Minimization of finite state automata through partition aggregation

Johanna Björklund[1] and Loek Cleophas[1,2]

[1] Umeå University, Dept. of Computing Science, Umeå, Sweden
[2] Stellenbosch University, Dept. of Information Science, Stellenbosch, South Africa
johanna@cs.umu.se, loek@fastar.org

Abstract. We present a minimization algorithm for finite state automata that finds and merges bisimulation-equivalent states, identified through partition aggregation. We show the algorithm to be correct and run in time $O\big(n^2d^2\,|\Sigma|\big)$, where n is the number of states of the input automaton M, d is the maximal outdegree in the transition graph for any combination of state and input symbol, and $|\Sigma|$ is the size of the input alphabet. The algorithm is slower than those based on partition refinement, but has the advantage that intermediate solutions are also language equivalent to M. As a result, the algorithm can be interrupted or put on hold as needed, and the derived automaton is still useful. Furthermore, the algorithm essentially searches for the maximal model of a characteristic formula for M, so many of the optimisation techniques used to gain efficiency in SAT solvers are likely to apply.

1 Introduction

Finite-state automata form one of the key concepts of theoretical computer science, and their computational and representational complexity is the subject of a great body of work. In the case of *deterministic* finite state automata (DFA), there is for every DFA M a minimal language-equivalent DFA M', and this M' is canonical with respect to the recognized language. In the general, non-deterministic case, no analogous result exists. In fact, NFA minimization is PSPACE complete [14] and the solution is not guaranteed to be unique. Moreover, given an NFA with n states, the minimization problem cannot be efficiently approximated within a factor $o(n)$, unless P = PSPACE [9].

Since NFA minimization is inherently difficult, attention has turned to efficient heuristic algorithms, that often, if not always, perform well. In this category we find bisimulation minimization. Intuitively, two states are bisimulation equivalent if every transition 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 a 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 exists 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 predominant method of finding $\mathcal{E}$ is through partition refinement. The states are initially divided into final and non-final states, and the 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 [17], 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 Section 4, we present an NFA minimization algorithm in which also the intermediate solutions are language-equivalent with 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 $O\big(n^2 d^2 \left|\Sigma\right|\big)$, where d is the maximal outdegree in the transition graph for any combination of state and input symbol, and Σ is the input alphabet.

The use of aggregation was inspired by a family of minimization algorithms for DFA (see Section 1.1), and we lift the technique non-deterministic devices. In the deterministic case, our algorithm runs in $O\big(n^2 \left|\Sigma\right|\big)$, 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 entails finding the 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 [12, 15, 10]. Ten Eikelder 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 [19]. This procedure was later used by Watson to formulate a DFA minimization algorithm that works through partition aggregation [20]. The algorithm runs in exponential time, and two mutually exclusive optimization methods were discussed in [21]. One uses memoziation to limit the number of recursive invocations; the other bases the implementation on the union-find data structure [2, 18, 11]. The latter method reduces the complexity from $O\big(|\Sigma|^{n-2} n^2\big)$ to $O\big(\alpha(n^2) n^2\big)$, where $\alpha(n)$, rougly 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 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 [7]. In [8, Chapter 7], Daciuk continues the discussion

of aggregation-based DFA minimization, starting from the work reported in [21]. He simplifies the presentation and generally improves the algorithm, including the removal of an incorrect combination of memoization and restricted recursion depth. The fact that this combination was problematic had been pointed out by Marco Almeida, and is also reported in [3]; Almeida had found apparently rare DFA cases in which the Watson-Daciuk algorithm returned non-minimal DFAs. In [3], Almeida et al. also present 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 $O\big(\alpha(n^2)n^2\big)$ for all practical cases, yet also claim it to be faster than the Watson-Daciuk one. Based on Almeida's reporting, Daciuk in [8, Section 7.4] provides a new version, presented as a compromise between the corrected Watson-Daciuk and the Almeida-Moreira-Reis algorithm, but does not discuss its efficiency. NFA minimisation has also received much attention, but we restrict ourselves to heuristics that compute weaker relations than the actual Nerode congruence (recalled in Section 2). In [17], three partition refinement algorithms were presented, one of which is essentially bisimulation minimization for NFA. The technique was revived in [1], then in the domain of finite-state tree automata. The paper was soon followed by bisimulation-minimization algorithms for weighted and unranked tree automata [4, 5], and also algorithms based on more general simulation relations [1, 13]. This work is to the best of our knowledge the first in which the bisimulation relation is computed through aggregation.

2 Preliminaries

Sets and numbers. 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, in particular, $[0] = \emptyset$. The cardinality of a set S is written $|S|$ and the powerset of S by $pow(S)$.

Relations. 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* 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. We denote the identity relation $\{(s, s) \mid s \in S\}$ on S by $\mathcal{I}_S$.

Strings. An alphabet is a finite nonempty set. The empty string is denoted by ε. For an alphabet Σ, a string is a sequence of symbols from Σ, in other words, an element of Σ^*. A string language is a subset of Σ^*.

Finite automata. A *non-deterministic finite state automaton* (or NFA, for short) 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*; $\delta = (\delta_f)_{f \in \Sigma}$ is a family of transition functions $\delta_f \colon Q \to pow(Q)$; $Q_I \subseteq Q$ is a set of *initial states*; and $Q_F \subseteq Q$ is a set of *final states*. The size of M is $|M| = |\delta|$.

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

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

The language recognised by M is $\mathcal{L}(M) = \{w \in \Sigma^* \mid \hat{\delta}_w(Q_I) \cap Q_F \neq \emptyset\}$. From here on, we identify δ with $\hat{\delta}$. If $|Q_I| \leq 1$, and if $|\delta_f(\{q\})| \leq 1$ 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 [16] is the coarsest *congruence relation* $\mathcal{E}$ on Q w.r.t the right-languages $\overrightarrow{\mathcal{L}}(q)$; i.e. $\mathcal{E}(p, q)$ if and only if $\overrightarrow{\mathcal{L}}(p) = \overrightarrow{\mathcal{L}}(q)$ for all $p, q \in Q$.

Propositional logic. We assume that the reader is familiar with propositional logic, but recall a few basics to fix the terminology. 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 V, and let $\mathrm{WF}(L)$ be the set of well-formed formulas over L. An *interpretation* of L is a partial function $V \to \mathbb{B}$. Given interpretations v and v', we say that v' is an *extension* of v if $v'(\pi) = v(\pi)$ for all $\pi \in dom(v)$. The set of all such extensions is written $\mathrm{Ext}(v)$.

A *substitution* of formulas for variables 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 $w_i \in \mathrm{WF}(L)$ is a formula. The *empty* substition 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 interpretation v of L can be seen as a substitution, in which $\pi \in dom(v)$ is replaced by $v(\pi)$. This allows us to extend v to a function $\mathrm{WF}(L) \to \mathbb{B}$ where $v(w) = \top$ if wv is a tautology, $v(w) = \bot$ if wv is a contradiction, and $v(w)$ is undefined otherwise. The formula w is *resolved* by v if $v(w) \in \{\top, \bot\}$, and v is a *model* for w if $v(w) = \top$. The set of models of w is denoted by $\mathrm{Mod}(w)$.

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. Due to space restrictions, the argumentation is kept at an intuitive level [3].

[3] Detailed proofs are provided in the appendix.

From here on, let $M = (Q, \Sigma, \delta, Q_I, Q_F)$ be a fixed but arbitrary NFA.

Definition 1 (Bisimulation, cf. [6, 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 $f \in \Sigma$,*

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

Condition 2 of Definition 1 can be expressed in a propositional logic, in which the variables are pairs of automata states. The variable corresponding to the pair $\langle p, q \rangle$ is true if and only if p and q satisfy Condition 2.

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

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

and by w_π the formula $\bigwedge_{f \in \Sigma} w_\pi^a$. Observe that w_π is negation-free. Finally, we denote by w_M the conjunction $\bigwedge_{\pi \in V_M} \pi \to w_\pi$.

We could also model Condition 1 in the formula w_M, but that would introduce negations and require a more complex algorithm than that which we are to present. To find the coarsest bisimulation relation for M, we instead start out from a partial interpretation of V_M satisfying Condition 1 of Definition 1 and search for a 'maximal' 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 V_M. Then $v \vee v'$ denotes the interpretation of V_M given by $(v \vee v')(\pi) = v(\pi) \vee v'(\pi)$, for every $\pi \in V_M$. We say that $v \in Mod(w)$ is* maximal *if $v \vee v' = v$ for every $v' \in Ext(v) \cap Mod(w)$.*

Due to the structure of w_M, its models are closed under variable-wise 'or'. In other words, if $v, v' \in \mathrm{Mod}(w_M)$, then $v \vee v' \in \mathrm{Mod}(w_M)$. From this we conclude that when a solution exists, it is unique.

Lemma 1. *Let v be a partial interpretation of V_M. If $Ext(v) \cap Mod(w_M) \neq \emptyset$, then there is a $\hat{v} \in Ext(v)$ that is a maximal model for w_M, and $\hat{v}$ is unique.*

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 V_M a relation $\sim_v$ on V_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 makes no difference between a state pair π for which $v(\pi) = \bot$, and a state pair for which v is undefined.

If v is an arbitrary model for w_M, then its associated relation need not be an equivalence relation, but for the maximal model, it is.

Lemma 2. *Let v be a partial interpretation of V_M such that $\sim_v$ is an equivalence relation, and let $\hat{v}$ be an extension of v that is a maximal model of V_M, then also $\sim_{\hat{v}}$ is an equivalence relation.*

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 V_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 undefined on all other state pairs.

Lemma 3. *$v_0 \in Mod(w_M)$ and $\sim_{v_0}$ is an equivalence relation.*

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

4 Algorithm

Aggregation-based algorithms for automata minimization start with a singleton partition, in which each state is viewed as a separate equivalence class, and iteratively merge partitions found to be equivalent. When all states are mutually distinguishable, the algorithm terminates. We use 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 an interpretation $\hat{v}$ of the set of variables $V_M = \{(p, q) \mid p, q \in Q\}$, where $\hat{v}(\pi) = \top$ means that π is a pair of equivalent states, and $\hat{v}(\pi) = \bot$ that π is a pair of distinguishable states. The interpretation $\hat{v}$ is an extension of v_0 as in Definition 5, and a maximal model for the characteristic formula w_M. Due to the structure of w_M such a model can, as we shall see, be computed greedily.

The construction of $\hat{v}$ is done by incrementally building up a substitution σ_i that replaces state pairs by logical formulas. This is done in such a way 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 V_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).

Algorithm 1 Aggregation-based bisimulation minimization algorithm.

1: **function** *minimize*(M)
2: $\quad\sigma_0 ::= \{\langle q, q\rangle \leftarrow \top \mid q \in Q\} \cup \{\langle p, q\rangle \leftarrow \bot \mid (p \in Q_F) \not\equiv (q \in Q_F)\}$
3: $\quad$**for** $\pi \in V_M \setminus dom(\sigma_i)$ **do**
4: $\quad\quad$*equiv*$(\pi, \{\pi\})$
5: $\quad$**end for**
6: $\quad$**return** $(M/\sim_{\sigma_i})$
7: **end function**

Algorithm 2 Point-wise computation of $\pi \in V_M$

1: **function** *equiv*(π, S)
2: $\quad$**while** $\exists \pi' \in var(w_\pi) \setminus dom(\sigma_i) \setminus S$ and $w_\pi\sigma_i$ is not resolved **do**
3: $\quad\quad$*equiv*$(\pi', S \cup \{\pi'\})$
4: $\quad$**end while**
5: $\quad$**if** $w_\pi\sigma_i$ is resolved **then**
6: $\quad\quad\sigma_{i+1} ::= \sigma_i\{\pi \leftarrow w_\pi\sigma_i\}$
7: $\quad$**else**
8: $\quad\quad\sigma_{i+1} ::= \sigma_i\{\pi \leftarrow w_\pi\sigma_i\{\pi \leftarrow \top\}\}$
9: $\quad$**end if**
10: **end function**

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 π for which equivalence or distinguishability should be determined, and a set S of such pairs. This set consists of pairs of states that are under investigation in earlier, 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_π, but which have note yet been resolved, nor form part of the call stack S. After these calls have been completed and we have exited the while loop, $w_\pi\sigma_i$ may or may not have been resolved. If it has, then we extend the substitution with the appropriate substitution rule, that is, one which replaces occurrences of π by the value of $w_\pi\sigma_i$. If $w_\pi\sigma_i$ has *not* been resolved, then we first rewrite $w_\pi\sigma_i$ by replacing every occurrence of π by $\top$ and then extend the substitution with a rule substituting π by $w_\pi\sigma_i\{\pi \leftarrow \top\}$. These operations clear cyclic dependencies, and the maximal model for the updated formula is the maximal model for the original one.

4.1 Correctness

Lemma 7 below says that during 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 $w_\pi\sigma_i$, $\pi \in V_M$. This avoids circular dependencies, and helps us prove that eventually, every variable will be resolved. Intuitively, it holds because every

time we update σ_i by adding a particular π to its domain, the assignment on Line 8 clears π from w_π in such a way that $\mathrm{Mod}(w_M\sigma_i)$ is kept unchanged.

It can be shown that throughout the computation, $var(w_\pi\sigma_i) \cap dom(\sigma_i) = \emptyset$, for every $\pi \in V_M$. Furthermore, it is always the case that $\hat{v}(v_0(w_M)) = \hat{v}(w_M\sigma_i)$.

We now make the following observation: Let σ_t be σ_i at the point of termination. Let $\overline{\sigma_t}$ be the interpretation of V_M given by $\overline{\sigma_t}(\pi) \equiv w_\pi\sigma_t$. Since $var(w_\pi\sigma_t) = \emptyset$, for every $\pi \in V_M$, the interpretation $\overline{\sigma_t}$ is total.

Theorem 2. *Algorithm 1 terminates, and when it does, the relation $\sim_{\overline{\sigma_t}}$ is the unique coarsest bisimulation equivalence on M.*

4.2 Complexity

We now analyse the asymptotic running time of Algorithm 1. We use the parameter r to capture the amount of nondeterminism in M. It is defined as $r = max_{q\in Q, f\in\Sigma} |\delta_f(q)|$. In particular, $r \leq 1$ whenever M is deterministic.

Lemma 4. *The function* equiv *is called at most once for each pair of states.*

Let us denote the *union* of all w_π, $\pi \in V_M$, i.e. that appear as right-hand sides in w_M, by rhs_M. In the update of σ_i on Line 8, 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 therefor represent $rhs_M\sigma_i$ as a directed acyclic graph (DAG) and allow node sharing between formulas. In the initial DAG, only nodes representing variables and the logical constants $\top$ and $\bot$ are shared, but as the algorithm proceeds, greater parts of the graph come to overlap. The construction is straight-forward but rather technical, so readers that are satisfied with a high-level view may want to continue to Theorem 3.

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

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

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

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

where $root(D(w \otimes w')) = v$, 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 $D(w_\pi)$, $w_\pi \in rhs_M$, and merging

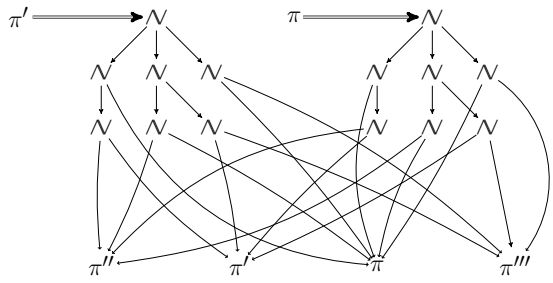

Fig. 1. Sketch of DAG $D(rhs_M\sigma_i)$, corresponding to $w_{\pi'}$ and w_π parts; references ref_{rhs} are depicted by double-lined arrows. $\curlywedge$ denotes a node labeled by either $\wedge$ or $\vee$.

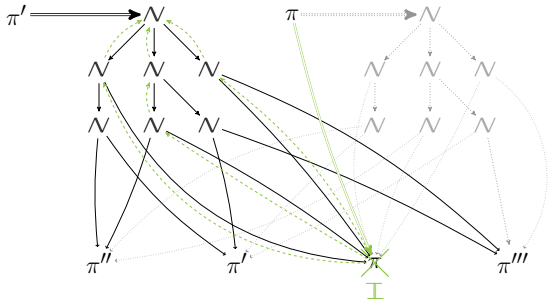

Fig. 2. Sketch of DAG $D(rhs_M\sigma_i)$, corresponding to $w_{\pi'}$ and w_π parts, case where π gets resolved to either $\top$ or $\bot$ (depicted by $\perp\!\!\!\top$). $\curlywedge$ denotes a node labeled by either $\wedge$ or $\vee$. The update to w_π for the aforementioned case is depicted in green (new edges, node relabelling, and propagation) and gray (removed edges and nodes).

all leaf nodes that have identical labels. Thus, for each state pair π 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 π, we have a reference $ref_{rhs}(\pi)$ to w_π, i.e. to the corresponding right hand side representation in the DAG. Part of the initial DAG is depicted in Figure 1.

During the computation, graph $D(rhs_M\sigma_i)$ is reorganised by changing the targets of certain edges, but the size of $D(rhs_M\sigma_i)$ never grows, apart from 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 $ref_{rhs}(\pi)$'s will continue to point at $w_\pi\sigma_i$, expression $w_\pi\sigma_i$ changes to reflect the latest σ_i, and will be simplified as much as possible.

There are two kinds of updates to a variable that can occur in our algorithm. The first is the resolution of $w_\pi\sigma$ to either $\top$ or $\bot$, which happens during ini-

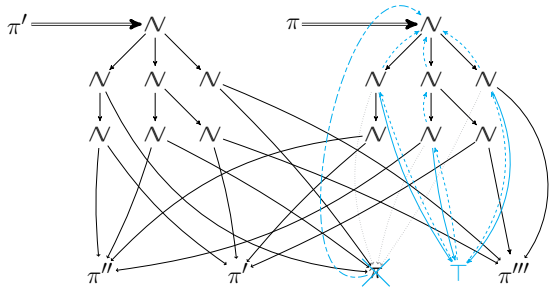

Fig. 3. Sketch of DAG $D(rhs_M\sigma_i)$, corresponding to $w_{\pi'}$ and w_π parts, case where π gets resolved to either $\top$ or $\bot$ (depicted by $\overline{\top}$). $\curlywedge$ denotes a node labeled by either $\wedge$ or $\vee$. Cyan indicates new edges in w_π for the aforementioned case, as well as the following upward propagation to simplify $w_\pi\sigma_{i+1}$; leftmost oddly dashed arrow depicts the replacement of leaf node π's label by a reference to the simplified $w_\pi\sigma_{i+1}$. Removed edges in $w_\pi\sigma_{i+1}$ are depicted dotted in gray. Note that if upward propagation reaches the root of the DAG part representing $w_\pi\sigma_{i+1}$, propagation continues upwards through the DAG part for $w'_\pi\sigma_{i+1}$, similar to Figure 2.

tialisation of σ on Line 2 of Algorithm 1, on Line 6 of Algorithm 2, and possibly on Line 8 of that algorithm as well. In this case, as sketched in Figure 2, three things must happen to our graph $D(rhs_M\sigma_i)$ to ensure that it reflects the updated $w_\pi\sigma_{i+1}$: Firstly, formula $w_\pi\sigma_i$ in $D(rhs_M\sigma_i)$ must be replaced by $\top$ or $\bot$ as the case may be. Thus, graph $D(rhs_M\sigma_i)$ is modified to remove the nodes and edges of such $w_\pi\sigma_i$ (barring the shared leaf nodes for elements of V_M). Secondly, the unique shared leaf node representing π in the DAG must be re-labeled to $\top$ or $\bot$. Thirdly, this re-labeling must be propagated upwards along each DAG branch leading to this node, now labeled $\top$ respectively $\bot$, as this resolution of π 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 $\wedge$, it can be resolved to (i.e. its subtree replaced by a reference to) $\bot$; with parent $\vee$, simplification is possible, i.e. getting rid of a child; in the case of $\top$ and parent $\vee$, it can be resolved to $\top$; with parent $\wedge$, simplification is possible, i.e. getting rid of a child; and so on upwards until no more change is made.

The second kind of update, sketched in Figure 3, is that corresponding to Line 8 of Algorithm 2 extending σ_i by a substitution for π by $w_\pi\sigma_i\{\pi \leftarrow \top\}$ in case the latter does *not* correspond to $\bot$ or $\top$. Again, a number of things must happen to $D(rhs_M\sigma_i)$. Firstly, references in $w_\pi\sigma_i$ to the unique shared leaf node for π itself must be replaced by references to $\top$. Secondly, this change must be propagated upwards along each DAG branch leading to this reference to $\top$, as this local resolution of π 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 $w_\pi\sigma_{i+1}$ may either resolve to $\top$, or still be a proper tree. In the first case,

the unique shared leaf node representing π in the DAG must be re-labeled to $\top$, and this change propagated upwards, as per steps two and three of the first kind of update, described in the previous paragraph. In the second case, the unique node π that may still be used in right-hand sides other than $w_\pi\sigma_{i+1}$, needs to have its label removed and replaced by a *reference* to the modified $w_\pi\sigma_{i+1}$.

Theorem 3 (Complexity). *Algorithm 1 is in* $O\big(n^2r^2\,|\Sigma|\big)$.

Recall that bisimulation minimization coincides with classical minimization in the case of DFA. Since $r \leq 1$ for such devices, the run time of our algorithm is comparable to that of the algorithm in [21].

4.3 Lazy evaluation

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*), the status of all pairs on which *equiv* has been called is known.

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

Lemma 5. *Throughout the computation, the following invariants hold: Firstly,* $var(\sigma_i) \subseteq S$. *Secondly, from the point the function* equiv *has reached Line 5 on the pair* π, *we have* $var(w_\pi\sigma_i) \subseteq S$ *for every subsequent* σ_i.

Theorem 4. *Every time the process control returns to* minimize, *every pair* π *on which equiv has been called is resolved.*

5 Conclusion

We have presented a minimization algorithm for NFA that identifies and merges bisimulation-equivalent states. In terms of running time, it is as efficient as any existing aggregation-based minimisation algorithm for DFA, but less efficient than current refinement-based minimisation algorithms for NFA. However, compared to the latter group, it has the advantage that intermediate solutions are usable for language-preserving reduction of the input automaton M. Thus, implementations can be interrupted in time-constrained settings, to save memory by transforming the input to a reduced, not necessarily minimal, equivalent automaton.

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, e.g., nondeterministic graph automata could be considered.

Certain highly connected automata might lead the algorithm to only converge in the final iteration. As future work, an empirical investigation of its behaviour on various automata would also be interesting.

References

1. P. A. Abdulla, L. Holík, L. Kaati, and T. Vojnar. A uniform (bi-)simulation-based framework for reducing tree automata. *Electronic Notes in Theoretical Computer Science*, 251(0):27 – 48, 2009.
2. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, Reading, Massachusetts, 1974.
3. M. Almeida, N. Moreira, and R. Reis. Incremental DFA minimisation. *RAIRO – Theoretical Informatics and Applications*, 48(2):173–186, 2014.
4. J. Björklund, A. Maletti, and J. May. Backward and forward bisimulation minimization of tree automata. *Theoretical Comp. Sci.*, 410(37):3539–3552, 2009.
5. J. Björklund, A. Maletti, and H. Vogler. Bisimulation minimisation of weighted automata on unranked trees. *Fundamenta Informatica*, 92(1-2):103–130, 2009.
6. P. Buchholz. Bisimulation relations for weighted automata. *Theoretical Computer Science*, 393(13):109 – 123, 2008.
7. L. Cleophas, D. G. Kourie, T. Strauss, and B. W. Watson. On minimizing deterministic tree automata. In J. Holub and J. Žďárek, editors, *Prague Stringology Conference, Prague, Czech Republic, 2009*, pages 173–182, 2009.
8. J. Daciuk. *Optimization of Automata*. Gdańsk University of Technology Publishing House, 2014.
9. G. Gramlich and G. Schnitger. Minimizing NFA's and regular expressions. *Journal of Computer and System Sciences*, 73(6):908–923, Sept. 2007.
10. J. E. Hopcroft. An n log n algorithm for minimizing the states in a finite automaton. In Z. Kohavi, editor, *The Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.
11. J. E. Hopcroft and J. D. Ullman. Set merging algorithms. *SIAM Journal on Computing*, 2(4):294–303, 1973.
12. D. A. Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 257:161–190 and 275–303, 1954.
13. A. Maletti. Minimizing weighted tree grammars using simulation. In *Finite-State Methods and Natural Language Processing, Pretoria, South Africa, 2009*, pages 56–68, Berlin, Heidelberg, 2010. Springer.
14. A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *13th Annual IEEE Symposium on Switching and Automata Theory*, pages 125–129, 1972.
15. E. F. Moore. Gedanken-experiments on sequential machines. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, Princeton, New Jersey, 1956.
16. A. Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.
17. R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.
18. R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.
19. H. ten Eikelder. Some algorithms to decide the equivalence of recursive types. Technical Report 93/32, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, 1991.
20. B. W. Watson. *Taxonomies and Toolkits of Regular Language Algorithms*. PhD thesis, Dept. of Mathematics and Comp. Sci., TU Eindhoven, 1995.
21. B. W. Watson and J. Daciuk. An efficient incremental DFA minimization algorithm. *Natural Language Engineering*, 9(1):49–64, 2003.

A Appendix

A.1 Lemma 6

Lemma 6. *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 π such that $(v \vee v')(\pi \rightarrow w_\pi)$ is false. This can only happen if $(v \vee v')(\pi) = \top$ but $(v \vee v')(w_\pi) \equiv \bot$. However, we know that $v(\pi) = \top$ or $v'(\pi) = \top$. Assume the former, without loss of generality. Then $v(w_\pi) = \top$ since $v \in \mathrm{Mod}(w_M)$. Now, that fact that more variables are assigned the value $\top$ in $v \vee v'$ cannot cause w_π to become false, since it is negation-free. Hence $(v \vee v')(w_\pi) \equiv \top$ too, which gives us a contradiction. □

A.2 Lemma 1

Proof. If v cannot be extended to a model for w_M then the statement is trivially true. If it can be extended to a model, then by Lemma 6 the union of all such extensions is a model for w_M. □

A.3 Lemma 2

Proof. Since v is reflexive, $v'(\langle p, p\rangle) = \top$ for every $p \in V$, 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 V_M. By taking the element-wise 'or' 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. Assume that the extension v' of v is such that $v'(\langle p, q\rangle) = v'(\langle q, r\rangle) = \top$. Using q as a bridge, it can be shown that $v'(w_{\langle p,r\rangle}) \equiv \top$, so v' can be extended by assigning $\langle p, r\rangle$ the value 'true'. Since $\hat{v}$ is already maximal, it has to be transitive. □

A.4 Lemma 3

Proof. We note that $\sim_{v_0} = \mathcal{I}_{V_M}$, so by construction, $v_0(w_M) \equiv \top$. Furthermore, $\mathcal{I}_{V_M}$ is clearly an equivalence relation, namely the finest one in which each state is an equivalence class of its own. □

A.5 Theorem 1

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

A.6 Lemma 7

Lemma 7. *At every point of the computation,* $var(w_\pi\sigma_i) \cap dom(\sigma_i) = \emptyset$*, for every* $\pi \in V_M$*.*

Proof. We prove the invariant by induction. Lemma 7 is trivially true after the initialisation of σ_0 in Algorithm 1.

Consider the assignment to σ_{i+1} on Line 6 of Algorithm 2. Due to Line 5 we know that $w_\pi\sigma_i \in \{\top, \bot\}$. Assume without loss of generality that $w_\pi\sigma_i = \top$. By induction, $var(w_{\pi'}\sigma_i) \cap dom(\sigma_i) = \emptyset$ for every $\pi' \in V_M$, so $var(w_{\pi'}\sigma_i\{\pi \leftarrow \top\}) \cap (dom(\sigma_i) \cup \{\pi\}) = \emptyset$.

Finally, consider the assignment to σ_{i+1} at the end of Algorithm 2. By the induction hypothesis, $var(w_\pi\sigma_i) \cap dom(\sigma_i) = \emptyset$. Since $\pi \notin var(w_\pi\sigma_i\{\pi \leftarrow \top\})$, it follows that $\pi \notin var(w_{\pi'}\sigma_i\{\pi \leftarrow w_\pi\sigma_i\{\pi \leftarrow \top\}\})$ for every $\pi' \in V_M$, so σ_i can safely be updated to $\sigma_i\{\pi \leftarrow w_\pi\sigma_i\{\pi \leftarrow \top\}\}$ without invalidating Lemma 7. □

A.7 Lemma 8

Lemma 8. *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. Looking at lines $5 - 9$, each call to *equiv* terminates with $dom(\sigma_i)$ greater than before, hence the number of calls of the while-loop in Algorithm 1 is also finite. □

A.8 Lemma 9

Lemma 9. *Throughout the execution of Algorithm 1,* $\hat{v}(v_0(w_M)) = \hat{v}(w_M\sigma_i)$*.*

Proof. The proof is by induction on σ_i. By construction, $v_0(w_M) \equiv w_M\sigma_0$, so $\hat{v}(v_0(w_M)) = \hat{v}(w_M\sigma_i)$.

In the main body of the algorithm, σ_i is updated on Line 6 and Line 8.

Consider first the assignment on Line 6. Since the value of $w_\pi\sigma_i$ is resolved, there are two cases:

- In the first, $w_\pi\sigma_i = \bot$. By the induction hypothesis, $\hat{v}(v_0(v_M)) = \hat{v}(w_M\sigma_i)$, so since there is no extension of $w_M\sigma_i$ that sets π to $\top$, there is no extension v of $w_M\sigma_i$ that does so either. If there were, then $\hat{v}(v_0(v_M)) \cup v \in \mathrm{Ext}(v_0(v_M))$, which contradicts the maximality of $\hat{v}(v_0(v_M))$. Hence, $\hat{v}(v_0(w_M)) = \hat{v}(w_M\sigma_{i+1})$.
- In the second, $w_\pi\sigma_i = \top$. Since there is at least one extension of $w_\pi\sigma_i$ in which π is assigned $\top$, the same must hold for $v_0(w_M)$, so by Lemma 6 and Lemma 1, π is assigned $\top$ in $\hat{v}(v_0(w_M)) = \hat{v}(w_M\sigma_{i+1})$.

Consider then the assignment on Line 8. We divide it into two steps. First, we note that for every $\pi \in V_M$, $\pi \to w_\pi \equiv \pi \to (w_\pi\{\pi \leftarrow \top\})$, and that $\pi \notin var(w_\pi\{\pi \leftarrow \top\})$. Furthermore, if $\pi \notin var(w_\pi)$, then $w_M \equiv w_M\{\pi \leftarrow w_\pi\}$. Hence, $w_M\sigma_i \equiv w_M\sigma_{i+1}$. □

A.9 Lemma 4

Proof. When function *equiv* is called with argument π, it adds π to S, which prevents it from being used as an argument again when the algorithm makes its recursive calls in the while loop. After the while loop, the substitution σ_i is extended to have π in its domain, and this prevents π from ever being used as an argument to *equiv* again. □

A.10 Theorem 3

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.

Let us look closer at the body of Algorithm 2. To satisfy the existence clause in the while loop of Algorithm 2, we traverse down the left-most path of $D(rhs_M\sigma_i)$, and this can be done in $O(\log n^2)$. Since we keep the DAG $D(rhs_M\sigma_i)$ in a simplified form, the if clause on Line 5 can be decided in constant time.

The update to σ_i on Line 6 is where the majority of the work is done. Here, we start at the node labelled π and follow all edges in $D(rhs_M\sigma_i)$ backwards, updating the graph structure to reflect the truth value of π. Everytime we follow and 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 6 is in $O(|D(rhs_M\sigma_i)|) = O(n^2r^2\,|\Sigma|)$ when summed over every call to Algorithm 2.

Local substitution $w_\pi\sigma_i\{\pi \leftarrow \top\}$ on Line 8 requires $O(r^2\,|\Sigma|)$ steps, and the update of σ_i to σ_{i+1} only takes constant time, requiring rerouting of one pointer.

To summarize, the complexity of Algorithm 2 is

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

which is in $O(n^2r^2\,|\Sigma|)$ if $\log n^2 \leq r^2\,|\Sigma|$. □

A.11 Lemma 5

Proof. (Sketch) The proof is by induction. When *equiv* is invoked the first time, $var(\sigma_0) = \emptyset$ and *equiv* has not reached Line 5 for any π, so both invariants are trivially true.

We consider the effect Line 5, 6, and 8 have on Invariants 1 and 2.

Line 5 only affects Invariant 2. Consider then the case when *equiv* reaches Line 5 for a variable π. Every π' in $var(w_\pi\sigma_i)$ has two possible origins;

- either π' was left because $\pi' \in S$, so Invariant 2 is trivially true,
- or $\pi' \in var(\sigma_i(\pi''))$ for some $\pi'' \in var(w_\pi) \setminus S$, and by the induction hypothesis, Invariant 2 is true.

Line 6 affects both invariants. The state is such that $\pi \in S$, but as the function is about to exit, π will be taken out of S. The assignment to σ_{i+1} replaces π by a constant, so $\pi \notin var(\sigma_i)$. In combination with the induction hypothesis,

Invariant 1 holds. Using the propagation technique described in Section 4.2, π is also removed from $w_{\pi'}$ for every previously called pair π', so Invariant 2 holds.

Line 8 also affects both invariants. Again, the algorithm is about to remove π from S. The application $\{\pi \leftarrow \top\}$ to σ_i in combination with the induction hypotheses *for both invariants* means that $\pi \notin var(\sigma_{i+1})$, so Invariant 1 holds. Propagation to $w_{\pi'}$ for every previously pair guarantees Invariant 2. □

A.12 Theorem 4

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. Due to Invariant 2, $var(w_\pi \sigma_i) = \emptyset$, so $w_\pi \sigma_i$ is resolved. □