Hyper-minimisation made efficient.

Paweł Gawrychowski and Artur Jeż[1]

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 [1, 2]: 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 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 [7], 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 [6].

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 [10] gave an $\mathcal{O}(|\delta| \log n)$ algorithm in this case. As $|\delta| = \mathcal{O}(|\Sigma| n)$, this refines the previously existing results.

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 [4] 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 [1, 2] 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 $\mathcal{O}(n^2)$ [1]. 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. 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 [1] and Badr et al. [2] 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 $L(N) \cap \Sigma^{\leq k} = L(M)$ [3]. 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 and accept the word 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 [8].

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 $L(w) := w^{-1}L$: a language that should be 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-Nerod 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: $q \equiv_L q' \iff L_M(q) = L_M(q')$. Then during the run of the minimisation algorithm, we start with partition of states into two classes F and $Q \setminus F$ and iteratively refine the partition until we are left with the set of equivalence classes of $\equiv_L$ [7, 10].

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. Thus if N' is hyper-minimal for N then removing the state *dummy* from N results in an automaton M' 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 [2]: 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 also the distance between words can be 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 us to state relation between words similar to the f-equivalence: we say that $w{\sim}u$ 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 as follows: $q{\sim}q'$ 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 need to characterise the distance between two states in a more operational manner. To this end for each $\ell \geq$ we introduce relation D^M_ℓ on $Q(M)$ defined by:

- $D^M_0(q, q')$ iff $q = q'$,
- $D^M_\ell(q, q')$ iff for all $a \in \Sigma$ either $D^M_{\ell-1}\big(\delta_M(q, a), \delta_M(q', a)\big)$ or both $\delta_M(q, a)$ and $\delta_M(q', a)$ are not defined.

Denote also $D^M(q, q') \iff \exists_\ell D^M_\ell(q, q')$. By an easy induction D^M_ℓ 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 ℓ greater than $|Q|$:

Lemma 1. *Consider an automaton M. Then $D^M = D^M_{|Q|}$.*

3 Hyper-minimisation

It is known [2, 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. Our algorithm also works in this fashion. Its novelty lays in the way pairs of states to be merged are found and the data structures that are employed.

On high level we iteratively calculate the distance between states. In the ℓ-th phase we calculate the relation D_ℓ on the remaining states. After that we can merge some states and continue to the following phase. This approach cannot result in 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 $\mathrm{sig}(q)$.

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 somehow in separate groups. Unfortunately, this is not the case. The problematic sets are states from the kernel inducing finite languages.

Fact 4 *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 $L(M) \sim L(M')$.*

Fact 5 *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 idea of our approach is that to bound the running time by $|\delta|$ rather than $n|\Sigma|$ we consider states of different signatures separately. By Fact 5 this can be done if there are no states inducing finite languages. Moreover, we can remove the states from the preamble that induce finite languages. The states that are in the kernel and induce finite languages pose a problem as on one hand they cannot be removed, on the other, their existence may cause states of different signatures to be f-equivalent. An intermediate approach turn out to work, we can remove the problematic states temporarily, calculate the D classes and then bring back the states from kernel that induce finite language. This is formalised in the following lemma.

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

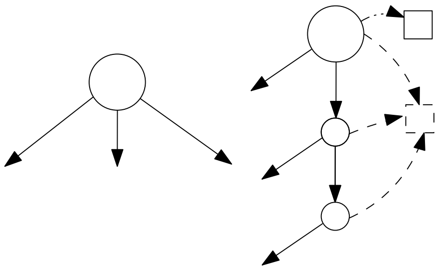

Fig. 1. Example of introducing gadgets on a single state.

Our algorithm has a relatively 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 Add-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

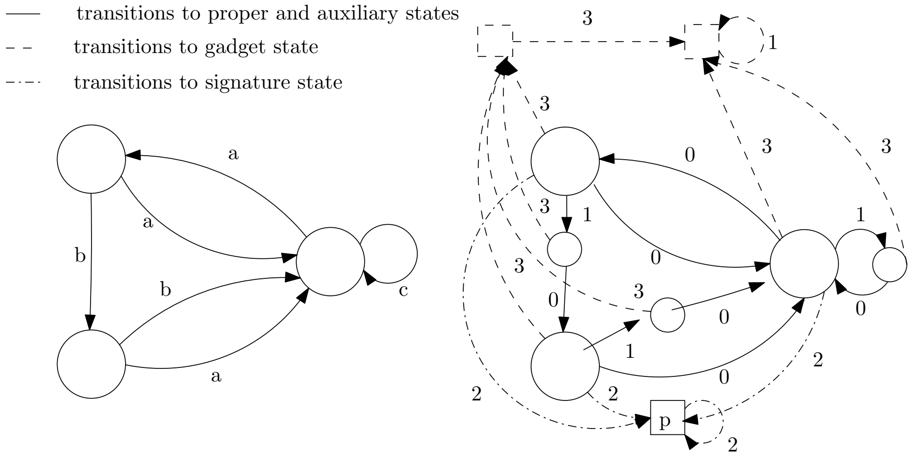

Fig. 2. Automaton M and its M'. There are two signatures.

is minimised and does not have states inducing finite language. To shorten the notation, let M'' denote Add-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 $s_{1,q}, s_{2,q}, \ldots s_{\ell,q}$ are assigned the signature $\mathrm{sig}(q)$.

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' = $ Add-Gadgets(M) is $\Theta(|\delta_M|)$ and it is easy to observe that it can be constructed in the same complexity (the only nontrivial part is that we need to group states with the same signatures using counting sort). We show that the $D^{M'}$ classes are of size $\mathcal{O}(n)$. This will later allow us to upper bound the running time by $\mathcal{O}(|\delta| \log n)$ instead of $\mathcal{O}(|\delta| \log |\delta|)$, which might be considerably larger.

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

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

3.3 Algorithm and Its Running Time

We now present Algorithm $\textsc{Compute-}f\textsc{-equivalent}(M)$ that calculates the hyper-minimal automaton f-equivalent to the one on the input. 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_2 that induce finite languages and denote the result by M_3. Then we built $M' = \textsc{Add-Gadgets}(M_3)$.

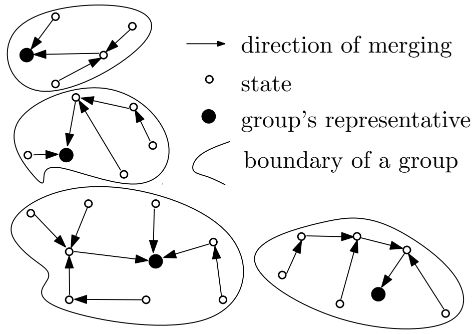

Fig. 3. Example of states in a group.

For the states of M' we built a dictionary which maps 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 [5] or a simpler variant with the same performance bounds [9]). We convert each tuple into an integer from $\{1\,..\,N\}$, 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 that already exists 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 $Q_{\mathrm{sig}_j, i}$.

When two states q, q' are to be merged, we join the one of the lower rank (wlg q') to the one of the higher rank (q) and update the rank to be the sum of ranks of q and q'. Then we update the dictionary by reinserting all states q''

such that $\delta_{M'}(q'', x) = q'$ for some $x \in \{0, 1, 2, 3\}$ has to be updated. To do this efficiently, for each state q we keep a list of its predecessors in M':

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

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 $P(M_2)$ to their f-equivalent states, i.e. those that are in the same $D^{M'}$-class.

Lemma 5. Compute-f-equivalent(M) *runs in expected time* $\mathcal{O}(|\delta| \log n)$ *and in worst-case time* $\mathcal{O}(|\delta| \log^2 n)$.

Theorem 1. Compute-f-equivalent(M) *properly hyper-minimises the automaton* M.

4 *k*-minimisation

We now consider the problem of k-*minimising* the automaton. Note that M and N are k-f-equivalent iff $d(L(M), L(N)) \le k$. The general scheme is the same as previously, this time we are in more difficult situation. First of all 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 show how to 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 (if it is not, we add transitions to a special trash state).

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 and automaton M recognising it. If $w \not\sim w'$ then for the hyper-minimal automaton $N \sim M$ it cannot hold that $N(w) = N(w')$. Thus the states $M(w)$,

$M(w')$ should not be merged in M in order to obtain N. Another obstruction in merging states is expressed by the notions of preamble and kernel. States from the kernel cannot be removed so in particular we cannot merge two states belonging the kernel. We employ similar approach for k-minimisation. Consider two words w, w' and let $M(w) = q$, $M(w') = q'$. Suppose that we would like to merge q to q'. If $L(q) \neq L(q')$ then $wL(w) \subseteq L(M)$ is changed to $wL(w')$, so it should hold that $|w| + d(L(w), L(w')) \leq k$. 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 $|w'| + d(L(w), L(w')) \leq k$. Choosing the smaller of those terms we obtain $\min(|w|, |w'|) + d(L(w), L(w')) \leq k$, 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 7. The first one needs some further refinements before it is put to work.

It is easy to see 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.

Lemma 6. *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 $|w_2| \geq \max(|w_1|, |w_3|)$ then $w_1 \sim_k w_3$*
3. *if $w_1 \sim_k w_2$, $w_2 \sim_k w_3$ and $|w_1| \leq \min(|w_2|, |w_3|)$ 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 7. *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

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

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 $\text{Rep}[q]$ the representative of q in its class of abstraction and we extend the notion of the representative word to words: $\text{Rep}[w] = \text{Rep}[M(w)]$.
3. $\text{Rep}[q] \neq \text{Rep}[q']$ implies $\text{Rep}[q] \not\sim_k \text{Rep}[q']$

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

Fact 7 *If w, w' are words for some states then* $\text{Rep}[w'] = \text{Rep}[w'']$ *implies* $w' \sim_k w''$ *and* $|\text{Rep}[w']| \geq |w'|$.

Fact 8 *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 $\langle \text{Rep}[\epsilon] \rangle$.

Fact 9 *N is k-minimal.*

We show that $L(N) \sim_k L(M)$. The natural attempt is to show that $N(w) = \langle \text{Rep}[w] \rangle$. Unfortunately this is not the case. We need to 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 to prove that $L(N) \sim_k L(M)$.

Lemma 8. *Suppose that M is minimised. If $|w| \geq k$ and* $\text{Rep}[w] = w$ *then* $L_M(w) = L_N(\langle w \rangle)$.

Lemma 9. *If $|w| \leq k$ and $N(w) = \langle u \rangle$ then $w \sim_k u$ and $|w| \leq |u|$.*

Lemma 10. *Let* $w = \text{Rep}[w]$ *then* $d(L_M(w), L_N(\langle w \rangle)) \leq \max(0, k - |w|)$.

Theorem 2. *Automaton N is k-minimal and $N \sim_k M$.*

4.3 Algorithm

We show how to efficiently calculate N = Equivalence-To-Automaton. An algorithm similar to Compute-f-equivalent 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.

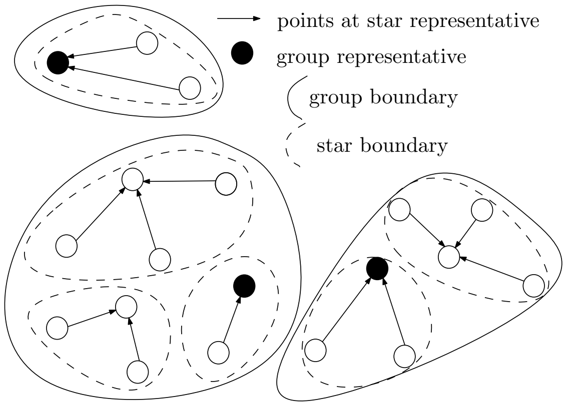

Fig. 4. Example of states in a group and in a star.

The algorithm works similar to Compute-equivalent-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' = Add-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, we built N = Equivalence-To-Automaton(M_1, star) treating the states in one star as states in one equivalence class of $\approx_k$.

The analysis of running time of Compute-equivalent-k is the same as the one of Compute-f-equivalent. All additional operations can be 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 11. *Two states $q, q' \in Q(M_1)$ are merged in ℓ-th phase of* Compute-equivalent-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 $|\operatorname{word}[q_1]| \geq |\operatorname{word}[q_j]|$ 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 $|\operatorname{word}[p_1]| \geq |\operatorname{word}[p_j]|$ 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 3. Compute-equivalent-k *correctly k-minimises the given automaton 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. 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.
2. 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.
3. C. Câmpeanu, N. Santean, and S. Yu. Minimal cover-automata for finite languages. *Theor. Comput. Sci.*, 267(1-2):3–16, 2001.
4. G. Castiglione, A. Restivo, and M. Sciortino. Hopcroft's algorithm and cyclic automata. pages 172–183, 2008.
5. 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.
6. D. Gries. Describing an algorithm by Hopcroft. *Acta Informatica*, 2:97–109, 1973.
7. J. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. *Technical Report*, CS-190, 1970.
8. 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.
9. R. Pagh and F. F. Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.
10. 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.

APPENDIX

A Proofs for Section 2

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

Proof. By induction on ℓ. If $\ell = 0$ then

$$D_0(q,q') \iff q = q' \iff L(q) = L(q') \iff d(q,q') \leq 0 \ .$$

If $\ell > 1$ then

$$d(q,q') \leq \ell \iff \forall_a d(\delta(q,a);\delta(q',a)) \leq \ell - 1 \iff \forall_a D_{\ell-1}(\delta(q,a);\delta(q',a)) \iff D_l(q,q') \ .$$

□

Lemma 1. *Consider an automaton M. Then $D = D_{|Q|}$.*

Proof. Let $d_0, d_1, \ldots$ be the numbers of equivalence classes of $D_0, D_1, \ldots$. By inductive definition of D_ℓ if $d_i = d_{i+1}$ then $d_i = d_j$ for all $j > i$ and $d(q,q') > i$ implies $d(q,q') = \infty$.

Since $d_0 \leq |Q|$ then the sequence $d_0, d_1, \ldots$ stabilises at position $d_{|Q|-1}$, i.e. there are no states q, q' such that $D_{|Q|}(q,q')$ and $\neg D_{|Q|-1}(q,q')$. Hence $D(q,q')$ implies $D_{|Q|}(q,q')$. □

B Proofs for Section 3

Fact 5 *Suppose that automaton M has no states q such that $L(q)$ is finite. Then* $\mathrm{sig}(q) \neq \mathrm{sig}(q')$ *implies $q \not\sim q'$.*

Proof. Since the signatures are different, then for some letter a $\delta(q,a)$ is defined and $\delta(q',a)$ is not. Then the language $L(\delta(q,a))$ is infinite. Hence the language $L(q) \cap a\Sigma^*$ is infinite and $L(q') \cap a\Sigma^*$ is empty. Hence $q \not\sim q'$. □

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 $D^{M'}(q,q') \iff L_M(q) \sim L_M(q')$.*

Proof. $\Rightarrow$ Suppose that $D^{M'}(q,q')$. Let ℓ be minimal such that $D^{M'}_\ell(q,q')$ for some ℓ. We prove by induction on ℓ that $L_M(q){\sim}L_M(q')$. If $\ell=0$ then $q=q'$ and $L_M(q)=L_M(q')$. Consider $\delta_{M'}(q,a)$ and $\delta_{M'}(q',a)$. If $\delta_{M'}(q,a)$ is not defined, then either $\delta_M(q,a)$ is not defined or $\delta_M(q,a)\in Q'$; in both cases $L_{M'}\big(\delta_M(q,a)\big)$ is finite. The same holds for $\delta_M(q',a)$. Hence $L_{M'}\big(\delta_M(q,a)\big){\sim}L_{M'}\big(\delta_M(q',a)\big)$, as they are both finite. On the other hand, if $\delta_{M'}(q,a)$ is defined, then also $\delta_{M'}(q',a)$ is defined and by definition of $D^{M'}_\ell$ it holds that

$$D^{M'}_{\ell-1}\big(\delta_{M'}(q,a);\delta_{M'}(q',a)\big)$$

Then by induction assumption

$$L_M\big(\delta_{M'}(q,a)\big){\sim}L_M\big(\delta_{M'}(q',a)\big)$$

Since this holds for every letter a, then $L_M(q){\sim}L_M(q)$.

$\Leftarrow$ Suppose that $L_M(q){\sim}L_M(q')$, we prove by induction on $d=d(L_M(q),L(q'))<\infty$ that $D^{M'}(q,q')$. If $d(q,q')=0$ then $q=q'$, as M is minimal. Thus $D^{M'}(q,q')$. Suppose that $d(q,q')=d>0$. Then for all $a\in\Sigma$ it holds that

$$d\Big(L_M\big(\delta_M(q,a)\big),L_M\big(\delta_M(q',a)\big)\Big)<d\ .$$

There are three cases: $\delta_M(q,a)$ may be not defined, $\delta_M(q,a)\in Q'$ or $\delta_M(q,a)\in Q\setminus Q'$; the same holds for q'. In the first two case the language $L_M(\delta_M(q,a))$ is finite, in the other it is infinite. Since

$$d(L_M(\delta_M(q,a)),L_M(\delta_M(q',a)))<\infty$$

then either both $L_M(\delta_M(q,a))$, $L_M(\delta_M(q',a))$ are finite or both are infinite.

Suppose first that both of them are finite. Then $\delta_{M'}(q,a)$ and $\delta_{M'}(q',a)$ are both not defined. So suppose that both the languages are infinite. Then $\delta_{M'}(q,a),\delta_{M'}(q',a)\notin Q'$ and hence they are defined. So by induction assumption $D^{M'}\big(\delta_{M'}(q,a);\delta_{M'}(q',a)\big)$.

We conclude that $D^{M'}(q,q')$. ◻

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

Proof. First note that it cannot hold that $D^{M'}(\mathit{trash},q)$ for $q\neq\mathit{trash}$: *trash* has transitions only to itself while every other state has a transition to a state different then *trash*.

Now by easy induction we show that sig_i is in different equivalence class of $D^{M'}$ then sig_{i+1}, …, sig_k. For inductive assumption consider $\delta_{M'}(\mathrm{sig}_{k-1},3)=\mathrm{sig}_k$ while $\delta_{M'}(\mathrm{sig}_k,3)=\mathit{trash}$. For inductive step note that $\delta_{M'}(\mathrm{sig}_i,3)=\mathrm{sig}_{i+1}$ while $\delta_{M'}(\mathrm{sig}_{i+j},3)=\mathrm{sig}_{i+j+1}$ (or *trash*) and those are different by the assumption.

Gadget state is not $D^{M'}$-equivalent to any signature state, as $\delta_{M'}(p,2) = p$ while $\delta_{M'}(q,2) = \mathit{trash}$ for q–signature state.

If q is a proper or auxiliary state then $\delta_{M'}(q,0) \neq \mathit{trash}$ while $\delta_{M'}(q',0) = \mathit{trash}$ for all other q'.

Consider any two states $s_{i,q}$ and $s_{j,q'}$ such that $D^{M'}(s_{i,q}, s_{j,q'})$ and suppose that $i \neq j$, so wlg. $i < j$. Then $\delta_{M'}(s_{j,q'}, 1^{\ell-j-1}03) = p$ while $\delta_{M'}(s_{j,q'}, 1^{\ell-j-1}03) = \mathit{trash}$. Contradiction, so $i = j$. Moreover, as $\delta_{M'}(s_{i,q}, 3) = \mathrm{sig}(q)$ and $\delta_{M'}(s_{i,q'}, 3) = \mathrm{sig}(q')$, hence $\mathrm{sig}(q) = \mathrm{sig}(q')$ □

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

Proof. $\Rightarrow$ Suppose that $D^M(q,q')$. Then $D^M_d(q,q')$ for some d. We proceed by induction on d. If $d = 0$ then $q = q'$ and the claim holds. So let $d > 1$. By construction $\delta_{M'}(q,2) = \delta_{M'}(q',2) = p$, $\mathrm{sig}(q) = \mathrm{sig}(q')$, $\delta_{M'}(q,3) = \delta_{M'}(q',3) = \mathrm{sig}(q)$. Let $\mathrm{sig}(q)$ have ℓ letters $\Sigma' = \{a_0, \ldots, a_{\ell-1}\}$. By definition of D^M_d, $D^M_{d-1}(\delta_M(q,a), \delta_M(q',a))$ for $a \in \Sigma'$ and $\delta_M(q,a) = \delta_M(q',a) = \mathit{trash}$ for $a \notin \Sigma'$.

Consider sequences 1^i0 for $i = 0, \ldots, \ell-1$. Then by the construction $\delta_{M'}(q, 1^i0) = \delta_M(q, a_i)$ and as $D^M_{d-1}(\delta_M(q,a_i), \delta_M(q',a_i))$ we obtain that $\delta_{M'}(q,1^i0) = \delta_{M'}(q',1^i0)$ by the induction assumption. Moreover, if a sequence x does not start with $2, 3$ or 1^i0 for $i \in \{0, \ldots, \ell-1\}$ then $\delta_{M'}(q,x) = \mathit{trash}$ by the construction of M'. Thus by easy induction we obtain that $D^{M'}_d(q,q')$.

$\Leftarrow$ Let $D^{M'}_d(q,q')$. We proceed by induction on d. If $d = 0$ then $q = q'$ and $D^M(q,q')$.

$D^{M'}(q,q')$ implies that $D^{M'}(\delta_{M'}(q,3), \delta_{M'}(q',3))$, but $\delta_{M'}(q,3) = \mathrm{sig}(q)$ and $\delta_{M'}(q',3) = \mathrm{sig}(q')$. Hence $\mathrm{sig}(q) = \mathrm{sig}(q')$. Let $a_0, \ldots, a_{\ell-1}$ be the associated alphabet. Otherwise, by repeatedly applying the definitions of $D^{M'}_{d-1}, D^{M'}_{d-2}, \ldots$ we obtain that $D^{M'}(\delta_{M'}(q,1^i0), \delta_{M'}(q,1^i0))$ for $i = 0, \ldots, \ell-1$. By the inductive assumption $D^M(\delta_M(q,a_i), \delta_M(q',a_i))$ for $i = 0, \ldots, \ell-1$, Since $\mathrm{sig}(q) = \mathrm{sig}(q')$ for all other letters a $\delta_M(q',a) = \delta_M(q',a) = \mathit{trash}$. Hence $D^M(q,q')$. □

Lemma 5. Compute-f-equivalent(M) *runs in expected time $\mathcal{O}(|\delta| \log n)$.*

Proof. Note that each time when q is merged to q' its rank at least doubles. On the other hand rank cannot be larger then $\mathcal{O}(n)$ — by easy induction rank of the vertex is the sum of ranks of all the vertices that were merged into it and by Lemma 3 each $D^{M'}$-equivalence class has size $\mathcal{O}(n)$. So each time $\delta(q,a)$ is changed, $\mathrm{rank}(\delta(q,a))$ doubles. Hence each $\delta(q,a)$ is redirected at most $\mathcal{O}(\log n)$ times. Summing up by all transitions this gives $\mathcal{O}(|\delta| \log n)$. Minimisation can be done in $\mathcal{O}(|\delta| \log n)$. All other operations clearly can be performed in amortised linear time.

If we use a deterministic dictionary, it gives another $\log n$ factor to each operation. □

Theorem 1. COMPUTE-f-EQUIVALENT(M) *properly hyper-minimises the automaton* M*.*

Proof. Removing the states from the preamble inducing finite languages can change the language recognised by the automaton only by a finite amount of words. Minimisation does not change the language recognised by the automaton. So it is enough to hyper-minimise M_2. Note that in M_3 there are no states in the preamble inducing finite languages.

Clearly COMPUTE-f-EQUIVALENT(M') properly computes the $D^{M'}$ equivalence classes. By Lemma 4 if $D^{M'}(q,q')$ then also $D^{M_3}(q,q')$, hence also D^{M_3} classes are properly computed. Let $Q'(M_2) = \{q : q \in K(M_2)$ and q induces finite language$\}$. Then on $Q(M_2)\backslash Q'(M_2)$ the D^{M_2} classes are also correctly computed, as M_2 has no states in the preamble inducing a finite language. Consider M_2 and two of its states q, q' such that $q \in P(M)$ and $q \sim q'$. Since $L_{M_2}(q)$ is infinite then $q \notin Q'$, as all states in Q' induce finite languages. in particular by Lemma 2 $q \sim_{M_2} q'$ is equivalent to $D^{M_3}(q,q')$. By [2, Thm. 3.2] the hyper-minimal automaton can be obtained by greedily merging q to p, whenever $q \sim_{M_2} p$ and q is in the preamble. And this is exactly what the algorithm does. Hence the obtained automaton is the hyper-minimal automaton f-equivalent to M. □

C Proofs for Section 4

Lemma 6. *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 $|w_2| \geq \max(|w_1|, |w_3|)$ then $w_1 \sim_k w_3$*
3. *if $w_1 \sim_k w_2$, $w_2 \sim_k w_3$ and $|w_1| \leq \min(|w_2|, |w_3|)$ then $w_1 \sim_k w_3$*

Proof. Assume that $u \sim_k w$, let us check that $ux \sim_k wx$ for $x \in \Sigma$. If $L(ux) = L(wx)$ then clearly $ux \sim_k ux$. If $ux \not\sim_k ux$ then $L(ux) \neq L(wx)$ and also $L(u) \neq L(w)$. Then by definition of $\sim_k$ it holds that $\min(|u|, |w|) + d(w, u) \leq k$. Let us calculate the similar expression for ux and wx:

$$\begin{aligned}
\min(|ux|, |wx|) + d(wx, ux) &= \min(|u|, |w|) + 1 + (\max\{|v| : v \in L(ux) \Delta L(wx)\} + 1) \\
&= \min(|u|, |w|) + (\max\{1 + |v| : v \in L(ux) \Delta L(wx)\} + 1) \\
&= \min(|u|, |w|) + (\max\{|xv| : v \in L(ux) \Delta L(wx)\} + 1) \\
&= \min(|u|, |w|) + (\max\{|xv| : xv \in L(u) \Delta L(w)\} + 1) \\
&\leq \min(|u|, |w|) + (\max\{|v| : v \in L(u) \Delta L(w)\} + 1) \\
&= \min(|u|, |w|) + d(u, w) \leq k \ .
\end{aligned}$$

Consider now condition 2. Firstly if $L(w_1) = L(w_2)$ then

$$\min(|w_1|, |w_3|) + d(w_1, w_3) \leq \min(|w_2|, |w_3|) + d(w_1, w_3) = \min(|w_2|, |w_3|) + d(w_2, w_3) \leq k \ .$$

If $L(w_2) = L(w_3)$ then

$$\min(|w_1|, |w_3|) + d(w_1, w_3) = \min(|w_1|, |w_3|) + d(w_1, w_2) \leq \min(|w_1|, |w_2|) + d(w_1, w_2) \leq k \ .$$

So suppose that $L(w_1) \neq L(w_2)$ and $L(w_2) \neq L(w_3)$. Then

$$\begin{aligned}\min(|w_1|, |w_2|) + d(w_1, w_2) = |w_1| + d(w_1, w_2) \leq k \ ,\\ \min(|w_2|, |w_3|) + d(w_2, w_3) = |w_3| + d(w_2, w_3) \leq k \ .\end{aligned}$$

Let us check that $w_1 \sim_k w_3$:

$$\begin{aligned}\min(|w_1|, |w_3|) + d(w_1, w_3) &\leq \min(|w_1|, |w_3|) + \max(d(w_1, w_2), d(w_2, w_3))\\ &= \max(\min(|w_1|, |w_3|) + d(w_1, w_2), \min(|w_1|, |w_3|) + d(w_2, w_3))\\ &\leq \max(|w_1| + d(w_1, w_2), |w_3| + d(w_2, w_3)) \leq k \ .\end{aligned}$$

Consider property 2 and its trivial cases: if $L(w_1) = L(w_2)$ then

$$\min(|w_1|, |w_3|) + d(w_1, w_3) = \min(|w_1|, |w_3|) + d(w_2, w_3) \leq \min(|w_2|, |w_3|) + d(w_2, w_3) \leq k \ .$$

If $L(w_2) = L(w_3)$ then

$$\min(|w_1|, |w_3|) + d(w_1, w_3) = |w_1| + d(w_1, w_3) = \min(|w_1|, |w_2|) + d(w_1, w_2) \leq k \ .$$

So suppose that $L(w_1) \neq L(w_2)$ and $L(w_2) \neq L(w_3)$. Then by assumption:

$$\begin{aligned}\min(|w_1|, |w_2|) + d(w_1, w_2) = |w_1| + d(w_1, w_2) \leq k \ ,\\ \min(|w_2|, |w_3|) + d(w_2, w_3) \leq k \ .\end{aligned}$$

Let us show that $w_1 \sim_k w_3$:

$$\begin{aligned}\min(|w_1|, |w_3|) + d(w_1, w_3) &= |w_1| + d(w_1, w_3)\\ &\leq \max(|w_1| + d(w_1, w_2), |w_1| + d(w_2, w_3))\\ &\leq \max(|w_1| + d(w_1, w_2), \min(|w_2|, |w_3|) + d(w_2, w_3)) \leq k \ .\end{aligned}$$

□

Lemma 7. *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.*

Proof. Suppose that for some $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ and for some $w \neq w' \in L'$ it holds that $M(w) = M(w')$. Since $w \not\sim_k w'$ then $\min(|w|, |w'|) + d(w, w') > k$, i.e. there is a word $u \in L(w) \Delta L(w')$, such that $\min(|w|, |w'|) + |u| \geq k$, so $|wu| \geq k$ and $|w'u| \geq k$. Wlg. assume that $u \in L(w) \setminus L(w')$. Then either both words $wu, w'u \in L(M)$ or for both words $wu, w'u \notin L(M)$. On the other hands $wu \in L$ and $w'u \notin L$. In both cases L and $L(M)$ differ on word of length at least k. □

Fact 7 *If w, w' are words for some states then* $\text{Rep}[w'] = \text{Rep}[w'']$ *implies* $w' \sim_k w''$ *and* $|\text{Rep}[w']| \geq |w'|$.

Proof. The thesis obvious is when the construction starts, as for all w it holds that $\text{Rep}[w] = w$. Suppose that when considering w_i there was w_j such that $\text{Rep}[w_j] = w_j$, $w_i \sim_k w_j$ and $|w_i| \leq |w_j|$. Then for any w such that $\text{Rep}[w] = w_i$ it holds that $|w| \leq |w_i|$ by induction assumption and $|w_i| \leq |w_j|$ by the case assumption. Moreover $w \sim_k w_i$ and $w_i \sim_k w_j$. Thus by Lemma 6(2) $w \sim_k w_j$. Hence the induction step ends. □

Fact 8 *The relation defined above satisfies conditions* (1)–(3)*.*

Proof. Condition (1): if $q \approx_k q'$ then by the algorithm $\text{Rep}[\text{word}[q]] \approx_k \text{Rep}[\text{word}[q']]$. By Fact 7 this implies $\text{word}[q] \sim_k \text{word}[q']$. Hence also $q \sim_k q'$ by Fact 6.

Condition (2) follows from the algorithm — when two classes of $\approx_k$ are merged we designate the longer of their representatives as the representative of the new class.

Condition (3) is obvious — if $\text{Rep}[q] \neq \text{Rep}[q']$ then words $\text{Rep}[q]$, $\text{Rep}[q']$ were considered and it turned out that $\text{Rep}[q] \not\sim_k \text{Rep}[q']$ □

Fact 9 *N is k-minimal.*

Proof. Consider set $L' = \{w : \langle w \rangle \in Q_N\}$. Then by properties of construction of N if $w, w' \in L'$ and $w \neq w'$ then $w \not\sim_k w'$. Then by Lemma 7 N is k-minimal. □

Fact 10 $\epsilon \sim_k \text{Rep}[\epsilon]$.

Proof. We know that $L(\epsilon) = L = L(q_0)$, hence $\epsilon \sim_k \text{word}[q_0]$. On the other hand $\text{Rep}[\epsilon] = \text{Rep}[\text{word}[q_0]]$, as $\text{Rep}[\epsilon]$ is the representative of q_0. Hence $\text{Rep}[\epsilon] \sim_k \text{word}[q_0]$ by condition (1) and $|\text{word}[q_0]| \leq |\text{Rep}[\epsilon]|$ by condition (2). Since $|\epsilon| \leq |\text{word}[q_0]| \leq |\text{Rep}[\epsilon]|$ then by Lemma 6(2) $\epsilon \sim_k \text{word}[q_0]$ and $\text{word}[q_0] \sim_k \text{Rep}[\epsilon]$ implies $\epsilon \sim_k \text{Rep}[\epsilon]$. □

Lemma 8. *Suppose that M is minimised. If $|w| \geq k$ and* $\text{Rep}[w] = w$ *then* $L_M(w) = L_N(\langle w \rangle)$.

Proof. Consider the set of states $Q'' = \{q : |\text{word}[q]| \geq k\}$. Consider $q \in Q''$ and $u = \text{word}[q]$. It holds that $\text{Rep}[u] = u$. Assume otherwise: suppose that $\text{Rep}[u] = u'$. Then $|u| \leq |u'|$ by Fact 7. Since the automaton M is minimised $L(w) \neq L(w')$. Hence $w \not\sim_k w'$, contradiction.

Since $q \in Q''$ implies that for all a also $\delta(q, a) \in Q''$, thus Q'' and its transition structure from M are kept in the resulting automaton N. So clearly for such word w it holds that $L_M(w) = L_N(\langle w \rangle)$.

Lemma 9. *If $|w| \leq k$ and $N(w) = \langle u \rangle$ then $w \sim_k u$ and $|w| \leq |u|$.*

Proof. By induction on $|w|$: if $w = \epsilon$, then $N(\epsilon) = \langle \mathrm{Rep}[\epsilon] \rangle$ and then by Fact 10 $\epsilon \sim_k \mathrm{Rep}[\epsilon]$.

Suppose now that $N(w) = \langle u \rangle$, $|w| \leq |u|$ and $w \sim_k u$. Consider any letter x. We show that $wx \sim_k ux$, $ux \sim_k v$ and $|wx| \leq |ux|, |v|$. Then by Lemma 6(2) this implies that $wx \sim_k v$.

By Lemma 6(1) $w \sim_k u$ implies $wx \sim_k ux$. Let $\delta_N(\langle u \rangle, x) = \langle v \rangle$. Then $\langle v \rangle = \mathrm{Rep}\, \langle ux \rangle$ by the construction of N, hence $v \approx_k ux$ by the construction and thus $v \sim_k ux$ by condition (2). If $|v| \geq k$ then $|v| \geq |wx|$. If $|v| < k$ then $M(v)$ is in the preamble of M, as all the states q in the kernel of M satisfy $|\,\mathrm{word}[q]| \geq k$. Then $|v| \geq |ux|$, as $M(ux) = M(v)$ and v is the representative of $M(v)$. Hence $|wx| = |w| + 1 \leq |u| + 1 \leq |v|$. □

Lemma 10. *Let $w = \mathrm{Rep}[w]$ then :*

$$d(L_M(w), L_N(\langle w \rangle)) \leq \max(0, k - |w|)$$

Proof. Note that $w = \mathrm{Rep}[w]$ means that $w = \mathrm{word}[q]$ for some q.

We prove the claim by a backward induction on $|w|$.

If $|w| \geq k$ then by Lemma 8 $d(L_N(w), L_M(\langle w \rangle)) = 0$.

Suppose now that $|w| < k$. Consider any letter x and let $N(wx) = \langle u \rangle$. By Lemma 9

$$d(L_M(wx), L_M(u)) + \min(|wx|, |u|) \leq k \ .$$

But again by Lemma 9 $|wx| \leq |u|$. Hence

$$d(L_M(wx), L_M(u)) \leq k - |wx| = k - |w| - 1 \ .$$

On the other hand by induction assumption $d(L_M(u), L_N(\langle u \rangle)) \leq k - |u| \leq k - |w| - 1$. Since d is pre-ultrametric, by Fact 1:

$$d(L_M(wx), L_N(\langle u \rangle)) \leq \max \Big(d(L_M(wx), L_M(u)); d\big(L_M(u), L_N(\langle u \rangle)\big)\Big) \leq k - |w| - 1$$

Then

$$d(L_M(w), L_N(\langle w \rangle)) = 1 + \max_{x \in \Sigma} d(L_M(wx), L_N(\langle \mathrm{Rep}[wx] \rangle)) \leq 1 + k - |w| - 1 = k - |w|.$$

□

Theorem 2. *Automaton N is k-minimal and $N \sim_k M$.*

Proof. By Fact 10 $\epsilon \sim_k \text{Rep}[\epsilon]$, hence $d(L(M), L_M(\text{Rep}[\epsilon])) \leq k$. By Lemma 10

$$d(L_M(\text{Rep}[\epsilon]), L_N(\langle \text{Rep}[\epsilon] \rangle)) \leq k$$

As d is a pre-ultrametric and $L(N) = L_N(\text{Rep}[\epsilon])$:

$$\begin{aligned} d(L(M), L(N)) &= d\Big(L(M), L_N\big(\langle \text{Rep}[\epsilon] \rangle\big)\Big) \\ &\leq \max\Big(d\big(L(M), L_M(\text{Rep}[\epsilon])\big); d\big(L_M(\text{Rep}[\epsilon]), L_N(\langle \text{Rep}[\epsilon] \rangle)\big)\Big) \leq k, \end{aligned}$$

thus M and N are k-f-equivalent.

By Fact 9 N is k-minimal. □

Lemma 11. *Two states $q, q' \in Q(M_1)$ are merged in ℓ-th phase of* Compute-equivalent-k *iff $d(q, q') \leq \ell$.*

Proof. Note that since the automaton M_1 is minimised and δ_{M_1} is total, $d_{M_1}(q, q') \leq \ell$ iff $D_\ell^{M_1}(q, q')$, by Fact 3. Let $s = |\Sigma|$. For the uniformity of the presentation assume that q, q' are merged in phase 0 iff $q = q'$. The proof follows from a more technical claim:

Claim. The states $q \neq q'$ are merged in phase

$$1 + \max_{j=0}^{s-1}\{\ell' : (\delta_{M'}(q, 1^j 0), \delta_{M'}(q', 1^j 0)) \text{ are merged in phase } \ell'\}$$

Proof. First of all note that if q, q' are merged then also, by easy induction on j, pairs of states $\delta_{M'}(q, 1^j), \delta_{M'}(q', 1^j)$ for $j = 0, \ldots s-1$ were merged (or are identical) — if two states are merged then their corresponding successors were merged previously (or are identical). Hence $(\delta_{M'}(q, 1^j 0); \delta_{M'}(q', 1^j 0))$ for $j = 0, \ldots s-1$ were merged (or are identical).

We proceed with the proof by induction on ℓ. Suppose that two states $q, q' \in Q(M)$ are merged in phase ℓ. If $\ell = 0$ then there is nothing to prove. If $\ell > 0$ then consider pairs of states $\big(\delta_{M'}(q, 1^j 0); \delta_{M'}(q', 1^j 0)\big)$ for $j = 0, \ldots, s-1$ (those correspond to states $\delta_M(q, a), \delta_M(q', a)$ for $a \in \Sigma$). Suppose first that one such pair $\big(\delta_{M'}(q, 1^j 0); \delta_{M'}(q', 1^j 0)\big)$ was not merged in phase $\ell - 1$. But then $\big(\delta_{M'}(q, 1^j); \delta_{M'}(q', 1^j)\big)$ could not be merged in phase $\ell - 1$. And so in phase ℓ also q, q' could no be merged.

Suppose now that all the pairs $\big(\delta_{M'}(q, 1^j 0); \delta_{M'}(q', 1^j 0)\big)$ were merged in phase $\ell - 1$ (or earlier). Then also in the same phase the pair of states $\big(\delta_{M'}(q, 1^{s-1}); \delta_{M'}(q', 1^{s-1})\big)$ was merged. Then by easy induction we can see that all the pairs of states of the form $\big(\delta_{M'}(q, 1^j); \delta_{M'}(q', 1^j)\big)$ for $j = 0, \ldots, s-2$ were merged in this phase — each such pair has two pairs of successors: $\big(\delta_{M'}(q, 1^j 1); \delta_{M'}(q', 1^j 1)\big)$ and $\big(\delta_{M'}(q, 1^j 0); \delta_{M'}(q', 1^j 0)\big)$ with the first being merged by (inner) induction hypothesis and the other by (outer) induction hypothesis. So consider all pairs of successors of (q, q') at the beginning of phase ℓ: $\big(\delta_{M'}(q, 0); \delta_{M'}(q', 0)\big)$, $\big(\delta_{M'}(q, 1); \delta_{M'}(q', 1)\big)$ and $\big(\delta_{M'}(q, 2); \delta_{M'}(q', 2)\big)$. Two of were merged in phase $\ell - 1$ and the third consists of the same state. Thus (q, q') is merged. □

By this claim the lemma clearly follows, as for $q, q' \in Q(M)$ if $d(q,q') < \infty$ then $d(q,q') = 0$ means that $q = q'$, as the automaton is minimised and otherwise $d(q,q') = 1 + \max_{a \in \Sigma} d\big(\delta_M(q,a), \delta_M(q,a)\big)$, i.e. it coincides with the definition of being merged in phase ℓ. $\square$

Theorem 3. COMPUTE-EQUIVALENT-k *correctly k-minimises the given automaton M.*

Proof. We show that when COMPUTE-EQUIVALENT-k stops working, stars define relation $\approx_k$. Hence by Theorem 2 COMPUTE-EQUIVALENT-k correctly computes the k-minimal automaton.

By invariant 1 the states in one star are in $\sim_k$. On the other hand if q, q' are stars representatives of different stars then $q \not\sim_k q'$ — either this follows from invariant 5, if q, q' are in the same group, and if they are in different groups, then their distance is ∞ by Lemma 1 and Lemma 11. Hence stars define a proper $\approx_k$ relation. $\square$

Lemma 12. *Invariants 1–6 are kept by* COMPUTE-EQUIVALENT-k*.*

Proof. Invariant 1 follows from algorithm — if two stars are merged then they are assigned a longer representative.

Invariant 2 follows from Lemma 6: suppose that two stars with representatives q and q' are merged and $|\operatorname{word}[q]| \geq |\operatorname{word}[q']|$. In particular $q \sim_k q'$ Consider any q'' in the star of q'. Then $|\operatorname{word}[q'']| \leq |\operatorname{word}[q']|$ and $q'' \sim_k q'$ by invariants 1 and 2. Then by Lemma 6(3) it holds that $q'' \sim_k q'$. Consider also any q''' from the star of q. Then $|\operatorname{word}[q''']| \leq |\operatorname{word}[q]|$ and $q''' \sim_k q$. Thus by Lemma 6(3) it holds that $q'' \sim_k q'''$.

Invariant 3 follows directly from the algorithm.

Before showing invariant 4 we prove a technical claim:

Claim. If a group consists of stars represented by $p_1, \ldots p_s$, $s > 1$ in ℓ-th phase then for each i it holds that $|\operatorname{word}[p_i]| + \ell > k$

Proof. Consider any two stars represented by p_i, $pj2$. Then by invariant 5

$$\min(|\operatorname{word}[p_i]|, |\operatorname{word}[p_j]|) + d(\operatorname{word}[p_i], \operatorname{word}[p_j]) > k$$

and by invariant 6

$$d(\operatorname{word}[p_i], \operatorname{word}[p_j]) \leq \ell$$

thus

$$\min(|\operatorname{word}[p_i]|, |\operatorname{word}[p_j]|) + \ell > k$$

$\square$

Invariant 4. The only non-trivial case is when after merging two groups, represented by q_1 and q_2, stars of q_1 and q_2 were merged. If both groups consisted of only one star then there is nothing to prove. If both groups consisted of several stars then by the Claim $|\operatorname{word}[q_1]| + \ell > k$ and $|\operatorname{word}[q_2]| + \ell > k$, and as $d(q_1, q_2) = \ell$ by invariant 6

$$\min(|\operatorname{word}[q_1]|, |\operatorname{word}[q_2]|) + d(q_1, q_2) > k$$

hence stars of q_1 and q_2 were not merged. So suppose that group of q_1 consists of unique star and group of q_2 has more stars. As

$$\min(|\operatorname{word}[q_1]|, |\operatorname{word}[q_2]|) + d(q_1, q_2) \leq k \text{ and } |\operatorname{word}[q_2]| + d(q_1, q_2) > k$$

then $|\operatorname{word}[q_1]| < |\operatorname{word}[q_2]|$. So q_2 is assigned as the representative of the new star. Other representatives of stars in this group did not change thus it should remain the group representative.

Invariant 5: suppose that two groups represented by p_1 and p'_1 and consisting of stars represented by $p_1, \ldots, p_i$ and $p'_1, \ldots, p'_{i'}$ are merged. Suppose first that stars of p_1 and p_2 were merged. As shown when proving invariant 4 this means that the new group has exactly the same set of star representatives as on the old groups, so the claim is shown.

So assume that the stars represented by p_1 and p'_1 were not merged. We show that for each p_j in the star of p_1 it holds that $|\operatorname{word}[p_j]| + \ell > k$. The same holds for the other star. If the star consists of one vertex then $p_1 = p_j$ and as the stars were not merged:

$$\min(|\operatorname{word}[p_1]|, |\operatorname{word}[p'_1]|) + \ell > k \ .$$

Otherwise by Claim we obtain that $|\operatorname{word}[p_j]| + \ell > k$. So for all p_j in the star of p_1 and $p'_{j'}$ in the star of p'_1:

$$\min(|\operatorname{word}[p_j]|, |\operatorname{word}[p'_{j'}]|) + \ell > k \ .$$

so $p_j \not\sim_k p'_{j'}$. Hence the invariant 5 is kept.

Invariant 6 follows from Lemma 11 — two groups from phase $\ell - 1$ are two equivalence classes of $D_{\ell-1}$. If they are further merged, then their representatives are at distance ℓ, thus those two groups are in one D_ℓ class. Moreover, all representatives at distance ℓ are merged, hence whole D_ℓ classes are obtained.

□

```
COMPUTE-f-EQUIVALENT(M)
  calculate P(M) and K(M)
  remove P_M ∩ {q : L(q) is finite }, obtaining M_1
  M_2 ← minimise(M_1)
  temporarily remove K_M ∩ {q : L(q) is finite }, obtaining M_3
  M′ ← ADD-GADGETS(M_3)
  initialize the dictionary
  for q ∈ Q(M′)
      do insert q into the dictionary
         create l[q] — a list {q′ ∈ Q : ∃x ∈ {0, 1, 2, 3} such that δ(q′, a) = q}
         calculate rank(q)
  while there are two states q, q′ assigned to the same tuple in the dictionary
      do MERGE-STATES(q, q′)
  bring back the states from K_M ∩ {q : L(q) is finite }
  for q ∈ Q(M)
      do q′ ← group[q]
         S ← {q}
         while group[q′] ≠ q′
             do S ← S ∪ {q′}
                q′ ← group[q]
         for q″ ∈ S
             do group[q″] ← q′
  for q ∈ P(M_2)
      do merge q to group[q]
```

```
MERGE-STATES(q, q′)
  if rank(q) > rank(q′)
    then swap q and q′
  delete all elements from l[q] from the dictionary and reinsert them, with q replaced by q′
  rank(q′) ← rank(q′) + rank(q)
  l[q′] ← l[q] concatenated with l[q′]
  group[q] ← q′
```

```
ADJUST-STARS-k(q, q′)
  MERGE-STATES-k(q,q')
  if ℓ + min(|word[q]|, |word[q′]|) ≤ k
    then if |word[q]| > |word[q′]|
            then swap q and q′
         star[q′] ← star[q]
```

```
Compute-equivalent-k(M)
 M ← minimise(M)
 for q ∈ Q(M)
       do calculate word[q] and set star[q] ← q
 M′ ← Add-Gadgets(M)
 for q ∈ Q(M′)
       do calculate word[q] and set group[q] ← q
 initialize the dictionary
 dist ← 0
 for q ∈ Q(M′)
       do insert q into the dictionary
          create l[q] — list {q′ ∈ Q : δ(q′, 0) = q or δ(q′, 1) = q or δ(q′, 1) = q}
          calculate rank(q)
 repeat
          while there are q, q′ ∈ Q(M′) \ Q(M) assigned to the same tuple in the dictionary
                do Merge-states(q,q')
          dist ← dist +1
          L ← list of non-trivial groups of states assigned to the same tuple in the dictionary
          for {q_1, ..., q_j} ∈ L
                do q ← q_1
                   for x ← 2 to j
                         do Merge-states(q_1, q)
                            Adjust-stars(q_1, q)
    until there are no two states assigned to the same tuple in the dictionary
 for q ∈ Q(M)
       do q′ ← group[q]
          S ← {q}
          while group[q′] ≠ q′
                do S ← S ∪ {q′}
                   q′ ← group[q]
          for q′′ ∈ S
                do group[q′′] ← q′
 for q ∈ Q(M)
       do q′ ← star[q]
          S ← {q}
          while star[q′] ≠ q′
                do S ← S ∪ {q′}
                   q′ ← star[q]
          for q′′ ∈ S
                do star[q′′] ← q′
 N ← Equivalence-To-Automata(M, star)
```