Hyper-minimisation made efficient

Paweł Gawrychowski and Artur Jeż *

Institute of Computer Science, University of Wrocław, Poland
gawry@cs.uni.wroc.pl, aje@ii.uni.wroc.pl

Abstract. We consider a problem of hyper-minimisation of an automaton [2, 3]: given a DFA M we want to compute a smallest automaton N such that the language $L(M) \Delta L(N)$ is finite, where Δ denotes the symmetric difference. We improve the previously known $\mathcal{O}(|\Sigma|n^2)$ solution by giving an expected $\mathcal{O}(|\delta| \log n)$ time algorithm for this problem, where $|\delta|$ is the size of the (potentially partial) transition function. We also give a slightly slower deterministic $\mathcal{O}(|\delta| \log^2 n)$ version of the algorithm.
Then we introduce a similar problem of k-minimisation: for an automaton M and number k we want to find a smallest automaton N such that $L(M) \Delta L(N) \subseteq \Sigma^{<k}$, i.e. the languages they recognize differ only on words of length less than k. We characterise such minimal automata and give algorithm with a similar complexity for this problem.
Key-words: finite automata, minimisation, hyper-minimisation, cover automata

1 Introduction

DFA is the simplest device recognising languages known in the formal language theory. Studying its properties is motivated by simplicity of the notion, possible applications of the result, connections with various areas in theoretical computer science and the apparent beauty of the results of automata theory.

DFA is defined as a quintuple $\langle Q, \Sigma, \delta, q_0, F\rangle$, where Q is the (finite) state-set, Σ — (finite) alphabet, δ — the transition function, q_0 — starting state, and $F \subseteq Q$ is the set of accepting states. By the usual convention, n denotes $|Q|$.

One of the classical problem in the field is minimisation of a given automaton M: two automatons M, N are *equivalent*, denoted by $M \equiv N$, if $L(M) = L(N)$. An automaton M is *minimal* if for each equivalent N it holds that $|Q(M)| \geq |Q(N)|$). *Minimisation* of an automaton is is a problem of giving a smallest equivalent automaton. A breakthrough was made by Hopcroft [9], who gave an algorithm running in time $\mathcal{O}(n \log n)$. When the alphabet is not fixed, his algorithm runs in time $\mathcal{O}(|\Sigma| n \log n)$, as addressed directly by Gries [7].

A recent development in the area was done by considering a *partial function* δ instead of full transition function: whenever $\delta(q_0, w)$ is not defined, the word w is not accepted. Valmari and Lehtinen [12] gave an $\mathcal{O}(|\delta| \log n)$ algorithm in this case. As $|\delta| = \mathcal{O}(|\Sigma| n)$, this refines the previously existing results.

* Supported by MNiSW grants number N206 024 31/3826 2006–2009 and N N206 259035 2008–2010.

The question whether any algorithm faster than $\mathcal{O}(|\delta| \log n)$ for automata minimisation is possible remains a challenging open problem. In particular, no argument suggesting that minimisation cannot be done in linear time is known. On the other hand, it is known [5] that there are automata for which all possible executions of the Hopcroft algorithm run in $\Theta(n \log n)$.

A recent notion of f-equivalence [2, 3] considers a minimisation of an automaton while allowing the resulting language to differ from the original one on a finite amount of words. Languages L and L' are *f-equivalent*, denoted by $L{\sim}L'$, if $|L \Delta L'| < \infty$, where Δ denotes the symmetric difference of two languages. Similarly, automata M, M' are f-equivalent, denoted by $M{\sim}M'$, if $L(M){\sim}L(M')$. Automaton M is *hyper-minimal*, if for every $M'{\sim}M$ it holds that $|Q(M)| \leq |Q(M')|$. This springs a natural question: how difficult is it to *hyper-minimise* an automaton N, i.e. to construct a hyper-minimal automaton $M{\sim}N$. It is known that such construction can be done in time $\mathcal{O}(n^2)$ [2]. We improve its runtime to (expected) $\mathcal{O}(|\delta| \log n)$, which can be determinized and run in $\mathcal{O}(|\delta| \log^2 n)$. As minimisation reduces to the hyper-minimisation, any substantially faster algorithm would be a major breakthrough in the field.

We then introduce a similar notion of *k-f-equivalence*, denoted by $\sim_k$: $L{\sim_k}L'$ if $\max\{|w| : w \in L \Delta L'\} < k$. An automaton M is *k-minimal* if for all M' such that $L(M){\sim_k}L(M')$ it holds that M has the least number of states. Similarly we study the problem of k-minimisation of a given automaton M.

We introduce relations that allow to better understand the structure of the k-minimal automata and characterise them. Using them we give an algorithm for k-minimisation working in (expected) $\mathcal{O}(|\Sigma| n \log n)$ time. Note that the algorithm reads k as part of the input and the running time does not depend on k. As before it can be determinised and have $\mathcal{O}(|\Sigma| n \log^2 n)$ running time. Since hyper-minimisation of M is equivalent to n-minimisation, one cannot expect that any algorithm faster than $\mathcal{O}(|\delta| \log n)$ can be found.

It should be noted that the $\mathcal{O}(n^2)$ algorithm and our algorithms work in a different way then the Hopcroft's algorithm, which iteratively refined the partition of states. Both Badr's [2] and Badr et al. [3] algorithms calculated the equivalence classes of $\sim$ and then greedily merged the appropriate states. Our algorithm for hyper-minimisation, as well as the one for k-minimisation, works in phases: roughly speaking, in the ℓ-th phase it finds pairs of states q, q' such that $\max\{|w| : w \in L(q) \Delta L(q')\} = \ell$ and merges them.

The notion of k-f-equivalence is somehow complementary to the problem of finding a minimal *cover automata* — for an automaton M such that k is the length of longest word in $L(M)$ we want to find a minimal automaton N such that $L(N) \cap \Sigma^{\leq k} = L(M)$ [4]. It was introduced with practical purpose in mind: if an automaton recognises a finite language then one can store the length of the longest recognised word and a cover automaton. The input is accepted if it is not too long and recognised by the cover automaton. A clever modification of Hopcroft algorithm can be applied in this setting, yielding a $\mathcal{O}(n \log n)$ algorithm for finding a minimal cover automaton [10].

In parallel, a similar work concerning hyper-minimisation was done by M. Holzer and A. Maletti [8], who independently gave a randomised algorithm for hyper-minimisation running in expected time $\mathcal{O}(|\Sigma| n \log n)$.

2 Preliminaries

For an automaton $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ we define its language $L = L(M)$ in the usual sense. By $M(w)$ we denote $\delta(q_0, w)$. We say that a word w induces a language $L(w) := w^{-1}L$: the language recognised after reading the word w. Let also $L_M(q)$ denote $L(\langle Q, \Sigma, \delta, q, F\rangle)$, where $M = \langle Q, \Sigma, \delta, q_0, F\rangle$, i.e. the language recognized by the automaton M with the starting state set to q.

A standard approach to minimisation is to consider a Myhill-Nerode relation on words, defined for the language $L = L(M)$ as

$$w \equiv_L w' \quad \text{iff} \quad \forall u \in \Sigma^* \; wu \in L \iff wu' \in L.$$

This relation has finitely many equivalence classes for regular languages and each such class corresponds to one state in the minimal DFA recognising the language L. To make this approach more efficient, it should be noted that if $M(w) = M(w')$ then $w \equiv_L w'$, i.e. the relation is in fact defined for states: $q \equiv_L q' \iff L_M(q) = L_M(q')$. The minimisation algorithm starts with partition of states into two classes F and $Q \setminus F$ and iteratively refine the partition until it is left with the set of equivalence classes of $\equiv_L$ [9, 12].

It is easy to see that the minimisation reduces to hyper-minimisation: consider an automaton M. Create M' by adding one accepting state *dummy*, one fresh letter \$ to the alphabet and extend the transition function by $\delta_N(q, \$) = \mathit{dummy}$ and for every letter $b \in \Sigma$ $\delta_N(\mathit{dummy}, b) = \mathit{dummy}$. Then for each $q \in Q(M)$ it holds that $L_N(q) = L_M(q) \cup \$(\Sigma \cup \{\$\})^*$, hence it is infinite. Remove the state *dummy* from an automaton hyper-minimal for M'. The obtained automaton is minimal for $L(M)$.

The relation $\sim$ plays a similar role in the problem of hyper-minimisation as $\equiv$ in the problem of minimisation. We refine $\sim$ so that it can be used to the problem of k-minimisation as well.

We employ the classification of states developed in [3]: the *preamble*, denoted by $P(M)$, or simply P, is a set of states q reachable from q_0 by finitely many words, i.e. $L(\langle Q, \Sigma, q_0, \delta, \{q\}\rangle)$ is finite. On the other hand, *kernel* $K(M)$, or simply K, is defined as $Q \setminus P$, i.e. set of states q such that language $L(\langle Q, \Sigma, q_0, \delta, \{q\}\rangle)$ is infinite.

The automaton M' is obtained by *merging state* q *to state* p in automaton $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ if M' is obtained by changing all transitions ending in q to transitions ending in p and deleting state q. If q was a starting state then p is the new starting state. Formally $M' = \langle Q \setminus \{q\}, \Sigma, \delta', q_0', F \setminus \{q\}\rangle$ where

$$\delta'(r, a) = \begin{cases} \delta(r, a) & \text{if } \delta'(r, a) \neq q \\ p & \text{if } \delta'(r, a) = q \end{cases}, \quad q_0' = \begin{cases} q_0 & \text{if } q_0 \neq q \\ p & \text{if } q_0 = q \end{cases}.$$

For two languages L, L' define their *distance* as

$$d(L, L') = \begin{cases} \max\{|u| : u \in L(w) \Delta L(w')\} + 1 & \text{if } L \neq L' \ , \\ 0 & \text{if } L = L' \ . \end{cases}$$

This definition can be easily extended to states by setting $d(q, q') = d(L(q), L(q'))$. If we fix a language L then the distance between words is defined as $d_L(w, w') = d(L(w), L(w'))$. Usually the language L is clear from the context and so we drop the index L.

Fact 1 (d is a pre-ultrametric) *For all languages L, L', L'' it holds that $d(L, L'') \leq \max(d(L, L'), d(L', L''))$.*

The distance between the words allows relation between words similar to the f-equivalence: $w \sim u$ iff $d(w, u) < \infty$ and $w \not\sim u$ otherwise. This relation is *right invariant*: $u \sim w$ implies $ux \sim wx$ for all words x.

Fact 2 *$\sim$ is a right invariant equivalence relation.*

We extend the f-equivalence to states and automata: $q \sim q'$ iff for every w, w' such that $M(w) = q$ and $M(w') = q'$ it holds that $w \sim w'$; $M \sim M'$ iff $q_0 \sim q'_0$. Not that the definition for automata coincides with the one given in the Introduction: $M \sim M'$ iff $L(M) \Delta L(M')$ is finite.

We characterise the distance between two states in an operational manner. To this end for each $\ell \geq 0$ we introduce relation D_ℓ^M on $Q(M)$ defined by:

- $D_0^M(q, q')$ iff $q = q'$,
- $D_\ell^M(q, q')$ iff for all $a \in \Sigma$ either $D_{\ell-1}^M\big(\delta_M(q, a), \delta_M(q', a)\big)$ or both $\delta_M(q, a)$ and $\delta_M(q', a)$ are not defined.

Denote also $D^M(q, q') \iff \exists_\ell D_\ell^M(q, q')$. By an easy induction it follows that both D_ℓ^M and D^M are equivalence relations. We drop the upper index M whenever the automaton is clear from the context.

Fact 3 *If M is minimal and the transition function is always defined then $D_\ell(q, q')$ iff $d(q, q') \leq \ell$.*

To compute the relation D, one does not need to consider $\ell > n$:

Lemma 1. *Consider an automaton M. Then $D^M = D_n^M$.*

3 Hyper-minimisation

Badr et. al claimed [3, Thm. 3.2] that any (greedy) algorithm that at each step merges p to q such that $p \equiv q \vee (p \sim q \wedge p \in P(M))$ correctly hyper-minimises the automaton. Unfortunately this is not the case, still the argument can be easily corrected — some additional care is needed, when merging two states from the preamble. We say that a linear order $<$ on Q is *valid*, if for every pair of states $p, q \in P$ and letter $a \in \Sigma$ it holds that $\delta(q, a) = p$ implies $q < p$ and for $q \in P$, $p \in K$ it holds that $q < p$.

Theorem 1. *Any (greedy) algorithm that at each step merges p to q such that $p \equiv q \vee (p \sim q \wedge p \in P(M) \wedge p < q)$ correctly hyper-minimises the automaton.*

Such an order can be easily constructed — it is enough to sort topologically the acyclic graph corresponding to the states of the preamble and arbitrarily define it on the kernel.

Our algorithm utilises this approach, similarly to the previously known ones. Its novelty lays in the way pairs of states to be merged are found and the data structures that are employed.

On high level in the ℓ-th phase we calculate for the remaining states the relation D_ℓ on the remaining states, i.e. the distance between them. After that we merge some states and continue to the following phase. This approach result in no more than $|Q|$ phases, by Lemma 1.

3.1 Signatures

Since we deal with partial δ function, it is important to treat differently the states q of the automaton with different sets of defined transitions: for a state q the set $\{a : \delta(q, a) \text{ is defined}\}$ is the *signature* of q, denoted by $\mathrm{sig}(q)$. This allows bounding the running time by $|\delta|$ rather than $n|\Sigma|$.

We would like to think that $\mathrm{sig}(q) \neq \mathrm{sig}(q')$ implies that $q \not\sim q'$ and hence we can minimise states with different signatures separately. This can be achieved, whenever there are no states inducing finite languages.

Fact 4 *Suppose that automaton M has no states q such that $L(q)$ is finite. Then* $\mathrm{sig}(q) \neq \mathrm{sig}(q')$ *implies* $q \not\sim q'$.

The states inducing finite languages are naturally divided into those in the preamble and those in the kernel. The former can be easily removed.

Fact 5 *Let M be an automaton and M' be obtained from M by removing the set of states $P(M) \cap \{q : |L(q)| < \infty\}$. Then for $q \in Q(M')$ it holds that $L_M(q) \sim L_{M'}(q)$. In particular $L(M) \sim L(M')$.*

Unfortunately, such a straightforward approach cannot be applied to states from the kernel inducing finite languages. On one hand such states cannot be removed, as this causes the language of the automaton to change on infinitely many words. On the other hand, their existence prevents us from using Fact 4. An intermediate approach turns out to work, we can remove the problematic states temporarily, calculate the D classes and then bring back those states.

Lemma 2. *Suppose that automaton M is minimal and has no states $q \in P(M)$ such that $L(q)$ is finite. Let M' be obtained from M by removing the states $Q' = \{q : q \in K(M) \wedge |L(q)| < \infty\}$. Then for $q, q' \in Q \setminus Q'$ it holds that $D^{M'}(q, q') \iff L_M(q) \sim L_M(q')$.*

3.2 Automata reduction

Our algorithm has a high time-dependency on $|\Sigma|$. Thus we reduce the input so that we can treat $|\Sigma|$ as a small constant. To this end we transform the input automaton M into other one, denoted by $\textsc{Gadgets}(M)$, using only four-letter alphabet $\Sigma' = \{0, 1, 2, 3\}$. On the other hand, we increase the number of states from n to $\mathcal{O}(|\delta|)$. On a high level, one can imagine that we encode possible letters of Σ as 0–1 sequences. The letters 2, 3 are used to indicate which states simulate the states from the original automaton and which states are technical gadgets and distinguish states of different signatures. For simplicity of the presentation, we add a special non-accepting state *trash* such that each transition not defined explicitly goes to *trash*. Assume that the automaton M is minimised and does not have states inducing finite language. To shorten the notation, let M' denote $\textsc{Gadgets}(M)$. Please consult Fig. 1 and Fig. 2 for an illustration. Partition the set of states into subsets with the same signature. For each signature sig with ℓ symbols $\{a_0, \ldots, a_{\ell-1}\}$ introduce $\ell - 1$ new *auxiliary states* per each state q of this signature, named $s_{1,q}, s_{2,q}, \ldots s_{\ell-1,q}$. By convention, let $s_{0,q} = q$, we call it a *proper state*. Then define transition function as $\delta_{M'}(s_{i,q}, 0) = \delta_M(q, i)$ for $i = 0, \ldots \ell-1$; $\delta_{M'}(s_{i,q}, 1) = s_{i+1,q}$ for $i = 0, \ldots \ell-2$. The newly created states $s_{1,q}, s_{2,q}, \ldots s_{\ell-1,q}$ are assigned the signature $\mathrm{sig}(q)$.

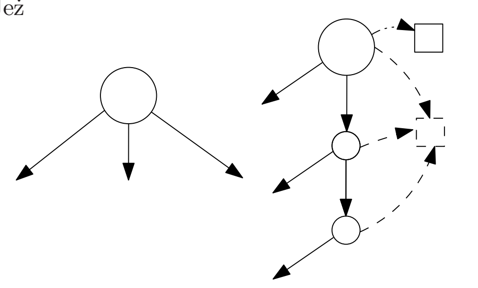

Fig. 1. Example of introducing gadgets on a single state.

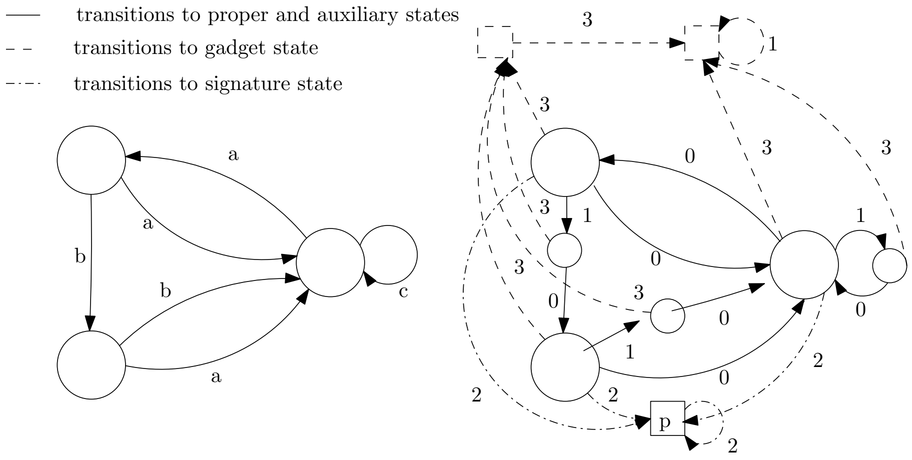

Fig. 2. Automaton M and its M'. There are two signatures.

Moreover, we distinguish proper states from auxiliary states by creating a *gadget state* p with $\delta_{M'}(p, 2) = p$ and adding the transition $\delta_{M'}(q, 2) = p$ for each state q. Then we distinguish states corresponding to different signatures by another gadget: let $\{\mathrm{sig}_1, \mathrm{sig}_2, \ldots, \mathrm{sig}_k\}$ be the set of all signatures. Introduce *signature states* $\{\mathrm{sig}_1, \mathrm{sig}_2, \ldots, \mathrm{sig}_k\}$ and add transitions $\delta_{M'}(\mathrm{sig}_i, 3) = \mathrm{sig}_{i+1}$ for $i = 1, \ldots, k-1$, $\delta_{M'}(\mathrm{sig}_k, 1) = \mathrm{sig}_k$ and $\delta_{M'}(s_{j,q}, 3) = \mathrm{sig}(q)$.

The size of the automaton M' is $\Theta(|\delta_M|)$ and M' can be constructed using in similar time bounds. The only nontrivial part is that we need to group states with the same signatures using counting sort. To upper bound the running time by $\mathcal{O}(|\delta|\log n)$ instead of $\mathcal{O}(|\delta|\log|\delta|)$ we show that the $D^{M'}$ classes are of size $\mathcal{O}(n)$.

Lemma 3. *Let $M' = \text{Gadgets}(M)$. Then none of the signature states nor the trash nor the state gadget is $D^{M'}$-equivalent to any other state. If $D^{M'}(s_{i,q}, s_{i',q'})$ then* $\text{sig}(q) = \text{sig}(q')$ *and* $i = i'$.

The automaton M' retains the basic properties of M, meaning that two states in M are f-equivalent iff they are equivalent in M'.

Lemma 4. *Let $q, q' \in Q(M)$ and $M' = \text{Gadgets}(M)$. Then $D^M(q,q')$ iff $D^{M'}(q,q')$.*

3.3 Algorithm and its running time

We now present Algorithm $\text{Comp-}f\text{-equiv}(M)$ calculating the hyper-minimal automaton f-equivalent to M.

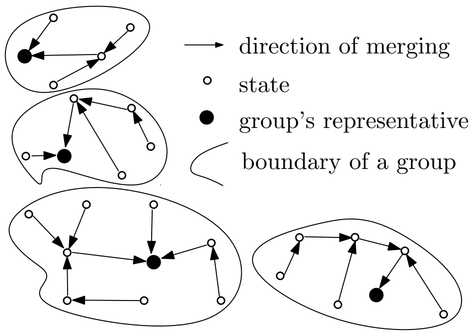

Fig. 3. Example of states in a group.

Preprocessing First of all we calculate $K(M)$ and $P(M)$. This takes just a linear time: we find the strongly connected components of the underlying graph and mark vertices that can be reached from those of them that are nontrivial (i.e. contain either more than two vertices or a loop).

Then we remove the states from the preamble inducing finite languages and minimise the automaton. Let us denote this automaton by M_1. We temporarily remove the states from the kernel of M_1 that induce finite languages and denote the result by M_2. Then we built $M' = \text{Gadgets}(M_2)$.

The algorithm also requires some simple data structures: for each state q a list of its predecessors $l[q]$

$$l[q] = \{q'' : \exists x \in \{0,1,2,3\} \text{ such that } \delta_{M'}(q'', x) = q\}$$

is created. Moreover, we store *rank* of q — the number of its predecessors.

The Theorem 1 requires a pre-computed linear valid order on Q. To fix such an order, sort topologically the states of the preamble and order the states of the kernel arbitrarily, such that they are all greater than the states of the preamble.

For technical reasons, we do not store the states in the dictionary, but rather their *dictionary representatives*, Dic. So a table of representatives is also created. In the beginning, for each state q, $\text{Dic}[q] = q$.

Dictionaries For the states of M' we built a dictionary mapping a tuple $\langle q_0, q_1, q_2, q_3 \rangle$ into a state q such that $\delta(q, i) = q_i$ for all i. Each state occurs at most once in the structure. To implement this dictionary, we use dynamic hashing with a worst-case constant time lookup and amortized expected constant time for updates (see [6] or a simpler variant with the same performance bounds [11]). We convert each tuple into an integer from $\{1 \mathinner{..} N\}$, where $N = |\delta|^4$ fits in a constant number of machine words, so we can hash it in a constant time. Whenever we insert a tuple already existing in the structure, we have a new pair of states to merge.

The analogous dictionary with $\mathcal{O}(\log n)$ time per operation is very simple for the deterministic case: note that by Lemma 3 if q, q' are merged then they are both either auxiliary states or proper states, they have the same signature and correspond to the same letter. So it is enough to built a separate dictionary for each set of states $Q_{\mathrm{sig}_j,i} = \{s_{i,q} : \mathrm{sig}(q) = \mathrm{sig}_j\}$. As there are only $\mathcal{O}(n)$ elements in such set by Lemma 3, it can be implemented as a balanced binary tree. In order to have a constant time access to the dictionary itself, we create a table of all signatures. For a single signature sig_j it keeps a pointer to the table indexed by $1 \ldots, \ell_j$, where ℓ_j is the size of the alphabet associated with sig_j. For an index i there is a pointer to the dictionary for the set $Q_{\mathrm{sig}_j,i}$. This $\mathcal{O}(\log n)$ bound per operation can be greatly improved if we are allowed to fully use the power of RAM model: plugging in the exponential search trees of [1] gives us a total running time of $\mathcal{O}(|\delta| \log n \frac{\log^2 \log n}{\log \log \log n})$.

Merging states Suppose two states $q > q'$ are to be merged. Let q_1 be the one of them with higher rank and q_1' the one with lower rank. We remove $\mathrm{Dic}[q_1']$ from the dictionary and keep only $\mathrm{Dic}[q_1]$ and set $\mathrm{Dic}[q]$ to $\mathrm{Dic}[q_1]$. The rank of q is set to be the sum of ranks q and q'.

Then we update the dictionary by reinserting all states q'' such that $\delta_{M'}(q'', x) = q_1'$ for some $x \in \{0, 1, 2, 3\}$. To do this efficiently, we scan $l[q_1']$. After the update $l[q']$ is appended to $l[q]$. We also store the information that q' was merged to q: a *group* consists of the states that were merged to a single state. A *representative* of the group is the unique state that survived the merging.

When all the merging is done, for each q we calculate the representative of q'es group: we inspect a sequence $q = q_0, q_1 \ldots, q_m$ such that q_i was merged to q_{i+1} and q_m was not merged to anything. Then $q_0, q_1 \ldots, q_{m-1}$ were all merged to q_m. All the states that were merged to a single q_m form a $D^{M'}$-class. This information can be used to merge the states from the $P(M_1)$ to their f-equivalent states, i.e. those that are in the same $D^{M'}$-class.

Theorem 2. Comp-f-equiv(M) *properly hyper-minimises the automaton* M *and runs in expected time* $\mathcal{O}(|\delta| \log n)$ *(*$\mathcal{O}(|\delta| \log^2 n)$ *worst-case).*

4 *k*-minimisation

We now consider the problem of *k-minimising* the automaton. Note that M and N are k-f-equivalent iff $d(L(M), L(N)) \leq k$. The general scheme is the same as previously, this time we are in more difficult situation: there is no notion similar to $\sim$ which works in the case of k-minimisation. Moreover, there is no theoretic characterisation of the k-minimal automaton. We begin with introducing a proper notion and describing the k-minimal automaton from a theoretical point of view. In particular we show that the k-minimal automaton can be obtained by merging some states into the others and the merging can be done in a (somehow) greedy fashion. Then we implement this approach. Unfortunately we were unable to efficiently deal with signatures in this case and the algorithm runs in $\mathcal{O}(|\Sigma| n \log n)$ time. We assume that the transition function is total.

4.1 Relation on states

We start with defining a relation playing the same role as $\sim$:

Definition 1. *We say that* $w \sim_k u$ *if* $d(w,u) = 0$ *or* $d(w,u) + \min(|w|,|u|) \le k$ *and* $w \not\sim_k u$ *otherwise.*

The intuition of this relation is similar to the one for $\sim$: consider any regular language L, an automaton M recognising it, and two words w, w'. Let $M(w) = q$, $M(w') = q'$. Suppose q is merged to q'. If $L(q) \neq L(q')$ then $wL(w) \subseteq L(M)$ is changed to $wL(w')$, so it should hold that $|w| + d(L(w), L(w')) \le k$. On the other hand if we were to merge q' to q then $w'L(w') \subseteq L(M)$ is changed to $w'L(w)$, hence it should hold that $|w'| + d(L(w), L(w')) \le k$. Choosing the smaller of those terms we obtain $\min(|w|,|w'|) + d(L(w), L(w')) \le k$, as in the definition of $\sim_k$. On the other hand, if $w \not\sim_k w'$ then it seems that we cannot merge the states q and q'. The second part of this intuition is formalised in Lemma 6. The first one needs some further refinements before it is put to work.

Note that $\sim_k$ is not an equivalence relation for any k. Still, it has some useful properties, which are quite close to being an equivalence relation and are essentially used in proofs of combinatorial properties and in the analysis of the algorithm for calculating the k-hyper-minimal automaton.

Lemma 5. *For all* k *the relation* $\sim_k$ *has the following properties*

1. *it is right invariant*
2. *if* $w_1 \sim_k w_2$*,* $w_2 \sim_k w_3$ *and* $|w_2| \ge \max(|w_1|,|w_3|)$ *then* $w_1 \sim_k w_3$
3. *if* $w_1 \sim_k w_2$*,* $w_2 \sim_k w_3$ *and* $|w_1| \le \min(|w_2|,|w_3|)$ *then* $w_1 \sim_k w_3$

As promised, sets of words $\{w_i\}_{i=1}^j$ such that for $w \neq w' \in \{w_i\}_{i=1}^j$ we have $w \not\sim_k w'$ can be used to lower-bound the size of the k-minimised automaton:

Lemma 6. *Consider* $L' \subseteq L$ *such that for each* $w \neq w' \in L'$ *it holds that* $w \not\sim_k w'$*. Then for each automaton* M *such that* $L \sim_k L(M)$ *it holds that* $M(w) \neq M(w')$*, in particular* M *has at least* $|L'|$ *states.*

The definition of $\sim_k$ can be easily extended to states: $q \sim_k q'$ if for all (w, w') such that $M(w) = q, M(w') = q'$ it holds that $w \sim_k w'$. One can easily see that this is equivalent to $q \sim_k q'$ when

$$d(q,q') = 0 \text{ or } d(q,q') + \min(\max(|w| : M(w) = q), \max(|w| : M(w) = q')) \le k$$

Instead of considering for a state q all the words w such that $M(w) = q$ it is enough to consider the longest one:

Definition 2. *For every state of $q \in Q$ let its* representative word *(called* word$[q]$*) be a longest word w such that $M(w) = q$ if $q \in P$ or any word with length at least k, if $q \in K$.*

This definition is designed in a way so that the $\sim_k$ equivalence between states could be expressed in terms of representatives of states:

Fact 6 word$[q] \sim_k$ word$[q']$ *iff* $q \sim_k q'$.

4.2 *k*-minimal automaton

We want to define a k-minimal automaton k-f-equivalent to M using relation $\sim_k$ on states of M. Since $\sim_k$ is not an equivalence relation, we need some additional refinement. To this end we construct an equivalence relation $\approx_k$ which refines $\sim_k$ defined on the set of states of M. Its equivalence classes correspond to states in the k-minimal automaton $M' \sim_k M$. The relation has the following properties:

1. $q \approx_k q'$ implies $q \sim_k q'$
2. for each class of abstraction $\{q_i\}_{i \in I}$ of $\approx_k$ we designate its representative word w — the longest of $\{\text{word}[q_i]\}_{i \in I}$. We denote by Rep$[q]$ the representative of q in its class of abstraction and we extend the notion of the representative word to words: Rep$[w]$ = Rep$[M(w)]$.
3. Rep$[q] \neq$ Rep$[q']$ implies Rep$[q] \not\sim_k$ Rep$[q']$

The relation is defined algorithmically. Let us first define Rep$[w_i] = w_i$ for all w_i such that $w_i =$ word$[q_i]$ for some state q_i. Then consider $\{w_1, \ldots, w_n\} = \{\text{word}[q]\}_{q \in Q}$ in any order. If for considered w_i there exists w_j such that Rep$[w_j] = w_j$, $w_i \sim_k w_j$ and $|w_i| \leq |w_j|$ then for all w such that Rep$[w] = w_i$ set Rep$[w] = w_j$ (choose any such j if there are many possible ones). In the end we extend the notion for states: Rep$[q]$ = Rep[word$[q]$] and set $q \approx_k q'$ if Rep$[q]$ = Rep$[q']$.

Fact 7 *The relation defined above satisfies conditions* (1)–(3)*.*

Consider again minimised automaton M, using relation $\approx_k$ on its states we define an automaton $N = \textsc{Equivalence-To-Automaton}(M, \approx_k)$ and later show how to create it efficiently. Construct an automaton N by taking

$$Q_N = \{\langle w \rangle : \text{Rep}[w] = w\} \quad \delta(\langle w \rangle, x) = \langle \text{Rep}[wx] \rangle \quad F_N = \{\langle w \rangle : M(w) \in F\}$$

and set a starting state $\langle \text{Rep}[\epsilon] \rangle$. By property 3 of $\approx_k$ one can easily derive that N is k-minimal.

The natural attempt to show that $L(N) \sim_k L(M)$ is to prove that $N(w) = \langle \text{Rep}[w] \rangle$. Unfortunately this is not the case. We proceed in a slightly more complicated fashion. First we argue that the defined automaton behaves well on short words and then show that $d(L_M(w), L_N(\langle w \rangle))$ can be upper-bounded in terms of $|w|$. Those facts allow proving that $L(N) \sim_k L(M)$.

Theorem 3. *Automaton N is k-minimal and $N \sim_k M$.*

4.3 Algorithm

We show how to efficiently calculate N = EQUIV-TO-AUTO. An algorithm similar to COMP-f-EQUIV is used. As we do not use different signatures for different states, we do not employ gadgets for signatures and additional symbol 3 in the alphabet. On the other hand we have to be a little more subtle now, as we are interested in calculating the classes of relation D_ℓ^M and not only $D^{M'}$. Roughly speaking in the ℓ-th phase we calculate the equivalence classes of D_ℓ^M. To this end we merge states not in some arbitrary fashion but in order representing the inductive definition of D_ℓ.

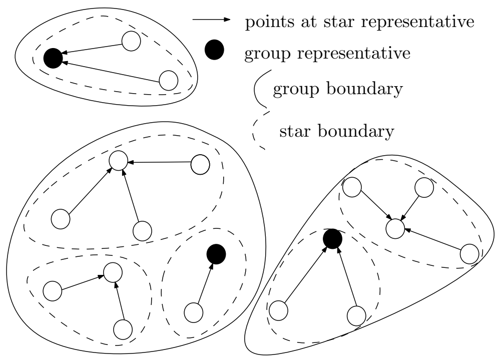

Fig. 4. Example of states in a group and in a star.

The algorithm works similar to COMP-EQUIV-k. We list only the important differences. We first minimise the automaton, obtaining M_1 and then introduce gadgets as in Section 3.2, except for the signature gadgets, which are not needed: M' = GADGETS(M_1). The algorithm works in phases. In one phase it first merges all the proper states that could be merged at the beginning of this phase. Then it starts merging the auxiliary states and gadget states. When there are no more auxiliary states to merge, the counter is increased and the next phase begins. Also more information is stored. A group of states that were merged together is subdivided into stars. Each star has its representative (star-representative). In particular group-representative is one of the star-representatives of stars forming this group.

If two groups, represented by q and q', are merged, we check whether $q \sim_k q'$. If so then two stars represented by q and q' are merged and assigned the longer of the two representative words. Note that this representative also becomes the new group representative.

When merging of groups is finished, N = EQUIV-TO-AUTO(M_1, star) is built, treating the states in one star as states in one equivalence class of $\approx_k$.

The running time analysis of COMP-EQUIV-k is the same as the one of COMP-f-EQUIV. All additional operations are done in constant time per operation.

The following lemma formalises the intuition that auxiliary states do not influence the process of merging state and thus phases correspond to calculating the distance between the states of the automaton.

Lemma 7. *Two states $q, q' \in Q(M_1)$ are merged in ℓ-th phase of* COMP-EQUIV-k *iff $d(q, q') \leq \ell$.*

To prove the correctness of the algorithm we show that the following invariants concerning groups and stars are kept: the first two invariants describe properties of stars, the following four the properties of groups.

1. Let $q_1, \ldots, q_i$ be all the states in a star. Then $|\,\mathrm{word}[q_1]| \geq |\,\mathrm{word}[q_j]|$ for $j = 2, \ldots, i$,
2. $q_j \sim_k q_{j'}$ for all $j, j' = 1, \ldots, i$.
3. group is a union of stars
4. Let $p_1, \ldots, p_{i'}$ be the star-representatives of star forming a group represented by p_1. Then $|\,\mathrm{word}[p_1]| \geq |\,\mathrm{word}[p_j]|$ for $j = 2, \ldots, i'$,
5. $p_j \not\sim_k p_{j'}$ for all $j \neq, j' \in \{1, \ldots, i'\}$,
6. the group consisting of proper states is an equivalence class of $D_\ell^{M_1}$

Theorem 4. COMP-EQUIV-k *correctly k-minimises M.*

Open problem

Is there a fully deterministic algorithm which hyper-minimize (or maybe even k-minimise) an automaton in time $\mathcal{O}(|\Sigma| n \log n)$?

References

1. A. Andersson and M. Thorup. Dynamic ordered sets with exponential search trees. *J. ACM*, 54(3):13, 2007.
2. A. Badr. Hyper-minimization in $\mathcal{O}(n^2)$. In O. H. Ibarra and B. Ravikumar, editors, *CIAA*, volume 5148 of *LNCS*, pages 223–231. Springer, 2008.
3. A. Badr, V. Geffert, and I. Shipman. Hyper-minimizing minimized deterministic finite state automata. *RAIRO - Theoretical Informatics and Applications*, 43(1):69–94, Jan 2009.
4. C. Câmpeanu, N. Santean, and S. Yu. Minimal cover-automata for finite languages. *Theor. Comput. Sci.*, 267(1-2):3–16, 2001.
5. G. Castiglione, A. Restivo, and M. Sciortino. Hopcroft's algorithm and cyclic automata. pages 172–183, 2008.
6. M. Dietzfelbinger, A. R. Karlin, K. Mehlhorn, F. M. auf der Heide, H. Rohnert, and R. E. Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM J. Comput.*, 23(4):738–761, 1994.
7. D. Gries. Describing an algorithm by Hopcroft. *Acta Informatica*, 2:97–109, 1973.
8. M. Holzer and A. Maletti. An n log n algorithm for hyper-minimizing states in a (minimized) deterministic automaton. In *CIAA*, pages 4–13, 2009.
9. J. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. *Technical Report*, CS-190, 1970.
10. H. Körner. On minimizing cover automata for finite languages in $\mathcal{O}(n \log n)$ time. In J.-M. Champarnaud and D. Maurel, editors, *CIAA*, volume 2608 of *LNCS*, pages 117–127. Springer, 2002.
11. R. Pagh and F. F. Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.
12. A. Valmari and P. Lehtinen. Efficient minimization of DFAs with partial transition. In S. Albers and P. Weil, editors, *STACS*, volume 08001 of *Dagstuhl Seminar Proceedings*, pages 645–656. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.