Acta Informatica (2021) 58:177–194
https://doi.org/10.1007/s00236-019-00363-5

ORIGINAL ARTICLE

Aggregation-based minimization of finite state automata

Johanna Björklund[1] · **Loek Cleophas**[2]

Received: 18 December 2018 / Accepted: 11 December 2019 / Published online: 6 January 2020

Abstract
We present a minimization algorithm for non-deterministic finite state automata that finds and merges bisimulation-equivalent states. The bisimulation relation is computed through partition aggregation, in contrast to existing algorithms that use partition refinement. The algorithm simultaneously generalises and simplifies an earlier one by Watson and Daciuk for deterministic devices. We show the algorithm to be correct and run in time $O(n^2 r^2 \,|\Sigma|)$, where n is the number of states of the input automaton M, r is the maximal out-degree in the transition graph for any combination of state and input symbol, and $|\Sigma|$ is the size of the input alphabet. The algorithm has a higher time complexity than derivatives of Hopcroft's partition-refinement algorithm, but represents a promising new solution approach that preserves language equivalence throughout the computation process. Furthermore, since the algorithm essentially computes the maximal model of a logical formula derived from M, optimisation techniques from the field of model checking become applicable.

1 Introduction

Finite-state automata (NFA) is a fundamental concept in theoretical computer science, and their computational and representational complexity is the subject of extensive investigations. In this work, we revisit the minimization problem for NFA, which inputs an automaton M with n states and outputs a minimal language equivalent automaton M'. In the case of *deterministic* finite state automata (DFA), it is well-known that M' is always unique and canonical with respect to the recognized language. In the more general, non-deterministic case, no analogous result exists and M' is typically only one of several equally compact automata. Moreover, finding any one of these is PSPACE complete [17], and the problem cannot even be efficiently approximated within a factor $o(n)$ unless P = PSPACE [12].

Since NFA minimization is inherently difficult, attention has turned to efficient heuristic minimization algorithms, that often, if not always, perform well. In this category we find bisimulation minimization. Intuitively, two states are bisimulation equivalent if every tran-

✉ Johanna Björklund
johanna@cs.umu.se

Loek Cleophas
loek@fastar.org

[1] Department of Computing Science, Umeå University, 901 87 Umeå, Sweden

[2] Department of Information Science, Stellenbosch University, Stellenbosch, South Africa

sition that can be made from one of them, can be mirrored starting from the other. More formally, an equivalence relation $\mathcal{E}$ on the states Q of an NFA M is a bisimulation relation if the following holds: (i) the relations respects the separation in M of final and non-final states, and (ii) for every $p, q \in Q$ such that $(p, q) \in \mathcal{E}$, if $p' \in Q$ can be reached from p on the symbol a, then there must be a $q' \in Q$ that can be reached from q on a, and $(p', q') \in \mathcal{E}$.

The transitive closure of the union of two bisimulation relations is again a bisimulation relation, so there is a unique coarsest bisimulation relation $\mathcal{E}$ of every NFA M. When each equivalence class of $\mathcal{E}$ is merged into a single state, the result is a smaller but language-equivalent NFA. If M is deterministic, then this approach coincides with regular DFA minimization. The currently predominant method of finding $\mathcal{E}$ is through partition refinement: The states are initially divided into final and non-final states, and the minimization algorithm resolves contradictions to the bisimulation condition by refining the partition until a fixed point is reached. This method is fast, and requires $O(m \log n)$ computation steps (see [20]), where m is the size of M's transition function. The drawback is that up until termination, merging equivalence classes into states will not preserve the recognized language.

In this paper, which extends and revises [5], we present an NFA minimization algorithm that produces intermediate solutions language-equivalent to M. Similarly to previous approaches, the algorithm computes the coarsest bisimulation relation $\mathcal{E}$ on M. However, the initial partition is entirely made up of singleton classes, and these are repeatedly merged until a fixed point is reached. The algorithm runs in time $O\big(n^2 \cdot (\log n^2 + r^2\,|\Sigma|)\big)$, where r is the maximal outdegree in the transition graph for any combination of state and input symbol, and Σ is the input alphabet. This is slower than the derivatives of Hopcroft's partition-refinement algorithm, out of which Paige and Tarjan's algorithm is one, but we believe that it is a useful first step, and it is still an open question whether partition aggregation can be computed as efficiently as partition refinement.

The use of aggregation was inspired by a family of minimization algorithms for DFAs (see Sect. 1.1), and we lift the technique to non-deterministic devices. In the deterministic case, our algorithm runs in $O\big(n^2\,|\Sigma|\big)$, which is the same as for the fastest aggregation-based DFA minimisation algorithms.

Another contribution is the computational approach: we derive a characteristic propositional-logic formula w_M for the input automaton M, in which the variables are pairs of states. The algorithm's main task is to compute a maximal model $\hat{v}$ of w_M, in the sense that $\hat{v}$ assigns 'true' to as many variables as possible. We show that if w_M is satisfiable, then $\hat{v}$ is unique and efficiently computable by a greedy algorithm, and $\hat{v}$ encodes the coarsest bisimulation relation on M.

1.1 Related work

DFA minimization has been studied extensively since the 1950s (see [13,15,18]). ten Eikelder [22] observed that the equivalence problem for recursive types can be formulated as a DFA reachability problem, and gave a recursive procedure for deciding equivalence for a pair of DFA states. This procedure was later used by Watson [23] to formulate a DFA minimization algorithm that works through partition aggregation. The algorithm runs in exponential time, and two mutually exclusive optimization methods were proposed by Watson and Daciuk [24]. One uses memoization to limit the number of recursive invocations; the other bases the implementation on the union-find data structure (see [2,14,21]). The union-find method reduces the complexity from $O\big(|\Sigma|^{n-2}n^2\big)$ down to $O\big(\alpha(n^2)n^2\big)$, where $\alpha(n)$, roughly speaking, is

the inverse of Ackermann's function. The value of this function is less than 5 for $n \leq 2^{2^{16}}$, so it can be treated as a constant.

The original formulation of the algorithm was later rectified by Daciuk [10], who discovered and removed an incorrect combination of memoization and restricted recursion depth. The fact that this combination was problematic had been pointed out by Almeida et al. [3], who had found situations in which the Watson–Daciuk algorithm returned non-minimal DFAs. Almeida et al. [3] also presented a simpler version, doing away with presumably costly dependency list management. Assuming a constant alphabet size, they state that their algorithm has a worst-case running time of $O(\alpha(n^2)n^2)$ for all practical cases, yet also claim it to be faster than the Watson–Daciuk one. Based on Almeida's reporting, Daciuk [10, Section 7.4] provided a new version, presented as a compromise between the corrected Watson-Daciuk and the Almeida-Moreira-Reis algorithm, but did not discuss its efficiency. The original version of the algorithm has been lifted to deterministic tree automata (a generalisation of finite state automata) both as an imperative sequential algorithm and in terms of communicating sequential processes (see [9]).

NFA minimisation has also received much attention, and we restrict our discussion to heuristics that compute weaker relations than the actual Nerode congruence (recalled in Sect. 2). Paige and Tarjan [20] presented three partition refinement algorithms, one of which is essentially bisimulation minimization for NFAs. The technique was revived by Abdulla et al. [1] for finite-state tree automata. The paper was soon followed by bisimulation-minimization algorithms for weighted and unranked tree automata by Björklund et al. [6] and Björklund et al. [7], and also algorithms based on more general simulation relations by Abdulla et al. [1] and Maletti [16]. Our work is to the best of our knowledge the first in which the bisimulation relation is computed through partition aggregation.

2 Preliminaries

2.1 Sets, numbers, and relations

We write $\mathbb{N}$ for the set of natural numbers, including 0. For $n \in \mathbb{N}$, $[n] = \{i \in \mathbb{N} \mid 1 \leq i \leq n\}$. Thus, $[0] = \emptyset$. The cardinality of a set S is written $|S|$ and the powerset of S by $pow(S)$. A binary relation $\otimes\colon S \times S \to S$ is *idempotent* if $s \otimes s = s$, for every $s \in S$.

A binary relation is an equivalence relation if it is reflexive, symmetric and transitive. Let $\mathcal{E}$ and $\mathcal{F}$ be equivalence relations on S. We say that $\mathcal{F}$ is *coarser* than $\mathcal{E}$ (or equivalently: that $\mathcal{E}$ is a *refinement* of $\mathcal{F}$), if $\mathcal{E} \subseteq \mathcal{F}$. The *equivalence class* or *block* of an element s in S with respect to $\mathcal{E}$ is the set $[s]_{\mathcal{E}} = \{s' \mid (s, s') \in \mathcal{E}\}$. Whenever $\mathcal{E}$ is obvious from the context, we simply write $[s]$ instead of $[s]_{\mathcal{E}}$. It should be clear that $[s]$ and $[s']$ are equal if s and s' are in relation $\mathcal{E}$, and disjoint otherwise, so $\mathcal{E}$ induces a partition $(S/\mathcal{E}) = \{[s] \mid s \in S\}$ of S. The *identity relation* on S is $\mathcal{I}_S = \{(s, s) \mid s \in S\}$.

An alphabet is a finite nonempty set. Given an alphabet Σ we write Σ^* for the set of all strings over Σ, and ε for the empty string. A string language is a subset of Σ^*.

2.2 Finite state automata

A *nondeterministic finite state automaton* is a tuple $M = (Q, \Sigma, \delta, Q_I, Q_F)$, where Q is a finite set of *states*; Σ is an alphabet of *input symbols*; the *transition function* $\delta = (\delta_f)_{f \in \Sigma}$

is a family of functions $\delta_f : Q \to pow(Q)$; $Q_I \subseteq Q$ is a set of *initial states*; and $Q_F \subseteq Q$ is a set of *final states*.

We immediately extend δ to $(\hat{\delta}_w)_{w\in\Sigma^*}$ where $\hat{\delta}_w : pow(Q) \to pow(Q)$ as follows: For every string $w \in \Sigma^*$ and set of states $P \subseteq Q$,

$$\hat{\delta}_w(P) = \begin{cases} P & \text{if } w = \varepsilon, \quad \text{and} \\ \bigcup_{p\in P} \hat{\delta}_{w'}(\delta_f(p)) & \text{if } w = fw' \text{ for some } f \in \Sigma, \quad \text{and } w' \in \Sigma^*. \end{cases}$$

The language recognised by M is $\mathcal{L}(M) = \{w \in \Sigma^* \mid \hat{\delta}_w(Q_I) \cap Q_F \neq \emptyset\}$. A state $q \in Q$ is *useless* if there do not exist strings $u, w \in \Sigma^*$ such that $q \in \hat{\delta}_u(Q_I)$ and $Q_F \cap \hat{\delta}_w(\{q\}) \neq \emptyset$. From here on, we identify δ with $\hat{\delta}$. If $|Q_I| \leq 1$, and if $\left|\delta_f(\{q\})\right| \leq 1$ for every $f \in \Sigma$ and $q \in Q$, then M is said to be *deterministic*.

Let $\mathcal{E}$ be an equivalence relation on Q. The *aggregated* NFA with respect to $\mathcal{E}$ is the NFA $(M/\mathcal{E}) = ((Q/\mathcal{E}), \Sigma, \delta', Q'_I, Q'_F)$ given by $\delta'_f([q]) = \{[p] | p \in \delta_f(q)\}$ for every $q \in Q$ and $f \in \Sigma$; $Q'_I = \{[q] \mid q \in Q_I\}$; and $Q'_F = \{[q] | q \in Q_F\}$.

The *right language* of $q \in Q$ is $\overrightarrow{\mathcal{L}}(q) = \{w \in \Sigma^* \mid \delta_w(\{q\}) \cap Q_F \neq \emptyset\}$. The Nerode congruence (see [19]) is the coarsest *congruence relation* $\mathcal{E}$ on Q with respect to the right-languages of the states in Q. This means that $(p, q) \in \mathcal{E}$ if and only if $\overrightarrow{\mathcal{L}}(p) = \overrightarrow{\mathcal{L}}(q)$ for all $p, q \in Q$.

2.3 Propositional logic

We assume that the reader is familiar with propositional logic, but recall some basic facts to fix terminology. It is important to note, that in the definitions that follow, interpretations are in general *partial* functions.

The Boolean values *true* and *false* are written as $\top$ and $\bot$, respectively, and we use $\mathbb{B}$ for $\{\top, \bot\}$. Let L be a propositional logic over the logical variables X, and let $\mathrm{WF}(L)$ be the set of well-formed formulas over L. An *interpretation* of L is a partial function $X \to \mathbb{B}$. Given interpretations v and v', we say that v' is an *extension* of v if $v'(x) = v(x)$ for all $x \in dom(v)$. The set of all such extensions is written $\mathrm{Ext}(v)$.

As usual, the semantics of a well-formed formula $w \in \mathrm{WF}(L)$ is a function from the set of all total interpretations (i.e., from all total mappings $X \to \mathbb{B}$), to $\mathbb{B}$. A total interpretation v is a *total model* for w if $w(v) = \top$ (by convention, hereafter this application as $v(w)$). The set of all total models for w is written $\mathrm{Mod}^{\mathrm{t}}(w)$. Given a pair of formulas $w, w' \in \mathbb{B}$, we write $w \equiv w'$ to denote that $\mathrm{Mod}^{\mathrm{t}}(w) = \mathrm{Mod}^{\mathrm{t}}(w')$.

A *substitution* of formulas for a finite set of variables X is a set $\{x_1 \leftarrow w_1, \ldots, x_n \leftarrow w_n\}$, where each $x_i \in X$ is a distinct variable and each $w_i \in \mathrm{WF}(L) \setminus X$ is a formula. The *empty* substitution is defined by the empty set. Let $\theta = \{x_1 \leftarrow w_1, \ldots, x_n \leftarrow w_n\}$ and $\sigma = \{y_1 \leftarrow w'_1, \ldots, y_k \leftarrow w'_k\}$ be two substitutions. Let X and Y be the sets of variables substituted for in θ and σ, respectively. The *composition* $\theta\sigma$ of θ and σ is the substitution $\{x_i \leftarrow w_i\sigma \mid x_i \in X\} \cup \{y_j \leftarrow w_j \mid y_j \in Y \setminus X\}$. The *application* of θ to a formula w is denoted $w\theta$ and defined by (simultaneously) replacing every occurrence of each x_i in w by the corresponding w_i. Finally, given a set of formulas $W \subseteq \mathrm{WF}(L)$, we let $W\theta = \{w\theta \mid w \in W\}$.

Every partial interpretation v of L can be seen as a substitution, in which $x \in dom(v)$ is replaced by $v(x)$, resulting in a new formula wv in $\mathrm{WF}(L)$ with variables in $X \setminus dom(v)$. This allows us to extend v to a function $\mathrm{WF}(L) \to ((X \to \mathbb{B}) \to \mathbb{B})$ defined by $v(w) = wv$.

Example 1 Consider the formulas $w = x_1 \to x_2$ and $w' = x_1 \wedge x_2$, and the partial interpretation $v = \{x_1 \leftarrow \bot\}$. Then $w \not\equiv w'$, and $w \not\equiv \top$, but $v(w) = \bot \to x_2 \equiv \top$ and $v(w') \equiv x_2$. ◇

Let v be a partial interpretation. The formula w is *resolved* by v if $v(w) \equiv \top$ or $v(w) \equiv \bot$. The interpretation of v is a *model* for w if $v(w) \equiv \top$, and the set of all models of w is denoted by $\mathrm{Mod}(w)$ (so $\mathrm{Mod}^{\mathrm{t}}(w)$ is a subset of $\mathrm{Mod}(w)$).

Conversely, given a substitution σ we can define a partial interpretation $\overline{\sigma} : X \to \mathbb{B}$, by $\overline{\sigma}(x) = x\sigma$.

The *join* of a pair of partial interpretations v and v' is the total interpretation $v \vee v' \colon X \to \mathbb{B}$ given by $(v \vee v')(x) = \top$ if $v(x) \equiv \top$ or $v'(x) \equiv \top$, and by $(v \vee v')(x) = \bot$ otherwise.

A formula in $\mathrm{WF}(L)$ is in *conjunctive normal form* (CNF) if it is a conjunction of clauses, where each clause is a disjunction of possibly negated variables. A formula is *negation-free* if no variable occurs negated.

3 Logical framework

In this section, we express the problem of finding the coarsest simulation relation on a finite automaton, as a problem of computing the maximal model of a propositional-logic formula.

From here on, $M = (Q, \Sigma, \delta, Q_I, Q_F)$ is a fixed but arbitrary NFA, free from useless states.

Definition 1 (*Bisimulation, cf.* [8], *Definition 3.1*) Let $\mathcal{E}$ be a relation on Q. It is a *bisimulation relation on* M if for every $(p, q) \in \mathcal{E}$,

1. $p \in Q_F$ if and only if $q \in Q_F$; and
2. for every symbol $f \in \Sigma$,

$$\begin{aligned} &\text{for every } p' \in \delta_f(p) \text{ there is a } q' \in \delta_f(q) \text{ such that } (p', q') \in \mathcal{E}, \text{ and} \\ &\text{for every } q' \in \delta_f(q) \text{ there is a } p' \in \delta_f(p) \text{ such that } (p', q') \in \mathcal{E}. \end{aligned}$$

We shall express the second of these conditions in a propositional logic, in which the variables are pairs of states. The resulting formula is such that if the variable $\langle p, q\rangle$ is assigned the value $\top$, then p and q must satisfy Condition 2 of Definition 1 for the whole formula to be true.

In the following, we take the conjunction of an empty set of Boolean values to be true (or $\top$), and the disjuction of an empty set of Boolean values to be false (or $\bot$).

Definition 2 (*Characteristic formula*) Let $X_M = \{\langle p, q\rangle \mid p, q \in Q\}$ be a set of propositional variables. For $x = \langle p, q\rangle \in X_M$ and $f \in \Sigma$, we denote by w_x^f the CNF formula

$$\bigwedge_{p' \in \delta_f(p)} \bigvee_{q' \in \delta_f(q)} \langle p', q'\rangle \quad \wedge \quad \bigwedge_{q' \in \delta_f(q)} \bigvee_{p' \in \delta_f(p)} \langle p', q'\rangle,$$

and by w_x the formula $\bigwedge_{f \in \Sigma} w_x^f$. It should be clear that, for every $f \in \Sigma$ and $x \in X_M$, the formulas w_x^f and w_x are negation-free. Finally, w_M denotes the conjunction $\bigwedge_{x \in X_M} (x \to w_x)$, and w_x is said to be the *right-hand side* of the implication $x \to w_x$.

We could also model Condition 1 of Definition 1 in the formula w_M, but that would introduce negations and make the presentation more involved. To find the coarsest bisimulation

relation for M, we start instead with a partial interpretation of X_M satisfying Condition 1 of Definition 1 and search for a 'maximal' total extension that also satisfies Condition 2. By 'maximal' we mean that it assigns as many variables as possible the value $\top$.

Definition 3 (*Maximal model*) Let v and v' be interpretations of X_M. We say that the total model $v \in \mathrm{Mod}^{\mathrm{t}}(w_M)$ is *maximal* if $v \vee v' = v$ for every $v' \in \mathrm{Ext}(v) \cap \mathrm{Mod}(w_M)$.

Due to the structure of w_M, its models are closed under the join operator.

Lemma 1 *If $v, v' \in Mod(w_M)$, then $v \vee v' \in Mod(w_M)$.*

Proof The interpretation $v \vee v'$ fails to satisfy w_M if there is some $x \in X_M$ such that $(v \vee v')(x \to w_x)$ is false. This can only happen if $(v \vee v')(x) = \top$ but $(v \vee v')(w_x) \equiv \bot$. However, if $(v \vee v')(x) = \top$ then $v(x) = \top$ or $v'(x) = \top$. Assume the former, without loss of generality. Then $v(w_x) \equiv \top$ since $v \in \mathrm{Mod}(w_M)$. Now, the fact that more variables are assigned the value $\top$ in $v \vee v'$ cannot cause w_x to become false, since it is negation-free. Hence $(v \vee v')(w_x) \equiv \top$ too, which gives us a contradiction. It follows that $v \vee v' \in \mathrm{Mod}^{\mathrm{t}}(w_M)$, and since $\mathrm{Mod}^{\mathrm{t}}(w_M) \subseteq \mathrm{Mod}(w_M)$, that $v \vee v' \in \mathrm{Mod}(w_M)$. □

From Lemma 1, we conclude that when a solution exists, it is unique.

Lemma 2 *Let v be a partial interpretation of X_M. If $Ext(v) \cap Mod(w_M) \neq \emptyset$, then there is a total interpretation $\hat{v} \in Ext(v)$ that is a maximal model of w_M, and $\hat{v}$ is unique.*

Proof If v cannot be extended to a model of w_M then the statement is trivially true. If it can be extended to a model, then by Lemma 1 the join of all such extensions is a model of w_M, and it is unique since join is idempotent. □

Given $v \in \mathrm{Mod}(w_M)$, Lemma 2 allows us to unambiguously write $\mathrm{Max}(M, v)$ for the unique maximal model of w_M in $\mathrm{Mod}^{\mathrm{t}}(w_M) \cap \mathrm{Ext}(v)$.

To translate our logical models back into the domain of bisimulation relations, we introduce the notion of their *associated relations*.

Definition 4 (*Associated relation*) We associate with every (partial) interpretation v of X_M a relation $\sim_v$ on X_M, given by $p \sim_v q \iff v(\langle p, q \rangle) = \top$. We say that the interpretation v is reflexive, symmetric, and transitive, respectively, whenever $\sim_v$ is.

Note that Definition 4 does not distinguish between a state pair x for which $v(x) = \bot$, and a state pair for which v is undefined. If v is an arbitrary model of w_M, then its associated relation need not be an equivalence relation, but for the maximal model, it is.

Lemma 3 *Let v be a partial interpretation of X_M such that $\sim_v$ is an equivalence relation, then also $\sim_{\hat{v}}$, where $\hat{v} = Max(M, v)$, is an equivalence relation.*

Proof Since $\sim_v$ is reflexive, $v(\langle p, p \rangle) = \top$ for every $p \in X$, so the associated relation of every extention of v is also reflexive.

Since the logical operators $\vee$ and $\wedge$ commute, every extension v' of v in which $v'(\langle p, q \rangle) = \top$ can be turned into a model v'' in which $v''(\langle q, p \rangle) = \top$ by swapping the order of every pair in X_M. By taking the join of v' and v'', we arrive at a greater model $v' \vee v''$ in which $(v' \vee v'')(\langle q, p \rangle) = (v' \vee v'')(\langle p, q \rangle) = \top$. Since $\hat{v}$ is the maximal model of v, it is necessarily already symmetric.

A similar argument holds for transitivity. Let v' be the transitive closure of $\hat{v}$, in other words, let v' be the complete interpretation that assigns the fewest number of variables in X_M the value $\top$, while still guaranteeing that for all $p, q, r \in X_M$, (i) $\hat{v}(\langle p, q\rangle) = \top$ implies $v'(\langle p, q\rangle) = \top$, and (ii) $v'(\langle p, q\rangle) = v'(\langle q, r\rangle) = \top$ implies that $v'(\langle p, r\rangle) = \top$.

We verify that v' is also a model for w_M, by checking that $v'(x \rightarrow w_x) \equiv \top$ for every $x \in X_M$. Assume that $\langle p, q\rangle \in X_M$ and $v'(\langle p, q\rangle) = \top$. Then, there is a sequence $P = \langle p_1, p_2\rangle, \langle p_2, p_3\rangle, \ldots, \langle p_{n-1}, p_n\rangle$, for some $n \in \mathbb{N}$, such that $p = p_1$, $q = p_n$, $\hat{v}(\langle p_i, p_{i+1}\rangle) = \top$ for every $i \in [n]$. Since $\hat{v}$ is a model for w_M, it must hold that $\hat{v}(w_{\langle p_i, p_{i+1}\rangle}) \equiv \top$ for every $i \in [n]$, so $v'(w_{\langle p_i, p_{i+1}\rangle}) \equiv \top$, for every $i \in [n]$ since v' assigns more variables the value $\top$ than $\hat{v}$ does, and since $w_{\langle p_i, p_{i+1}\rangle}$ is negation-free. Suppose for the sake of contradiction that $P' = \langle p_1, p_2\rangle, \ldots, \langle p_{k-1}, p_k\rangle$ is a prefix of P such that $v'(w_{\langle p_1, p_k\rangle}) \equiv \top$ but $v'(w_{\langle p_1, p_{k+1}\rangle}) \not\equiv \top$, and that P is the shortest such prefix. We know that

$$v'\left(\bigwedge_{p'_1 \in \delta_f(p_1)} \bigvee_{p'_k \in \delta_f(p_k)} \langle p'_1, p'_k\rangle \wedge \bigwedge_{p'_k \in \delta_f(p_k)} \bigvee_{p'_1 \in \delta_f(p_1)} \langle p'_1, p'_k\rangle\right) \equiv \top.$$

and

$$v'\left(\bigwedge_{p'_k \in \delta_f(p_k)} \bigvee_{p'_{k+1} \in \delta_f(p_{k+1})} \langle p'_k, p'_{k+1}\rangle \wedge \bigwedge_{p'_{k+1} \in \delta_f(p_{k+1})} \bigvee_{p'_k \in \delta_f(p_k)} \langle p'_k, p'_{k+1}\rangle\right) \equiv \top.$$

This means that for every $p'_1 \in \delta_f(p_1)$ there is some $p'_k \in \delta_f(p_k)$ such that $v'(\langle p'_1, p'_k\rangle) = \top$, and that for this p'_k there is some $p'_{k+1} \in \delta(p_{k+1})$ such that $v'(\langle p'_k, p'_{k+1}\rangle) = \top$. Since v' is transitive, also $v'(\langle p'_1, p'_{k+1}\rangle) = \top$. It follows that

$$v'\left(\bigwedge_{p'_1 \in \delta_f(p_1)} \bigvee_{p'_{k+1} \in \delta_f(p_{k+1})} \langle p'_1, p'_{k+1}\rangle \wedge \bigwedge_{p'_{k+1} \in \delta_f(p_{k+1})} \bigvee_{p'_1 \in \delta_f(p_1)} \langle p'_1, p'_{k+1}\rangle\right) \equiv \top,$$

so $v'(w_{\langle p_1, p_{k+1}\rangle}) \equiv \top$, that is, a contradiction. Since $\hat{v}$ is already maximal, it has to be transitive. □

We introduce a partial interpretation v_0 to reflect Condition 1 of Definition 1 and use this as the starting point for our search.

Definition 5 Let v_0 be the partial interpretation of X_M such that

$$v_0(\langle p, p\rangle) = \top \text{ for every } p \in Q,$$
$$v_0(\langle p, q\rangle) = \bot \text{ for every } p, q \in Q \text{ with } p \in Q_F \not\equiv q \in Q_F,$$

and v_0 is undefined on all other state pairs.

Lemma 4 *The interpretation v_0 is in Mod(w_M) and $\sim_{v_0}$ is an equivalence relation.*

Proof To verify that v_0 is a model for w_M, we must ensure that $v_0(\langle p, p\rangle \rightarrow w_{\langle p,p\rangle}) \equiv \top$ for every $p \in Q$. By definition, $w_{\langle p,p\rangle} =$

$$\bigwedge_{p' \in \delta_f(p)} \bigvee_{p'' \in \delta_f(p)} \langle p', p''\rangle \quad \wedge \quad \bigwedge_{p' \in \delta_f(p)} \bigvee_{p'' \in \delta_f(p)} \langle p', p''\rangle.$$

This means that for every $p' \in \delta_f(p)$ we know that there is some $p'' \in \delta_f(p)$ (namely p' itself), such that $v_0(\langle p', p'' \rangle) = \top$, so $v_0(w_{\langle p,p \rangle}) \equiv \top$.

For the second part of the statement, we note that $\sim_{v_0} = \mathcal{I}_{X_M}$. Furthermore, $\mathcal{I}_{X_M}$ is clearly an equivalence relation, namely the finest one in which each state is an equivalence class of its own. □

We summarize this section's main findings in Theorem 1.

Theorem 1 *There is a unique maximal extension $\hat{v} = Max(M, v_0)$ of v_0 in $Mod(w_M)$, and the relation $\sim_{\hat{v}}$ is the coarsest bisimulation relation on M.*

Proof From Lemma 4 it follows that v_0 is a model of w_M, and that it encodes an equivalence relation. From Lemma 2 that v_0 can be extended to a unique maximal model $\hat{v}$ for w_M. From Definitions 1 and 2 that $\hat{v}$ encodes a bisimulation relation, from Lemma 3 that $\hat{v}$ is an equivalence relation. □

4 Algorithm

An aggregation-based minimisation algorithm starts with a singleton partition, in which each state is viewed as a separate block, and iteratively merges blocks found to be equivalent. When all blocks have become mutually distinguishable, the algorithm terminates. We take the same approach for the more general problem of minimizing NFAs with respect to bisimulation equivalence. The procedure is outlined in Algorithm 1 and the auxiliary Algorithm 2.

The input to Algorithm 1 is an NFA $M = (Q, \Sigma, \delta, Q_I, Q_F)$. The algorithm computes the interpretation $\hat{v}$ of the set of variables $X_M = \{\langle p, q \rangle \mid p, q \in Q\}$, where $\hat{v}(x) = \top$ means that x is a pair of equivalent states, and $\hat{v}(x) = \bot$ that x is a pair of distinguishable states. The interpretation $\hat{v}$ is an extension of v_0, in the meaning of Definition 5, and a maximal model for the characteristic formula w_M. Due to the structure of w_M this maximal model can, as we shall see, be computed greedily.

The maximal model $\mathrm{Max}(M, v_0)$ is derived by incrementally assembling a substitution σ, which replaces state pairs by logical formulas. When outlining the algorithm, we add an index to σ to address distinct assignments to σ. The method is such that (i) the substitution is eventually a total function, and (ii) no right-hand side of the substitution contains a variable that is also in the domain of the substitution. In combination, this means that when the algorithm terminates, the logical value of every variable is resolved to $\top$ or $\bot$. The substitution thus comes to represent a total interpretation of X_M. In the computations, σ_i is a global variable. It is initialised such that it substitutes $\top$ for each pair of identical states, and $\bot$ for each pair of states that differ in their finality (see Line 2 of Algorithm 1). Following this initialisation, the function *equiv* (see Algorithm 2) is called for each pair of states not yet resolved by the substitution.

Function *equiv* has two parameters: the pair of states x for which equivalence should be determined, and a set S of pairs of states that are under investigation in previous, though not yet completed, invocations of the function. In other words, S contains pairs that are higher up in the call hierarchy. The function recursively invokes itself with those pairs of states that occur as a variable in formula $w_x\sigma_i$, but which have not yet been resolved, nor form part of the call stack S.

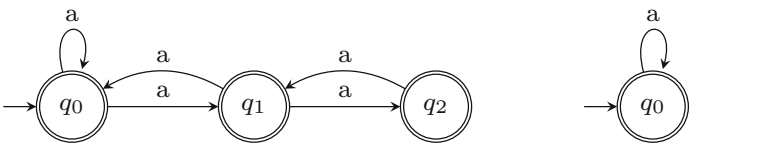
After these calls have been completed and the while loop exited, the following two steps are taken: First, the formula $w_x\sigma_i\{x \leftarrow \top\}$ is derived from $w_x\sigma_i$ by replacing every occurrence of x by $\top$, and second, the substitution σ_{i+1} is derived from σ_i by adding a rule that substitutes x

Algorithm 1 Aggregation-based bisimulation minimization algorithm.

1: **function** *minimize*(M)
2: $\sigma_0 ::= \{\langle q, q\rangle \leftarrow \top \mid q \in Q\} \cup \{\langle p, q\rangle \leftarrow \bot \mid (p \in Q_F) \neq (q \in Q_F)\}$
3: **for** $x \in X_M \setminus dom(\sigma_i)$ **do**
4: $equiv(x, \{x\})$
5: **end for**
6: **return** $(M/\sim_{\sigma_i})$
7: **end function**

Algorithm 2 Point-wise computation of $x \in X_M$

1: **function** $equiv(x, S)$
2: **while** $\exists x' \in var(w_x\sigma_i) \setminus S$ and $w_x\sigma_i$ is not resolved **do**
3: $equiv(x', S \cup \{x'\})$
4: **end while**
5: $\sigma_{i+1} ::= \sigma_i\{x \leftarrow w_x\sigma_i\{x \leftarrow \top\}\}$
6: **end function**



(a) A non-minimal NFA **(b)** A minimal NFA

Fig. 1 Two NFA of different sizes for $\mathcal{L} = \{a^*\}$

by $w_x\sigma_i\{x \leftarrow \top\}$. When combined, these steps clear cyclic dependencies, while guaranteeing that the maximal model for the updated formula remains the same.

Example 2 To illustrate the algorithm and sketch the intuition behind it, we consider the automaton in Fig. 1a. The automaton represents a non-minimal NFA for the language $\mathcal{L} = \{a^*\}$; for comparison, Fig. 1b represents the minimal NFA for the same language.

The non-minimal NFA gives rise to nine pairs of states as variables. For the pair $\langle q_0, q_1\rangle$, for example, the corresponding formula $w_{\langle q_0,q_1\rangle}$ is

$$(\langle q_0, q_0\rangle \vee \langle q_0, q_2\rangle) \wedge (\langle q_1, q_0\rangle \vee \langle q_1, q_2\rangle) \wedge (\langle q_0, q_0\rangle \vee \langle q_1, q_0\rangle) \wedge (\langle q_0, q_2\rangle \vee \langle q_1, q_2\rangle).$$

Line 1 of Algorithm 1 ensures that the three pairs of identical states all resolve to $\top$. In other words,

$$\sigma_0 = \bigcup_{i\in\{0,\ldots,2\}} \{\langle q_i, q_i\rangle \leftarrow \top\}$$

This means that $w_{\langle q_0,q_1\rangle}\sigma_0 = (\langle q_1, q_0\rangle \vee \langle q_1, q_2\rangle) \wedge (\langle q_0, q_2\rangle \vee \langle q_1, q_2\rangle)$. As observed in the proof of Lemma 3, the solution will be symmetric, so we need only consider, without loss of generality, the three pairs $\langle q_0, q_1\rangle$, $\langle q_0, q_2\rangle$ and $\langle q_1, q_2\rangle$ and the corresponding formula for each of these.

Assuming that the 'for' loop in Algorithm 1 initially selects the pair $\langle q_0, q_1\rangle$, a call to $equiv(\langle q_0, q_1\rangle, \{\langle q_0, q_1\rangle\})$ occurs. In the called function *equiv*, the existential quantification on Line 2 will be true, namely for each of the other two of the three pairs indicated above, i.e., for $\langle q_0, q_2\rangle$ and $\langle q_1, q_2\rangle$.

Assuming $\langle q_0, q_2\rangle$ is selected, $equiv(\langle q_0, q_2\rangle, \{\langle q_0, q_1\rangle, \langle q_0, q_2\rangle\})$ is called. We have that $w_{\langle q_0,q_2\rangle} = \langle q_0, q_1\rangle \wedge \langle q_1, q_1\rangle \wedge (\langle q_0, q_1\rangle \vee \langle q_1, q_1\rangle)$, so $w_{\langle q_0,q_2\rangle}\sigma_0 = \langle q_0, q_1\rangle$. Since $\langle q_0, q_1\rangle$

is on the stack, the function returns, and we have

$$\sigma_1 = \bigcup_{i\in\{0,\ldots,2\}} \{\langle q_i, q_i\rangle \leftarrow \top\} \cup \{\langle q_0, q_2\rangle \leftarrow \langle q_0, q_1\rangle\} \ .$$

The function now calls *equiv* with $equiv(\langle q_1, q_2\rangle, \{\langle q_0, q_1\rangle, \langle q_1, q_2\rangle\})$. Since $w_{\langle q_1,q_2\rangle} = (\langle q_0, q_1\rangle \wedge \langle q_1, q_2\rangle)$ we have $w_{\langle q_1,q_2\rangle}\sigma_1 = (\langle q_0, q_1\rangle \wedge \langle q_1, q_2\rangle)$. Now, also $\langle q_1, q_2\rangle$ is on the stack so the function returns and because $(\langle q_0, q_1\rangle \wedge \langle q_2, q_1\rangle)\{\langle q_2, q_1\rangle \leftarrow \top\} = \langle q_0, q_1\rangle$, we have

$$\sigma_2 = \bigcup_{i\in\{0,\ldots,2\}} \{\langle q_i, q_i\rangle \leftarrow \top\} \cup \{\langle q_0, q_2\rangle \leftarrow \langle q_0, q_1\rangle, \langle q_1, q_2\rangle \leftarrow \langle q_0, q_1\rangle)\}.$$

The options on Line 2 of Algorithm 2 have now been exhausted, so the call to *equiv* returns with

$$\sigma_3 = \cup_{i\in\{0,\ldots,2\}}\{\langle q_i, q_i\rangle \leftarrow \top\} \cup \{\langle q_0, q_2\rangle \leftarrow \top, \langle q_1, q_2\rangle \leftarrow \top, \langle q_0, q_1\rangle \leftarrow \top)\}.$$

Thus, all three states have been identified as equivalent and can be merged into a single one, yielding the automaton shown in Fig. 1b. ◇

4.1 Correctness

The correctness proof is based on the fact that throughout the computation, $var(w_x\sigma_i) \cap dom(\sigma_i) = \emptyset$, for every $x \in X_M$. In other words, at every point of the computation, the set of variables that occur in the domain of σ_i is disjoint from the set of variables that occur in $w_x\sigma_i$, $x \in X_M$. This invariant means that there are no circular dependencies, and helps us prove that eventually, every variable will be resolved. Intuitively, the invariant holds because every time σ_i is updated by adding a variable x to its domain, the assignment on Line 5 of Algorithm 2 clears x from $w_x\sigma_i$ while keeping $\mathrm{Max}(M, v_0) = \mathrm{Max}(M, \overline{\sigma}_i)$. The formal argument is:

Lemma 5 *At every point of the computation, $var(w_x\sigma_i) \cap dom(\sigma_i) = \emptyset$, for every $x \in X_M$.*

Proof The proof is by induction. Lemma 5 is trivially true after the initialisation of σ_0 in Algorithm 1.

Consider the assignment to σ_{i+1} on Line 5 of Algorithm 2. By the induction hypothesis, $var(w_x\sigma_i) \cap dom(\sigma_i) = \emptyset$. Since $x \notin var(w_x\sigma_i\{x \leftarrow \top\})$, it follows that $x \notin var(w_{x'}\sigma_i\{x \leftarrow w_x\sigma_i\{x \leftarrow \top\}\})$ for every $x' \in X_M$, so σ_i can safely be updated to $\sigma_i\{x \leftarrow w_x\sigma_i\{x \leftarrow \top\}\}$ without invalidating Lemma 5. □

Let us now ensure that the recursive calls always come to an end.

Lemma 6 *Algorithm 1 terminates.*

Proof We need only consider calls to function *equiv*. Since S grows with each recursive call to *equiv* on Line 3 of Algorithm 2, the recursion is finite. Due to Line 5, each call to *equiv* terminates with $dom(\sigma_i)$ greater than before, hence the number of calls of the while-loop is also finite. □

It remains to verify every intermediate solution is a partial solution.

Lemma 7 *Throughout the execution of Algorithm 1, and for every $x \in dom(\sigma_i)$, the formula $(x \to w_x)\sigma_i$ is a tautology.*

Proof The proof is by induction on the index of σ_i. There are two cases. First, if $\sigma_0(x) = \top$, then $x = \langle p, p \rangle$ for some $p \in Q$, which means that by Definition 2 of the characteristic formula, $w_x\sigma_0$ is a tautology, and so is $(x \to w_x)\sigma_0$. Second, If $\sigma_0(x) = \bot$, then $(\bot \to w_x)\sigma_0$ is clearly a tautology.

We continue to consider the inductive step, which extends the substitution by letting $\sigma_{i+1} = \sigma_i\{y \leftarrow w_y\sigma_i\{y \leftarrow \top\}\}$. For every $x \in dom(\sigma_{i+1})$, there are two cases:

- The variable $x \in dom(\sigma_i)$. By the induction hypothesis, $(x \to w_x)\sigma_i$ is a tautology, and replacing every occurrence of a variable in a tautology with one and the same formula yields a new tautology.
- The variable $x = y$, in which case $(y \to w_y)\sigma_{i+1}$ expands to the tautology

$$((w_y\sigma_i)\{y \leftarrow \top\}) \to ((w_y\sigma_i)\{y \leftarrow (w_y\sigma_i)\{y \leftarrow \top\}\}),$$

which completes the proof. □

Lemma 8 *Throughout the execution of Algorithm 1,* $Max(M, v_0) \in Mod(w_M\sigma_i)$.

Proof The proof is by induction on the index of σ_i. By construction, $\overline{\sigma}_0 = v_0$, which establishes the base case.

We continue to consider the inductive step, which extends the substitution by letting $\sigma_{i+1} = \sigma_i\{x \leftarrow w_x\sigma_i\{x \leftarrow \top\}\}$.

We prove that for every $x' \in X_M$, $\mathrm{Max}(M, v_0) \in \mathrm{Mod}((x' \to w_{x'})\sigma_{i+1})$. Due to the conjunctive structure of w_M, we can take advantage of the fact that

$$\mathrm{Mod}(w_M\sigma_{i+1}) = \bigcap_{x' \in X_M} \mathrm{Mod}((x' \to w_{x'})\sigma_{i+1}).$$

The argument has three cases:

- The variable $x' \in dom(\sigma_i)$. By Lemma 7, $(x' \to w_{x'})\sigma_{i+1}$ is a tautology. This ensures that $\mathrm{Max}(M, v_0) \in \mathrm{Mod}((x' \to w_{x'})\sigma_{i+1})$.
- The variable $x' = x$, in which case $(x \to w_x)\sigma_{i+1}$ is again a tautology by Lemma 7, so $\mathrm{Max}(M, v_0) \in \mathrm{Mod}((x' \to w_{x'})\sigma_{i+1})$.
- The variable $x' \in X_M \backslash dom(\sigma_{i+1})$. By the induction hypothesis, the model $\mathrm{Max}(M, v_0) \in \mathrm{Mod}((x' \to w_{x'})\sigma_i)$, and if $x' \notin dom(\sigma_{i+1})$, then $x' \notin dom(\sigma_i)$, so the model $\mathrm{Max}(M, v_0) \in \mathrm{Mod}(x' \to w_{x'}\sigma_i)$. If $\mathrm{Max}(M, v_0)(x) = \top$, then $\mathrm{Max}(M, v_0)(w_x\sigma_i) = \top$, and since w_x is negation free, it must be the case that $\mathrm{Max}(M, v_0)(w_x\sigma_i\{x \leftarrow \top\}) = \top$, so $\mathrm{Max}(M, v_0)$ is in

$$\mathrm{Mod}(x' \to w_{x'}(\sigma_i\{x \leftarrow w_x\sigma_i\{x \leftarrow \top\}\}) = \mathrm{Mod}((x' \to w_{x'})\sigma_{i+1}).$$

This completes the case analysis and the proof. □

Lemma 9 *Throughout the execution of Algorithm 1,* $Max(M, v_0) = Max(M, \overline{\sigma}_i)$.

Proof The proof is by induction on the index of σ_i. By construction, $\overline{\sigma}_0 = v_0$, so the base case is trivially true.

We continue to consider the inductive step, which extends the substitution by letting $\sigma_{i+1} = \sigma_i\{x \leftarrow w_x\sigma_i\{x \leftarrow \top\}\}$.

We first observe that if σ_i is updated to $\sigma_i' = \sigma_i\{x \leftarrow w_x\}$, then by Lemma 8, we have $\mathrm{Max}(M, v_0) \in \mathrm{Ext}(\overline{\sigma}_i) \cap \mathrm{Mod}(w_M\sigma_i)$, from which it follows that $\mathrm{Max}(M, v_0) \in \mathrm{Ext}(\overline{\sigma}_i')$. Next, we note that $x \to w_x \equiv (x \to w_x\{x \leftarrow \top\})$, so since $\mathrm{Max}(M, v_0) \in \mathrm{Ext}(\overline{\sigma}_i')$, we also have $\mathrm{Max}(M, v_0) \in \mathrm{Ext}(\overline{\sigma}_{i+1})$. □

Let σ_t be the value of σ_i at the point of termination, in other words, when control reaches Line 6 of Algorithm 1.

Observation 2 *Since $var(w_x\sigma_t) = \emptyset$, for every $x \in X_M$, the interpretation $\overline{\sigma}_t$ is total.*

Lemmas 6, 9, and Observation 2 are combined in Theorem 3.

Theorem 3 *Algorithm 1 terminates, and when it does, the relation $\sim_{\overline{\sigma}_t}$ is the unique coarsest bisimulation equivalence on M.*

4.2 Complexity

Let us now discuss the efficient implementation of Algorithm 1. The key idea is to keep the representation of the characteristic formula and the computed substitutions small by linking recurring structures, rather than copying them. We use the parameter r to capture the amount of nondeterminism in M. It is defined as $r = max_{q\in Q, f\in\Sigma} \left|\delta_f(q)\right|$. In particular, $r \leq 1$ whenever the automaton M is deterministic.

Let us denote the *union* of all w_x, $x \in X_M$, in other words, the formulas that appear as right-hand sides in w_M, by rhs_M. In the update of σ_i on Line 5, some of these formulas may be copied into others, so the growth of $rhs_M\sigma_i$ is potentially exponential. For the sake of compactness we therefore represent $rhs_M\sigma_i$ as a directed acyclic graph (DAG) and allow node sharing between formulas. In the following, we represent a DAG as a tuple (V, E, l), where V is a set of *nodes*, $E \subseteq V \times V$ is a set of *(directed) edges*, and $l : V \to X_M \cup \{\vee, \wedge, \bot, \top\}$ is a labelling function that labels each node with a variable name or logical symbol. In the initial DAG, only nodes representing variables and the logical constants $\top$ and $\bot$ are shared, but as the algorithm proceeds, more substantial parts of the graph come to overlap. The construction is straight-forward but has many steps, so readers that are satisfied with a high-level view may want to continue to Theorem 4.

Definition 6 (*DAG representation of formulas*) Let L be the propositional logic $(X_M, \{\vee, \wedge, \top, \bot\})$ and let $w \in \mathrm{WF}(L)$. The (rooted, labelled) *DAG representation* $D(w)$ of w is recursively defined. For every $x \in V_M \cup \{\top, \bot\}$,

$$D(x) = (\{u\}, \emptyset, \{(u, x)\}) \text{ with } \mathrm{root}(D(x)) = u.$$

The DAG $D(x)$ thus consists of a single node u labelled x, and u is the root of $D(x)$. For $\otimes \in \{\vee, \wedge\}$ and $w, w' \in \mathrm{WF}(L)$, we derive $D(w \otimes w')$ from $D(w) = (V, E, l)$ and $D(w') = (V', E', l')$ by letting

$$\begin{aligned} D(w \otimes w') = (&V \cup V' \cup \{u\},\\ &E \cup E' \cup \{(u, \mathrm{root}(D(w))),\ (u, \mathrm{root}(D(w')))\},\\ &l \cup l' \cup \{(u, \otimes)\}), \end{aligned}$$

where $\mathrm{root}(D(w \otimes w')) = u$, and then merging leaf nodes with identical labels.

Given the above definition, we obtain the many-rooted DAG representation $D(rhs_M)$ of rhs_M by taking the disjoint union of $D(w_x)$, $w_x \in rhs_M$, and merging all leaf nodes that have identical labels. Thus, for each state pair x and for each of $\top$ and $\bot$, there is a single leaf node in $D(rhs_M)$.

Throughout the computation, we maintain a DAG representing $D(rhs_M\sigma_i)$. This is initialised to $D(rhs_M\emptyset)$ and then immediately updated to $D(rhs_M\sigma_0)$. On top of this DAG, we assume that for each pair x, we have a reference $ref_{rhs}(x)$ to w_x, in other words, to the

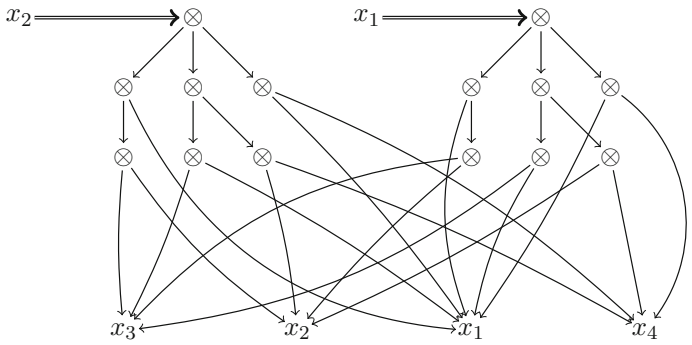


Fig. 2 An example initial DAG $D(rhs_M\sigma_0)$ with state-pair variables $x_1, \ldots, x_4$. References ref_{rhs} are drawn with double-lined arrows. The symbol $\otimes$ denotes a node labeled by $\wedge$ or $\vee$

corresponding right-hand side representation in the DAG. Figure 2 illustrates the structure of the initial DAG.

During the computation, the graph $D(rhs_M\sigma_i)$ is reorganised by changing the targets of certain edges, but $D(rhs_M\sigma_i)$ does not grow. The exceptions are a potential once-off addition of $\top$ and $\bot$ labelled nodes during initialisation in Algorithm 1, and the addition of a single outgoing edge to each of the initial leave nodes. Moreover, every time a variable is resolved, $D(rhs_M\sigma_i)$ is updated to reflect this; while the $ref_{rhs}(x)$'s will continue to point at $w_x\sigma_i$, the expression $w_x\sigma_i$ changes to reflect the latest σ_i, and will be simplified as much as possible.

There are two cases to consider at Line 5 of Algorithm 2. The first of these is that $w_x\sigma_i\{x \leftarrow \top\}$ resolves to $\top$ or $\bot$. In this case a number of adjustments are made to $D(rhs_M\sigma_i)$ to reflect the updated $w_x\sigma_{i+1}$ (illustrated in Fig. 3):

1. The formula $w_x\sigma_i$ in $D(rhs_M\sigma_i)$ is replaced by $\top$ or $\bot$ as the case may be. Thus, the graph $D(rhs_M\sigma_i)$ is modified to remove the nodes and edges of such $w_x\sigma_i$.
2. The unique shared leaf node representing x in the DAG is re-labeled to either $\top$ or $\bot$.
3. The re-labeling is propagated upwards along each DAG branch leading to this node, now labeled $\top$ respectively $\bot$, as this resolution of x may lead subtrees rooted further up this branch to resolve to either $\bot$ or $\top$ as well. In the case of $\bot$, if the immediate parent is labeled by $\wedge$, then it can be resolved to (i.e., replaced by a reference to) $\bot$. If the parent is instead labelled $\vee$, then we can simplify the graph by deleting the edge and if it was the last edge, also resolving the parent to $\bot$. In the case of $\top$ and parent $\vee$, the parent can be resolved to $\top$. In the case of $\top$ with parent $\wedge$, a simplification is possible by deleting the edge between them, and if it was the last edge, resolving the parent to $\top$. This processes continues until no more simplifications or resolutions are possible.

In the second case, $w_x\sigma_i\{x \leftarrow \top\}$ does *not* resolve to $\bot$ or $\top$. Here, as illustrated in Fig. 4, two updates are made to $D(rhs_M\sigma_i)$:

1. The references in $w_x\sigma_i$ to the unique shared leaf node for x itself are replaced by references to $\top$.
2. The change is propagated upwards along each DAG branch leading to this reference to $\top$, as this local resolution of x may either simplify (in case of $\wedge$) or resolve (in case of $\vee$) subtrees rooted further up in $rhs_M\sigma_i$. The resulting modified right-hand side $w_x\sigma_{i+1}$ may either resolve to $\top$, or still be a proper tree. In the first case, the unique shared leaf node representing x in the DAG is re-labeled to $\top$. This change is then propagated upwards, as

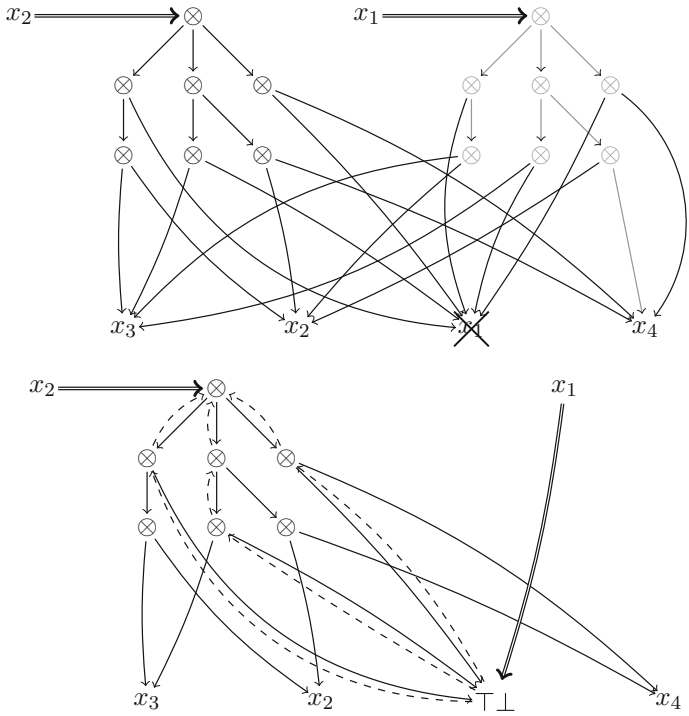


Fig. 3 The update of the DAG $D(rhs_M\sigma_i)$ in the case where x gets resolved to either $\top$ or $\bot$. The symbol $\otimes$ denotes a node labeled by either $\wedge$ or $\vee$. The upper part of the images shows the nodes and edges that are about to be deleted (outlined in gray), and the lower part shows how the information is propagated through the graph (dashed lines)

described in the previous paragraph. In the second case, the node x may still be used in right-hand sides other than $w_x\sigma_{i+1}$, and is replaced there by a reference to the modified $w_x\sigma_{i+1}$.

The above graph manipulations permit an efficient implementation of Algorithm 1. In our complexity analysis, we assume that the sets of state pairs involved provide constant time insertion and deletion. This is an idealized view of the matter. In practice, one can represent such a set as a hash table indexed by pairs of states. With this implementation, both set operators will essentially take constant time, as long as the hash table is sufficiently large to make hash collisions rare (see [10, p. 207]).

Theorem 4 (Complexity) *Algorithm 1 is in* $O\big(n^2r^2\,|\Sigma|\big)$.

Proof The initialisation of σ_0 in Algorithm 1 can be done it $O(n^2)$, whereupon the algorithm proceeds to call Algorithm 2, which is in total called $O(n^2)$ times, over the entire execution of Algorithm 1.

Let us look closer at the body of Algorithm 2 on input x and S. To satisfy the existence clause in the 'while' loop of Algorithm 2, the algorithm needs to decide what variable to resolve next. To do this, the algorithm finds the left-most leaf (i.e., node with no outgoing edges) in the DAG representation of $w_x\sigma_i$. In other words, the algorithm follows the left-most path from the root downwards in $w_x\sigma_i$ (the top-most subfigure of Fig. 2 gives an idea of what path looks like).

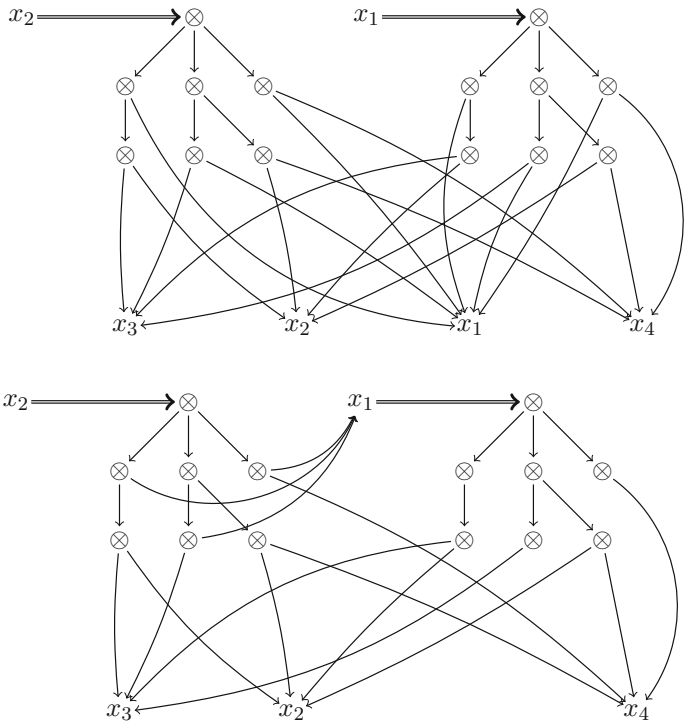


Fig. 4 The update of the DAG $D(rhs_M\sigma_i)$ in the case where x is *not* resolved to $\top$ or $\bot$. The symbol $\otimes$ denotes a node labeled by either $\wedge$ or $\vee$. The upper image shows the nodes and edges that are about to be deleted (outlined in gray), and the lower image shows the new edges added in

In total, over the entire run of Algorithm 1, the algorithm must in the worst case traverse each path from the root of $D(w_x)$ to a leaf in $D(w_x)$, for every $x \in X_M$. The algorithm must however only go *down* each path once, even when the representations of the formula $w_x, x \in X_M$ start to link up (like in the lower subfigure Fig. 4). The reason is that after a call to *equiv* with argument x and S, the only variables left in $w_x\sigma_i$ are those in S, and these will all be resolved to $\top$ or $\bot$ at a later stages of the algorithm, at which points the DAG representing $w_x\sigma$ will be simplified, until it has at last a decided Boolean value. The cost for these simplifications is discussed in the next paragraph, and the cost for deciding what variable to resolve next is a total $|D(rhs_M)| = O(n^2r^2\,|\Sigma|)$ when summed over every call to Algorithm 2.

The update to σ_i on Line 5 is where the majority of the work is done. First, the local substitution $w_x\sigma_i\{x \leftarrow \top\}$ on Line 5 requires $O(r^2\,|\Sigma|)$ steps. As argued above, there are two cases:

- If $w_x\sigma_i\{x \leftarrow \top\}$ resolves to $\top$ or $\bot$, we start at the node labelled x and follow all edges in $D(rhs_M\sigma_i)$ backwards, updating the graph structure to reflect the truth value of x. Every time we follow an edge, we are able to simplify $D(rhs_M\sigma_i)$ by removing at least one edge and one node. This means that the total amount of work done at Line 5 is in $O(|D(rhs_M\sigma_i)|) = O(n^2r^2\,|\Sigma|)$ when summed over every call to Algorithm 2.
- If, instead, $w_x\sigma_i\{x \leftarrow \top\}$ is not resolved, then the update of σ_i to σ_{i+1} only takes constant time, as it only involves the redirection of one pointer.

In summary, the time complexity of Algorithm 2 is

$$O\big(n^2 + n^2 r^2 \,|\Sigma| + n^2 \cdot (r^2 \,|\Sigma|) + n^2 r^2 \,|\Sigma|\big)\,,$$

which simplifies to the expression in the statement of Theorem 4. □

Recall that bisimulation minimization coincides with classical minimization in the case of DFA. Since we always have $r \leq 1$ for such devices, we arrive at Corollary 1. This means that the runtime of our algorithm, when applied to deterministic automata, is comparable to that of the algorithm by Watson and Daciuk [24].

Corollary 1 *Algorithm 1 minimizes any* DFA *in* $O\big(n^2 \,|\Sigma|\big)$.

4.3 Lazy evaluation and heuristic improvements

We argued from the outset that one advantage of the aggregation approach is that also intermediate solutions are language-equivalent with the original automaton. Let us now show that every time that the call hierarchy returns to the level of Algorithm 1 (i.e., the function *minimize*), every variable $x \in X_M$ on which *equiv* has been called, either from Algorithm 1 or through a recursive call from *equiv* itself, has also been resolved to $\top$ or $\bot$.

We use $var(\sigma_i)$ as shorthand for $\bigcup_{x \in dom(\sigma_i)} var(\sigma_i(x))$.

Lemma 10 *Throughout the computation,* $var(\sigma_i) \subseteq S$.

Proof The proof is by induction on the index of σ_i. When Algorithm 2 is invoked for the first time, $var(\sigma_0) = \emptyset$, so the statement is true.

In the construction of σ_{i+1} we know from the induction hypothesis that $var(\sigma_i) \subseteq S$. The modification of σ_i consists of substituting $w_x\sigma_i\{x \leftarrow \top\}$ for x. The right-hand side w_x only contains variables that are in S, or in $dom(\sigma_i)$, so by the induction hypothesis $var(w_x\sigma_i) \subseteq S$. Finally, all occurrences of $x \in var(w_x\sigma_i)$ are replaced by $\top$, so x can safely be removed from S when the function exits, without violating the statement of Lemma 10. □

Theorem 5 *Every time the process control returns to* minimize*, every pair x on which equiv has been called is resolved.*

Proof Every time the call hierarchy returns to *minimize*, the stack S empties. Let σ_i be the state of the substitution when this happens. By Lemma 10, $var(w_x\sigma_i) = \emptyset$, so $w_x\sigma_i$ is resolved. □

Finally, we note that Algorithm 1 can be extended to add $(p, q) \to \bot$ to σ_0 for each pair of states p and q such that the sets of unique symbols that label the transitions from p and q differ. This technique does not affect the asymptotic running time of the algorithm, but may be of practical value.

5 Conclusion

We have presented a minimization algorithm for NFAs that identifies and merges bisimulation-equivalent states. In terms of running time, it is as efficient as any existing aggregation-based minimisation algorithm for DFAs, but less efficient than the fastest refinement-based minimisation algorithms derived from Hopcroft's algorithm for NFAs. However, compared

to the latter group, it has the advantage that intermediate solutions are usable for language-preserving reduction of the input automaton M.

The algorithm is the first to compute the coarsest bisimulation relation on M through partition aggregation. Also the logical framework used for representation and computation appears to be new for this application. For this reason, the investigation of optimization techniques similar to those used in SAT solvers is an interesting future endeavour. Furthermore the generalization of the algorithm to, for example, nondeterministic tree automata could be considered.

A disadvantage of the proposed algorithm is that it builds the entire characteristic formula for the input machine, before the actual evaluation starts. For some cases, where state similarity can be disregarded after considering a fraction of the formula, this is not well-spent work. We are therefore interested in approaches that assemble the clauses of the characteristic formula as they are needed. The updated algorithm would likely have the same worst-case complexity as the original one, but may perform better in the average case. Average time complexity is in itself a relevant item of study. Recently, Bassino et al. [4] and David [11] published on the average-case complexities of the well-known Hopcroft's and Moore's algorithms, and a similar analysis could shed further light on Algorithm 1.

Acknowledgements Open access funding provided by Umea University.

Funding Funding was provided by the Swedish Research Council (Grant No. 621-2012-455).

References

1. Abdulla, P.A., Holík, L., Kaati, L., Vojnar, T.: A uniform (bi-)simulation-based framework for reducing tree automata. Electron. Notes Theor. Comput. Sci. **251**, 27–48 (2009)
2. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
3. Almeida, M., Moreira, N., Reis, R.: Incremental DFA minimisation. RAIRO Theor. Inform. Appl. **48**(2), 173–186 (2014). https://doi.org/10.1051/ita/2013045
4. Bassino, F., David, J., Nicaud, C.: On the average complexity of Moore's state minimization algorithm. In: Proceedings of the 26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009), Freiburg, Germany, Leibniz International Proceedings in Informatics (LIPIcs), vol. 3, pp. 123–134 (2009)
5. Björklund, J., Cleophas, L.: Minimization of finite state automata through partition aggregation. In: Drewes, F., Martín-Vide, C., Truthe, B. (eds.) Proceedings of the 11th International Conference on Language and Automata Theory and Applications (LATA 2017), Umeå, Sweden, Lecture Notes in Computer Science, vol. 10168, pp. 223–235 (2017)
6. Björklund, J., Maletti, A., May, J.: Backward and forward bisimulation minimization of tree automata. Theor. Comput. Sci. **410**(37), 3539–3552 (2009)
7. Björklund, J., Maletti, A., Vogler, H.: Bisimulation minimisation of weighted automata on unranked trees. Fundamenta Informatica **92**(1–2), 103–130 (2009)
8. Buchholz, P.: Bisimulation relations for weighted automata. Theor. Comput. Sci. **393**(1–3), 109–123 (2008)

9. Cleophas, L., Kourie, D.G., Strauss, T.: Watson BW (2009) On minimizing deterministic tree automata. In: Holub, J., Žďárek, J. (eds.) Prague Stringology Conference, Prague, Czech Republic, pp. 173–182 (2009)
10. Daciuk, J.: Optimization of Automata. Gdańsk University of Technology Publishing House, Gdańsk (2014)
11. David, J.: Average complexity of Moore's and Hopcroft's algorithms. Theor. Comput. Sci. **417**, 50–65 (2012)
12. Gramlich, G., Schnitger, G.: Minimizing NFA's and regular expressions. J. Comput. Syst. Sci. **73**(6), 908–923 (2007)
13. Hopcroft, J.E.: An n log n algorithm for minimizing the states in a finite automaton. In: Kohavi, Z. (ed.) The Theory of Machines and Computations, pp. 189–196. Academic Press, Cambridge (1971)
14. Hopcroft, J.E., Ullman, J.D.: Set merging algorithms. SIAM J. Comput. **2**(4), 294–303 (1973). https://doi.org/10.1137/0202024
15. Huffman, D.A.: The synthesis of sequential switching circuits. J. Frankl. Inst. **257**, 161–190 and 275–303 (1954)
16. Maletti, A.: Minimizing weighted tree grammars using simulation. In: Finite-State Methods and Natural Language Processing, Pretoria, South Africa, 2009, pp. 56–68. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14684-8_7
17. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: 13th Annual IEEE Symposium on Switching and Automata Theory, pp. 125–129 (1972)
18. Moore, E.F.: Gedanken-experiments on sequential machines. In: Shannon, C., McCarthy, J. (eds.) Automata Studies, pp. 129–153. Princeton University Press, Princeton (1956)
19. Nerode, A.: Linear automaton transformations. Proc. Am. Math. Soc. **9**(4), 541–544 (1958)
20. Paige, R., Tarjan, R.: Three partition refinement algorithms. SIAM J. Comput. **16**(6), 973–989 (1987)
21. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. J. ACM **22**(2), 215–225 (1975). https://doi.org/10.1145/321879.321884
22. ten Eikelder H (1991) Some algorithms to decide the equivalence of recursive types. Technical Report 93/32, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven
23. Watson, B.W.: Taxonomies and toolkits of regular language algorithms. Ph.D. thesis, Department of Mathematics and Computer Science, TU Eindhoven (1995)
24. Watson, B.W., Daciuk, J.: An efficient incremental DFA minimization algorithm. Nat. Lang. Eng. **9**(1), 49–64 (2003). https://doi.org/10.1017/s1351324903003127