and $\text{Pre}(M) = Q \setminus \text{Ker}(M)$ is the set of preamble states. The sets $\text{Ker}(M)$ and $\text{Pre}(M)$ can be computed in time $O(m)$, where $m = |Q \times \Sigma|$. The DFA M recognizes the language $L(M) = L(q_0, M) = \bigcup_{q \in F} L(M, q)$.

An equivalence relation $\equiv \subseteq S \times S$ is a reflexive, symmetric, and transitive binary relation. The equivalence class of an element $s \in S$ is $[s]_{\equiv} = \{s' \in S \mid s \equiv s'\}$ and $[S]_{\equiv} = \{[s]_{\equiv} \mid s \in S\}$. A weak partition of S is a set Π such that (i) $A \subseteq S$ for every $A \in \Pi$, (ii) $A_1 \cap A_2 = \emptyset$ for all different $A_1, A_2 \in \Pi$, and (iii) $S = \bigcup_{A \in \Pi} A$. An equivalence relation $\equiv \subseteq Q \times Q$ on the states of the DFA $M = (Q, \Sigma, q_0, \delta, F)$ is a congruence relation on M if $\delta(q_1, \sigma) \equiv \delta(q_2, \sigma)$ for all $q_1 \equiv q_2$ and $\sigma \in \Sigma$.

Let $M = (Q, \Sigma, q_0, \delta, F)$ and $N = (P, \Sigma, p_0, \mu, G)$ be two DFAs. A mapping $h: Q \rightarrow P$ 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. Finally, we say that the DFAs M and N are (transition and DFA) isomorphic if there exists a (transition and DFA, respectively) isomorphism $h: Q \rightarrow P$.

The DFAs M and N are equivalent if $L(M) = L(N)$. Clearly, (DFA) isomorphic DFAs are equivalent. Two states $q \in Q$ and $p \in P$ are equivalent, denoted by $q \equiv p$, if $L(q, M) = L(p, N)$.^a The equivalence $\equiv \subseteq Q \times Q$ is a congruence relation on M . The DFA M is minimal if it does not have equivalent states (i.e., $q_1 \equiv q_2$ implies $q_1 = q_2$ for all $q_1, q_2 \in Q$). The name ‘minimal’ is justified by the fact that there does not exist a DFA with strictly fewer states that recognizes the same language as a minimal DFA. A minimal DFA that is equivalent to M can be computed efficiently using HOPCROFT’s algorithm [8], which runs in time $O(m \log n)$ where $m = |Q \times \Sigma|$ and $n = |Q|$. Moreover, minimal DFAs are equivalent if and only if they are isomorphic.

Similarly, the DFAs M and N are almost-equivalent if $L(M)$ and $L(N)$ are almost-equal. The states $q \in Q$ and $p \in P$ are almost-equivalent, which is denoted by $q \sim p$, if $L(q, M)$ and $L(p, M)$ are almost-equal. The almost-equivalence $\sim \subseteq Q \times Q$ is also a congruence. The minimal DFA M is hyper-minimal if it does not have a pair $(q_1, q_2) \in Q \times Q$ of different, but almost-equivalent states such that $\{q_1, q_2\} \cap \text{Pre}(M) \neq \emptyset$. Again, the name ‘hyper-minimal’ is justified by the fact that there does not exist a DFA with strictly fewer states that recognizes an almost-

^aWhile it might not be clear from the notation $q \equiv p$ to which DFA a state belongs, it will typically be clear from the context. In particular, we might have $M = N$; i.e., we might relate two states from the same DFA.

Algorithm 1 Structure of a hyper-minimization algorithm.

Require: a DFA $M = (Q, \Sigma, q_0, \delta, F)$ with $m = |Q \times \Sigma|$ and $n = |Q|$
 $M \leftarrow \text{MINIMIZE}(M)$ // HOPCROFT's algorithm; $O(m \log n)$
2: $\sim \leftarrow \text{COMPAEQUIV}(M)$ // compute almost-equivalence; $O(m \log n)$
 $M \leftarrow \text{MERGESTATES}(M, \text{Ker}(M), \sim)$ // merge almost-equivalent states; $O(m)$
4: **return** M

equivalent language (see Theorem 3.4 of [2]). A hyper-minimal DFA that is almost-equivalent to M is called “hyper-minimal for M ” and can be computed efficiently using the algorithms of [5, 7], which also run in time $O(m \log n)$. A structural characterization of hyper-minimal DFAs is presented in Theorems 3.8 and 3.9 of [2], which we reproduce here.

Theorem 1 (see [2]) *Let $M = (Q, \Sigma, q_0, \delta, F)$ and $N = (P, \Sigma, p_0, \mu, G)$ be almost-equivalent DFAs. Then $\underline{\delta}(q_0, w) \sim \underline{\mu}(p_0, w)$ for every $w \in \Sigma^*$. In addition, if M and N are hyper-minimal, then there exists a mapping $h: Q \rightarrow P$ such that*

- $q \sim h(q)$ for every $q \in Q$,
- h yields a transition isomorphism between $\text{Pre}(M)$ and $\text{Pre}(N)$, and
- h yields a DFA isomorphism between $\text{Ker}(M)$ and $\text{Ker}(N)$.

3. Hyper-minimization

Hyper-minimization as introduced in [2] is a form of lossy compression with the goal of reducing the size of a minimal DFA at the expense of a finite number of errors. More formally, hyper-minimization aims to find a hyper-minimal DFA for an input DFA. Several hyper-minimization algorithms exist [2, 1, 5, 7], and the overall structure of the hyper-minimization algorithm of [7] is displayed in Algorithm 1. For the following discussion let $M = (Q, \Sigma, q_0, \delta, F)$ be a DFA, and let $m = |Q \times \Sigma|$ and $n = |Q|$ be the number of its transitions and the number of its states, respectively.

The most interesting component of Algorithm 1 is the merging process. In general, the merge of a state $p \in Q$ into another state $q \in Q$ redirects all incoming transitions of p to q . If $p = q_0$ then q is the new initial state. The finality of q is not changed even if p is final. Clearly, the state p can be deleted after the merge if $p \neq q$. Formally, $\text{merge}_M(p \rightarrow q) = (P, \Sigma, p_0, \mu, F)$, where $P = (Q \setminus \{p\}) \cup \{q\}$ and for every $q' \in Q$ and $\sigma \in \Sigma$

$$p_0 = \begin{cases} q & \text{if } q_0 = p \\ q_0 & \text{otherwise} \end{cases} \quad \text{and} \quad \mu(q', \sigma) = \begin{cases} q & \text{if } \delta(q', \sigma) = p \\ \delta(q', \sigma) & \text{otherwise.} \end{cases}$$

Lemma 2. *Let $p, q \in Q$ and $N = \text{merge}_M(p \rightarrow q)$. Then*

$$L(M) \triangle L(N) = \{uw \mid u \in L(M, p), w \in L(p, M) \triangle L(q, M)\} .$$

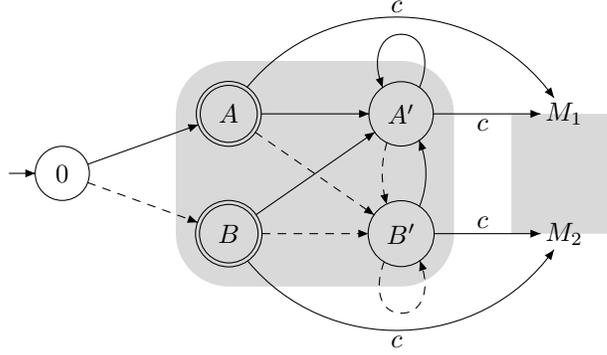


Fig. 1. An example DFA, where unbroken lines are a -transitions and dashed lines are b -transitions.

Consequently, M and $\text{merge}_M(p \rightarrow q)$ are almost-equivalent if $q \sim p$ and $p \in \text{Pre}(M)$. The hyper-minimization algorithms of [2, 1, 5, 7] only perform such merges. More precisely, the procedure `MERGE STATES` merges almost-equivalent states in the mentioned fashion until the obtained DFA is hyper-minimal. The number of errors introduced in this way differs among several hyper-minimal DFA for M and depends on the merges performed. In this contribution, we develop an algorithm that computes a hyper-minimal DFA for M that commits the minimal number of errors among all hyper-minimal DFAs for M . A DFA N is *hyper-optimal* for M if it is hyper-minimal and the cardinality of the symmetric difference between $L(M)$ and $L(N)$ is minimal among all hyper-minimal DFAs. Note that a hyper-optimal DFA for M is hyper-minimal for M . Moreover, our algorithm returns the exact number of errors, and we could also return a compact representation of the actual error strings. Overall, we thus solve a problem that remained open in [2].

An extreme example is presented in Fig. 1. If we run the hyper-minimization algorithms of [2, 1, 5, 7], then we obtain one of the two first DFAs of Fig. 2. Both of them commit $2 + |L(M_1) \Delta L(M_2)|$ errors. If we let $L(M_1) = \Sigma^k$ for some $k \in \mathbb{N}$ and $L(M_2) = \emptyset$, then they commit $2 + |\Sigma|^k$ errors. On the other hand, the optimal DFA is the third DFA of Fig. 2, and it commits only 2 errors (irrespective of M_1 and M_2). This shows that the gap in the number of errors can be very significant.

4. Computing the number of errors

Next, we show how to efficiently compute the number of errors that are caused by a single merge (see Lemma 2). For this we first compute the size of the difference between almost-equivalent states $p \sim q$. From now on, let $M = (Q, \Sigma, q_0, \delta, F)$ be a minimal DFA. In our examples, we will always refer to our running example DFA M_{ex} , which is presented in Fig. 3. Its kernel states are $\text{Ker}(M_{\text{ex}}) = \{E, F, I, J, K, L, M\}$ and the following partition represents its almost-equivalence:

$$\{0\} \quad \{A\} \quad \{B\} \quad \{C, D\} \quad \{E\} \quad \{F\} \quad \{G, H, I, J\} \quad \{K, L, M\}.$$

6 A. Maletti and D. Quernheim

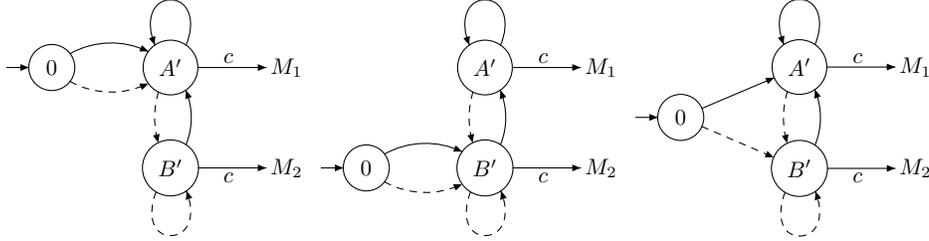


Fig. 2. Three hyper-minimal DFAs for the DFA of Fig. 1, where unbroken lines are a -transitions and dashed lines are b -transitions.

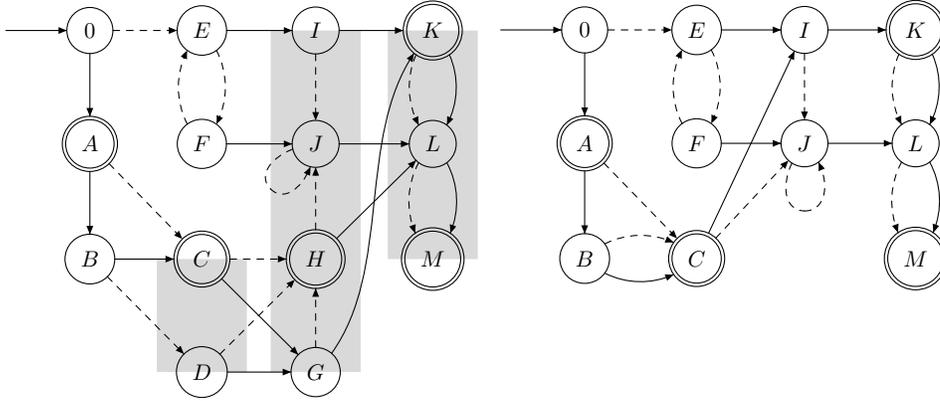


Fig. 3. Example DFA M_{ex} (left) and optimal hyper-minimal DFA N_{ex} (right) for M_{ex} , where unbroken lines are a -transitions and dashed lines are b -transitions.

In comparison to the DFA M_{ex} of Fig. 3, the DFA N_{ex} of Fig. 3 commits the following seven errors: $\{aaaab, aaab, aab, aabab, aabb, abab, abb\}$. Note that existing algorithms will only find hyper-minimal DFAs that commit 16 errors, and the worst hyper-minimal DFA for M_{ex} commits 29 errors.

Definition 3. For every $q \sim p$, let

$$E_{q,p} = \begin{cases} 0 & \text{if } q = p \\ \sum_{\sigma \in \Sigma} E_{\delta(q,\sigma), \delta(p,\sigma)} + \begin{cases} 0 & \text{if } q \in F \iff p \in F \\ 1 & \text{otherwise} \end{cases} & \text{otherwise.} \end{cases}$$

Lemma 4. $E_{q,p} = |L(q, M) \Delta L(p, M)|$ for every $q \sim p$.

Proof. Let $q \sim p$. Then $|L(q, M) \Delta L(p, M)|$ is finite by definition, and we let $k_{q,p} = \max \{|w| \mid w \in L(q, M) \Delta L(p, M)\}$, where $\max \emptyset = -\infty$. Now, we prove the statement by induction on $\mathbb{N} \cup \{-\infty\}$. First, suppose that $k_{q,p} = -\infty$. Then

Algorithm 2 COMPE: Compute the error matrix E .

Require: minimal DFA $M = (Q, \Sigma, q_0, \delta, F)$ and states $q \sim p$

Global: error matrix $E \in \mathbb{Z}^{Q \times Q}$ initially 0 on the diagonal and -1 elsewhere

```

if  $E_{q,p} = -1$  then
2:    $c \leftarrow ((q \in F) \text{ xor } (p \in F))$  // set errors to 1 if  $q$  and  $p$  differ on finality
       $E_{q,p} \leftarrow c + \sum_{\sigma \in \Sigma} \text{COMPE}(M, \delta(q, \sigma), \delta(p, \sigma))$  // recursive calls
4: return  $E_{q,p}$  // return the computed value

```

$L(q, M) = L(p, M)$, which yields that $q \equiv p$. Since M is minimal, we conclude that $q = p$ and $E_{q,p} = 0$, which proves the induction base. Second, suppose that $k_{q,p} \geq 0$, and let $W = \{\sigma w \mid \sigma \in \Sigma, w \in L(\delta(q, \sigma), M) \Delta L(\delta(p, \sigma), M)\}$. Obviously, $W \subseteq L(q, M) \Delta L(p, M) \subseteq W \cup \{\varepsilon\}$ and $k_{\delta(q, \sigma), \delta(p, \sigma)} < k_{q,p}$ for every $\sigma \in \Sigma$. The empty string ε is in $L(q, M) \Delta L(p, M)$ if and only if q and p differ on finality. Moreover, $E_{\delta(q, \sigma), \delta(p, \sigma)} = |L(\delta(q, \sigma), M) \Delta L(\delta(p, \sigma), M)|$ for every $\sigma \in \Sigma$ by induction hypothesis. Since $k_{q,p} \geq 0$, we have $q \neq p$ and

$$E_{q,p} = \sum_{\sigma \in \Sigma} E_{\delta(q, \sigma), \delta(p, \sigma)} + \begin{cases} 0 & \text{if } q \in F \iff p \in F \\ 1 & \text{otherwise,} \end{cases}$$

which proves the induction step and the statement. \square

Let us illustrate Algorithm 2 on the example DFA M_{ex} of Fig. 3. We list some error matrix entries together with the corresponding error strings. Note that the error strings are not computed by the algorithm, but are presented for illustrative purposes only.

$$\begin{array}{lll} E_{G,H} = 5 & \{\varepsilon, a, aa, ab, b\} & E_{H,I} = 4 & \{\varepsilon, a, aa, ab\} & E_{K,L} = 3 & \{\varepsilon, a, b\} \\ E_{G,I} = 1 & \{b\} & E_{H,J} = 1 & \{\varepsilon\} & E_{K,M} = 2 & \{a, b\} \\ E_{G,J} = 4 & \{a, aa, ab, b\} & E_{I,J} = 3 & \{a, aa, ab\} & E_{L,M} = 1 & \{\varepsilon\} \end{array}$$

Theorem 5. Algorithm 2 can be used to compute all $E_{q,p}$ with $q \sim p$ in time $O(mn)$.

Proof. Clearly, the initialization and the recursion for $E_{q,p}$ are straightforward implementations of its definition (see Definition 3). Moreover, each individual call takes only time $O(|\Sigma|)$ besides the time taken for the recursive calls. Since each call computes one entry in the matrix and no entry is ever recomputed, we obtain the time complexity $O(|\Sigma| \cdot n^2) = O(mn)$ because $m = |\Sigma| \cdot n$. \square

In addition, we need to compute the number of strings that lead to a preamble state (see Lemma 2). This can easily be achieved with a folklore algorithm (see Algorithm 3 and Lemma 4 of [4]) that computes the number of paths from q_0 to

8 *A. Maletti and D. Quernheim*

Algorithm 3 COMPACCESS: Compute the number of paths to a preamble state.

Require: a minimal DFA $M = (Q, \Sigma, q_0, \delta, F)$ and a preamble state $q \in \text{Pre}(M)$

Global: access path vector $w \in \mathbb{N}^Q$ initially 1 at q_0 and 0 elsewhere

```

if  $w_q = 0$  then
2:   $w_q \leftarrow \sum_{(p,\sigma) \in \delta^{-1}(q)} \text{COMPACCESS}(M, p)$  // recursive calls
return  $w_q$  // return the computed value

```

each preamble state. Mind that the graph of the DFA M restricted to its preamble states $\text{Pre}(M)$ is acyclic. Overall, the algorithm is very similar to Algorithm 2, but we will not present a formal comparison here.

Theorem 6 (see [4]) *Algorithm 3 can be used to compute the number of paths to each preamble state in time $O(m)$.*

Proof. The correctness is obvious using the observation that p is a preamble state for every $(p, \sigma) \in \delta^{-1}(q)$ with $q \in \text{Pre}(M)$. Clearly, the call $\text{COMPACCESS}(M, q)$ terminates in constant time if the value w_q has already been computed. Moreover, each transition can be considered at most once in the sum in line 2, which yields the time complexity $O(m)$. \square

Algorithm 3 computes the following values for the DFA M_{ex} of Fig. 3:

$$w_0 = w_A = w_B = w_D = 1 \quad w_C = 2 \quad w_G = 3 \quad w_H = 6 .$$

Overall, we can now efficiently compute the number of errors (or a representation of the errors itself) caused by a single merge operation. However, multiple merges may affect each other. An error that is introduced by one merge might be removed by a subsequent merge, so that we cannot simply obtain the exact error count by adding the error counts for all performed merges.

5. Optimal state merging

The previous section suggests how to compute a hyper-optimal DFA for a given minimal DFA $M = (Q, \Sigma, q_0, \delta, F)$ with $m = |Q \times \Sigma|$ and $n = |Q|$. We can simply compute the exact set of errors for each hyper-minimal DFA for M and select a DFA with a minimal error count. By Theorem 1 we can easily enumerate all hyper-minimal DFAs for M , so that the above procedure would be effective. However, in this section, we show that we can also obtain a hyper-optimal DFA using only local decisions. This is possible since the structural differences among hyper-minimal DFAs for M mentioned in Theorem 1 cause different errors. Roughly speaking, Theorem 1 shows that two hyper-minimal DFAs for M can only differ on

- the initial state,

- finality of preamble states, and
- transitions from preamble to kernel states.

Now, let us identify the strings and potential errors associated with each of the three differences. Recall that \sim is the almost-equivalence relating the states of M . To simplify the following discussion, we introduce some additional notation. For every $q \in Q$, let $K_q = \{p \in \text{Ker}(M) \mid p \sim q\}$. In other words, the set K_q contains all kernel states that are almost-equivalent to the state q . Moreover, let $P_{\sim} = \{B \in [Q]_{\sim} \mid B \subseteq \text{Pre}(M)\}$ be the set of blocks of almost-equivalent and exclusively preamble states. Now we define sets of strings that correspond to the three types of differences mentioned above:

- Let $W_0 = \bigcup_{q \in K_{q_0}} \Sigma^*$.
- Let $W_B = \bigcup_{q \in B} L(M, q)$ for every $B \in P_{\sim}$.
- For every $B \in P_{\sim}$ and $\sigma \in \Sigma$ with $\bigcup_{q \in B} K_{\delta(q, \sigma)} \neq \emptyset$, let

$$W_{B, \sigma} = \{u\sigma w \mid u \in W_B, w \in \Sigma^*\} .$$

Lemma 7. *The following is a weak partition of Σ^* :*

$$\{W_0\} \cup \{W_B \mid B \in P_{\sim}\} \cup \{W_{B, \sigma} \mid B \in P_{\sim}, \sigma \in \Sigma, \bigcup_{q \in B} K_{\delta(q, \sigma)} \neq \emptyset\} .$$

Proof. Clearly, $W_0 = \Sigma^*$ if $K_{q_0} \neq \emptyset$ or $W_0 = \emptyset$ otherwise. Suppose the former; i.e., there exists $q \in K_{q_0}$. Let $p \in \text{Pre}(M)$ be a preamble state. Since M is minimal, there exists a string $w \in L(M, p)$. Moreover, $p = \underline{\delta}(q_0, w) \sim \underline{\delta}(q, w)$ because $q_0 \sim q$ and \sim is a congruence. Clearly, $\underline{\delta}(q, w)$ is a kernel state due to the fact that q is a kernel state. Consequently, every preamble state $p \in \text{Pre}(M)$ is almost-equivalent to some kernel state, which proves that $[p]_{\sim} \notin P_{\sim}$ for every $p \in \text{Pre}(M)$. This yields that the statement is correct if $K_{q_0} \neq \emptyset$.

In the second case, let $K_{q_0} = \emptyset$. Then $W_0 = \emptyset$. Clearly, $W_{B_1} \cap W_{B_2} = \emptyset$ for all different $B_1, B_2 \in P_{\sim}$ because $\{L(M, q) \mid q \in Q\}$ is a partition of Σ^* . Using the same reasoning, we can show that W_{B_1} and $W_{B_2, \sigma}$ are disjoint for all $B_1, B_2 \in P_{\sim}$ and suitable $\sigma \in \Sigma$ using the additional observation that $K_{\underline{\delta}(q_0, w)} \neq \emptyset$ for every $w \in W_{B_2, \sigma}$, whereas $K_{\underline{\delta}(q_0, w)} = \emptyset$ for every $w \in W_{B_1}$. Finally, let $B_1, B_2 \in P_{\sim}$ and suitable $\sigma_1, \sigma_2 \in \Sigma$. Suppose that there exists $w \in W_{B_1, \sigma_1} \cap W_{B_2, \sigma_2}$. When processing w by M there can only be one transition from a preamble state to a kernel state, which in both cases has to be achieved by the letter $\sigma_1 = \sigma_2$. Moreover, the state before taking this transition is unique, which yields that also $B_1 = B_2$. Consequently, we have shown that all sets are disjoint.

It remains to prove that all of Σ^* is covered. Let $w \in \Sigma^*$ be an arbitrary string. If $K_{\underline{\delta}(q_0, w)} = \emptyset$, then $w \in W_{[\underline{\delta}(q_0, w)]_{\sim}}$. On the other hand, let $K_{\underline{\delta}(q_0, w)} \neq \emptyset$. Then there exists a prefix u of w such that $K_{\underline{\delta}(q_0, u)} \neq \emptyset$ and $K_{\underline{\delta}(q_0, v)} = \emptyset$ for all strict prefixes v of u . Then $w \in W_0$ if $u = \varepsilon$ and $w \in W_{[\underline{\delta}(q_0, v)]_{\sim}, \sigma}$ where $u = v\sigma$ and $\sigma \in \Sigma$. This concludes the proof. \square

The previous lemma shows that error strings in the mentioned sets are independent and cover all potential errors. For our example DFA M_{ex} of Fig. 3 we have

$$W_0 = \emptyset \quad W_{\{C,D\}} = \{aaa, aab, ab\} \quad W_{\{C,D\},a} = \{uaw \mid u \in W_{\{C,D\}}, w \in \Sigma^*\} .$$

Next we address all individual differences between hyper-minimal DFAs for M . We start with the initial state.

Lemma 8. *If $K_{q_0} \neq \emptyset$, then each hyper-minimal DFA for M is obtained by pruning $\text{merge}_M(q_0 \rightarrow q)$ for some $q \in K_{q_0}$. Moreover, it commits exactly $E_{q_0,q}$ errors.*

Proof. Let $N = (P, \Sigma, p_0, \mu, G)$ be a hyper-minimal DFA for M . By Theorem 1, the DFA N consists of only kernel states and is isomorphic to the subautomaton of M that is determined by $\text{Ker}(M)$. Moreover, $q_0 \sim p_0$, which yields that N is isomorphic to $\text{merge}_M(q_0 \rightarrow q)$ for some $q \in K_{q_0}$. By Lemma 2 we have that $L(q, M) = L(q, N) = L(N)$ and $L(M) = L(q_0, M)$. This yields that $L(M) \triangle L(N) = L(q_0, M) \triangle L(q, M)$, of which the size is $E_{q_0,q}$ by Lemma 4. \square

We can compute the number $E_{q_0,q}$ of errors caused by the merge of q_0 into an almost-equivalent kernel state $q \in K_{q_0}$ using Algorithm 2 of Section 4. This simple test is implemented in lines 1–2 of Algorithm 5.

Second, let us consider a block $B \in P_{\sim}$ of almost-equivalent preamble states. Such a block must eventually be merged into a single preamble state p in the hyper-minimal DFA N , for which we need to determine finality because the preamble states of two hyper-minimal DFAs for M are only related by a transition isomorphism (see Theorem 1).

Lemma 9. *Let $B \in P_{\sim}$ and $N = (P, \Sigma, p_0, \mu, G)$ be a hyper-minimal DFA for M . Then N commits either $\sum_{q \in B \cap F} w_q$ or $\sum_{q \in B \setminus F} w_q$ errors of W_B .*

Proof. The set W_B contains all strings that take the DFA M into some state of B . Moreover, all those strings take the hyper-minimal DFA N into a single state $p \in P$; i.e., $L(N, p) = W_B$ by Theorem 1. Let

$$W'_B = \{w \in W_B \mid w \in L(M)\} \quad \text{and} \quad W''_B = \{w \in W_B \mid w \notin L(M)\} ;$$

i.e., the partition into accepted and rejected strings (by M) of W_B , respectively. Consequently, it is sufficient to compare the size of those sets because if $p \in G$ (i.e., p is a final state of N), then all strings of W''_B are errors. This is due to the fact that they are rejected by M , but accepted by N . On the other hand, the strings of W'_B are errors if p is non-final. Finally

$$\begin{aligned} |W'_B| &= |\{w \in W_B \mid q \in F, w \in L(M, q)\}| \\ &= |\{w \in \Sigma^* \mid q \in B \cap F, w \in L(M, q)\}| = \sum_{q \in B \cap F} w_q , \end{aligned}$$

Algorithm 4 COMPFINALITY: Determine finality of a block of preamble states.

Require: a minimal DFA $M = (Q, \Sigma, q_0, \delta, F)$ and a block $B \in P_{\sim}$

Global: error count e

$(\bar{f}, f) \leftarrow \left(\sum_{q \in B \cap F} w_q, \sum_{q \in B \setminus F} w_q \right)$ // errors for non-final and final state

2: $e \leftarrow e + \min(\bar{f}, f)$ // add smaller value to global error count
 select $q \in B$ such that $q \in F$ if $\bar{f} > f$ // select appropriate state

4: **return** q // return selected state

and similarly, $|W_B''| = \sum_{q \in B \setminus F} w_q$. \square

Consequently, if and only if more strings are accepting (i.e., $|W_B'| > |W_B''|$), then the preamble state $p \in P$ of N should be accepting. This decision is codified in Algorithm 4. On our example DFA M_{ex} of Fig. 3 and the block $B = \{C, D\}$ it compares $W_B' = \{aaa, ab\}$ and $W_B'' = \{aab\}$, and thus decides that the state C of the DFA N_{ex} of Fig. 3 should be final. Note that Lemma 7 shows that the errors are distinct for different blocks B_1 and B_2 . All of the following algorithms will use the global variable e , which will keep track of the number of errors. Initially, it will be set to 0 and each discovered error will increase it. Finally, we assume that the vector $w \in \mathbb{N}^Q$ (see Algorithm 3) and the error matrix $E \in \mathbb{Z}^{Q \times Q}$ (see Algorithm 2) have already been computed and can be accessed in constant time.

Lemma 10. COMPUTEFINALITY(M, B, w) adds the smallest number of errors of W_B committed by a hyper-minimal DFA N for M . It runs in time $O(|B|)$ and returns a final state (of M) if and only if $W_B \subseteq L(N)$.

Proof. Algorithm 4 implements the method of Lemma 9 in the given run-time. \square

For the third criterion, let us again consider a block $B \in P_{\sim}$ of almost-equivalent preamble states and a symbol $\sigma \in \Sigma$ such that $\bigcup_{q \in B} K_{\delta(q, \sigma)} \neq \emptyset$. Clearly, $K_{\delta(q_1, \sigma)} = K_{\delta(q_2, \sigma)}$ for all $q_1, q_2 \in B$ because \sim is a congruence on M . We need to determine the kernel state that will be the new transition target. By Theorem 1 it has to be a kernel state because $\delta(q, \sigma)$ is almost-equivalent to a kernel state.

Lemma 11. Let $N = (P, \Sigma, p_0, \mu, G)$ be a hyper-minimal DFA for M , and let $B \in P_{\sim}$ and $\sigma \in \Sigma$ be such that $K = \bigcup_{q \in B} K_{\delta(q, \sigma)} \neq \emptyset$. Then the DFA N commits $\sum_{q \in B} w_q \cdot E_{\delta(q, \sigma), q'}$ errors of $W_{B, \sigma}$ for some $q' \in K$.

Proof. Since $W_{B, \sigma} = \{u\sigma v \mid u \in W_B, v \in \Sigma^*\}$, each string $w \in W_{B, \sigma}$ has a prefix $u\sigma$ with $u \in W_B$. Clearly, each $u \in W_B$ takes the DFA M into some state of B , and the hyper-minimal DFA N into a state $p \in P$ such that $L(N, p) = W_B$ by Theorem 1. Moreover, $\mu(p, \sigma) = p'$ for some $p' \in \text{Ker}(N)$, for which an equivalent

state $q' \in Q$ exists in M by Theorem 1 because the kernels of M and N are DFA isomorphic. Consequently, $L(p', N) = L(q', M)$ and N accepts the strings

$$\{u\sigma v \mid u \in W_B, v \in L(q', M)\} \subseteq W_{B,\sigma}$$

and rejects the remaining strings of $W_{B,\sigma}$. On the other hand, the DFA M accepts the strings $\bigcup_{q \in B} \{u\sigma v \mid u \in L(M, q), v \in L(\delta(q, \sigma), M)\} \subseteq W_{B,\sigma}$ and rejects the remaining strings of $W_{B,\sigma}$. Clearly, $\delta(q, \sigma) \sim q'$. Consequently, the errors are exactly $\bigcup_{q \in B} \{u\sigma v \mid u \in L(M, q), v \in L(\delta(q, \sigma), M) \triangle L(q', M)\} \subseteq W_{B,\sigma}$, which yields the $\sum_{q \in B} w_q \cdot E_{\delta(q, \sigma), q'}$ errors of $W_{B,\sigma}$ because the decomposition is unique. \square

Recall that w_q and $E_{q,p}$ have been pre-computed already. Next, we discuss the full merging algorithm (see Algorithm 5). The initial state is handled in lines 1–2. In lines 5–7 we first handle the already discussed decision for the finality of blocks B of preamble states and perform the best merge into state q . In lines 8–11 we determine the best target state for all transitions from a preamble to a kernel state. The smallest error count is added to the global error count in line 10 and the corresponding designated kernel state is selected as the new target of the transition in line 11. This makes all preamble states that are almost-equivalent to this kernel state unreachable, so they can be removed. On our example DFA M_{ex} of Fig. 3, we have that $\delta(C, a) = G$ is a transition from the block $\{C, D\} \in P_{\sim}$ to a kernel state. Consequently, we compare $\sum_{q \in \{C, D\}} w_q \cdot E_{\delta(q, a), q'}$ for all kernel states $q' \in K_G$:

$$\sum_{q \in \{C, D\}} w_q \cdot E_{\delta(q, a), I} = 2 \cdot 1 + 1 \cdot 1 = 3 \quad \text{and} \quad \sum_{q \in \{C, D\}} w_q \cdot E_{\delta(q, a), J} = 2 \cdot 4 + 1 \cdot 4 = 12 .$$

Theorem 12. *Algorithm 5 runs in time $O(mn)$ and returns a hyper-optimal dfa for M . In addition, the number of committed errors is returned.*

Proof. The time complexity is easy to check, so we leave it as an exercise. Since the choices (finality, transition target, initial state) are independent by Lemma 7, all hyper-minimal DFAs for M are considered in Algorithm 5 by Theorem 1. Consequently, we can always select the local optimum for each choice (using Lemmata 8, 9, and 11) to obtain a global optimum, which proves that the returned number is the minimal number of errors among all hyper-minimal DFAs. Mind that the number of errors would be infinite for a hyper-minimal DFA that is not almost-equivalent to M . Moreover, it is obviously the number of errors committed by the returned DFA, which proves that the returned DFA is hyper-optimal for M . \square

Corollary 13 (of Theorem 12) *For every DFA M we can obtain a hyper-optimal DFA for M in time $O(mn)$.*

6. Empirical results

In order to evaluate the algorithm, we compare it to another hyper-minimization algorithm [7] that does not aim for low error profile. Since the algorithm of [7] is

Algorithm 5 OPTMERGE: Optimal merging of almost-equivalent states.

Require: a minimal DFA $M = (Q, \Sigma, q_0, \delta, F)$ and its almost-equivalent states \sim

Global: error count e ; initially 0

```

if  $K_{q_0} \neq \emptyset$  then
2:   return  $((Q, \Sigma, \arg \min_{q \in K_{q_0}} E_{q_0, q}, \delta, F), \min_{q \in K_{q_0}} E_{q_0, q})$ 
    $N \leftarrow M$  where  $N = (P, \Sigma, p_0, \mu, G)$            // initialize output DFA
4:   for all  $B \in P_{\sim}$  do
    $q \leftarrow \text{COMPFINALITY}(M, B)$            // determine finality of merged state
6:   for all  $p \in B$  do
    $N \leftarrow \text{merge}_N(p \rightarrow q)$            // perform the merges
8:   for all  $\sigma \in \Sigma$  do
   if  $K = K_{\delta(q, \sigma)} \neq \emptyset$  then
10:     $e \leftarrow e + \min_{q \in K} \left( \sum_{p \in B} w_p \cdot E_{\delta(p, \sigma), q} \right)$            // add best error count
         $\mu(q, \sigma) \leftarrow \arg \min_{q \in K} \left( \sum_{p \in B} w_p \cdot E_{\delta(p, \sigma), q} \right)$            // update follow state
12:  return  $(N, e)$ 

```

(“don’t-care”) non-deterministic (in the selection of merge targets), we implemented a simple stack discipline, which always pops the first element. For a varying set of parameters, 100 random DFAs have been generated and run through both algorithms. The number of saved states as well as the number of errors are reported. First we explain how the test DFAs were generated, describe the experimental setup, and then present and discuss the results.

We use an algorithm based on the original algorithm in HANNEFORTH’s FSM<2.0> library [6], which generates random non-deterministic finite automata. This model is closely related to KARP’s model of random directed graphs (see Chapter 2 of [3] or [13] for a discussion of different models). The only difference is the introduction of an additional parameter: the *cyclicity* a . The complete set of parameters is as follows:

- $|Q|$ This integer limits the number of states in the non-deterministic automaton.
- $|\Sigma|$ This integer coincides with the number of alphabet symbols.
- d_δ Uniform probability determining whether a given transition $p \xrightarrow{\sigma} q$ exists; we call $d_\delta \cdot |Q|$ the transition density.
- d_F Uniform probability for a given state to be final.
- a This real-valued parameter $0 \leq a \leq 1$ controls the cyclicity by constraining “backward-pointing” transitions. In particular, if $a = 0$, then the automaton will be acyclic, and if $a = 1$, then all transitions are equally probable.

A non-deterministic automaton M is generated in the following way: (i) The set of states is $Q = \{0, 1, 2, \dots, |Q| - 1\}$ with initial state 0. (ii) A state $q \in Q$ is final if

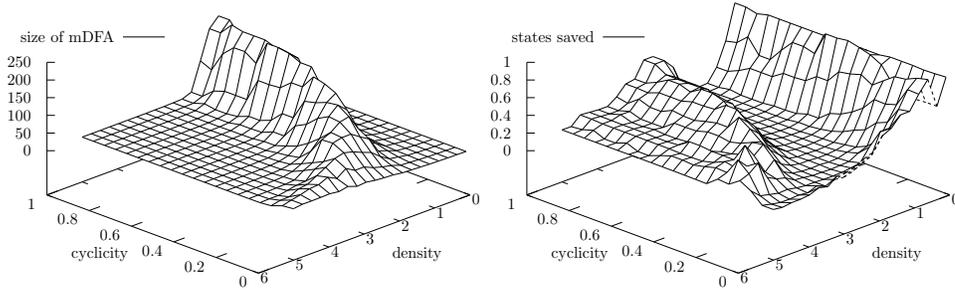


Fig. 4. Hyper-minimization performance for non-deterministic automata with 30 states, $|\Sigma| = 2$ and $0.3 \leq d_F \leq 0.7$. “Density” refers to $d_\delta \cdot |Q|$. Left: Average size of the minimal DFA. Right: Ratio of states saved by hyper-minimization. Values range over the full scale; i.e., they approach 0 outside the ridge and inside the valley.

and only if $f_q < d_F$, where $0 \leq f_q \leq 1$ is a random value. (iii) Finally, for every $(q, a, p) \in Q \times \Sigma \times Q$, we generate a random number $0 \leq f_{(q,a,p)} \leq 1$. The transition $q \xrightarrow{a} p$ is present in M if and only if

$$f_{(q,a,p)} < \begin{cases} d_\delta & \text{if } p > q \\ a \cdot d_\delta & \text{otherwise.} \end{cases}$$

The latter case corresponds to “backward-pointing” transitions and creates cycles.

For each set of parameters, we have generated 100 DFAs. These DFAs were obtained by determinizing and minimizing the randomly generated non-deterministic test automata. All DFAs have then been hyper-minimized, and the optimal hyper-minimal DFAs have been compared to the ones resulting from naïve hyper-minimization.^b The obtained results are shown in Figs. 4 and 5.

Figure 4 shows the size of the minimal DFAs and the potential of saving states by hyper-minimization. The left graph in Fig. 4 shows a ridge, which corresponds to cases in which DFA minimization is hard and results in a large minimal DFA [13]. It is located around a transition density of $d_\delta \cdot |Q| = 1.25$ for a cyclicity of 1, and it moves to higher densities for less cyclic automata. Essentially, the same ridge was observed by [13] (for the case $a = 1$). The right graph in Fig. 4 shows that these hard instances for DFA minimization are also hard for hyper-minimization in the sense that only very few states can be saved. However, for the remaining instances a considerable reduction in the number of states is achievable by hyper-minimization.

If we focus on the contribution of this paper, then we find that the number of errors can be considerably reduced. Figure 5 shows the absolute number of errors for hyper-minimal DFAs (left graph) and the ratio of errors avoided by the hyper-optimal automaton (right graph). The absolute number of errors for the hard instances, which can only be reduced a little, is higher than for the easy instances. However,

^bThe complete C++ source code will be made available, and the FSM<2.0> library is available at <http://tagh.de/tom/?p=1737>.

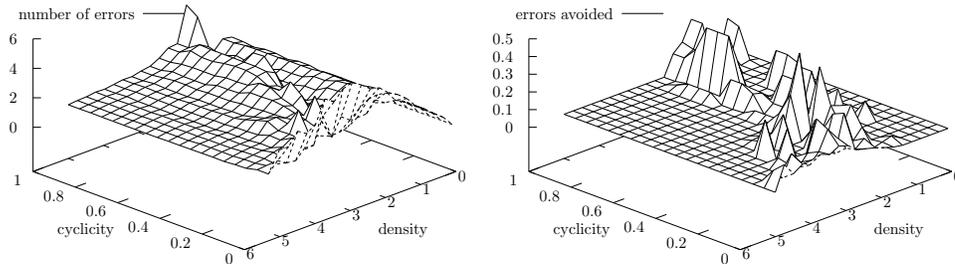


Fig. 5. Hyper-optimization performance. Left: Absolute number of errors in naïve hyper-minimal DFAs. Right: Ratio of errors avoided by hyper-optimization.

the hyper-optimal DFAs avoid a higher ratio of errors for the hard instances, which dramatically reduces the number of committed errors paid for the small reduction.

References

- [1] A. Badr, “Hyper-minimization in $O(n^2)$,” *Int. J. Found. Comput. Sci.* **20** (2009) 735–746.
- [2] A. Badr, V. Geffert and I. Shipman, “Hyper-minimizing minimized deterministic finite state automata,” *RAIRO Theor. Inf. Appl.* **43** (2009) 69–94.
- [3] B. Bollobás, *Random graphs* (Cambridge University Press, 2001).
- [4] D. Eppstein, “Finding common ancestors and disjoint paths in DAGs,” Tech. Rep. 95-52, University of California, Irvine, 1995.
- [5] 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.
- [6] T. Hanneforth, “*fsm2* — A scripting language interpreter for manipulating weighted finite-state automata,” in *Proc. 8th Int. Workshop Finite-State Methods and Natural Language Processing* (Springer, 2009), vol. 6062 of LNCS, pp. 13–30.
- [7] M. Holzer and A. Maletti, “An $n \log n$ algorithm for hyper-minimizing a (minimized) deterministic automaton,” *Theor. Comput. Sci.* **411** (2010) 3404–3413.
- [8] 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.
- [9] J. E. Hopcroft, R. Motwani and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison Wesley, 2007), 3rd edn.
- [10] C. D. Johnson, *Formal Aspects of Phonological Description*, no. 3 in Monographs on Linguistic Analysis (Mouton, The Hague, 1972).
- [11] M. Mohri, “Finite-state transducers in language and speech processing,” *Comput. Linguist.* **23** (1997) 269–311.
- [12] S. Schewe, “Beyond hyper-minimisation—minimising dbas and dpas is np-complete,” in *Proc. IARCS Ann. Conf. Foundations of Software Technology and Theoretical Computer Science* (Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2010), vol. 8 of *LIPICs*, pp. 400–411.
- [13] D. Tabakov and M. Y. Vardi, “Experimental evaluation of classical automata constructions,” in *Proc. 12th Int. Conf. Logic for Programming, Artificial Intelligence, and Reasoning* (Springer, 2005), vol. 3835 of LNCS, pp. 396–411.
- [14] S. Yu, “Regular languages,” in *Handbook of Formal Languages*, eds. G. Rozenberg and

16 *A. Maletti and D. Quernheim*

A. Salomaa (Springer, 1997), vol. 1, chap. 2, pp. 41–110.