Backward and Forward Bisimulation Minimization of Tree Automata [⋆]

Johanna Högberg

Department of Computing Science, Umeå University
SE 90187 Umeå, Sweden

Andreas Maletti [2,*]

International Computer Science Institute
1947 Center Street, Suite 600, Berkeley, CA 94704, USA

Jonathan May [3]

Information Sciences Institute, University of Southern California
Marina Del Rey, CA 90292, USA

Abstract

We improve on an existing [P.A. Abdulla, J. Högberg, L. Kaati. *Bisimulation Minimization of Tree Automata.* Int. J. Found. Comput. Sci. 18(4): 699–713, 2007] bisimulation minimization algorithm for finite-state tree automata by introducing backward and forward bisimulation and developing minimization algorithms for them. Minimization via forward bisimulation is also effective on deterministic tree automata, faster than the previous algorithm, and yields the minimal equivalent deterministic tree automaton. Minimization via backward bisimulation generalizes the previous algorithm and can yield smaller automata but is just as fast. We demonstrate implementations of these algorithms on a typical task in natural language processing.

Key words: tree automaton, bisimulation, minimization, partition refinement, tree language
2000 MSC: 68Q45, 68Q25

⋆ This is an extended version of: J. Högberg, A. Maletti, J. May. *Backward and Forward Bisimulation Minimization of Tree Automata.* Proc. 12th Int. Conf. Implementation and Application of Automata, LNCS 4783, p. 109–121, Springer Verlag, 2007.
* Corresponding author.
Email addresses: johanna@cs.umu.se (Johanna Högberg), maletti@icsi.berkeley.edu (Andreas Maletti), jonmay@isi.edu (Jonathan May).
[1] Supported by Vinnova's program Vinnmer.
[2] On leave from *Technische Universität Dresden, Faculty of Computer Science, 01062 Dresden, Germany* with support by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD).
[3] Partially supported by NSF grant IIS-0428020.

1. Introduction

Automata minimization has a long and well-studied history. For deterministic finite (string) automata (dfa) efficient algorithms exist. The renowned algorithm by Hopcroft [1] runs in time $O(m \log n)$ where m and n are, respectively, the number of transitions and states of the input automaton. The situation is worse for general finite-state automata (fsa), which include non-deterministic finite automata (nfa). The minimization problem for nfa is PSPACE-complete [2] and cannot even be efficiently approximated within the factor $o(n)$ unless P = PSPACE [3–5]. The problem must thus be restricted to allow algorithms of practical value, and one possibility is to settle for a partial minimization. This was done in [6] for *finite-state tree automata* (fta), which are a generalization of fsa that recognize tree languages and are used in applications such as model checking [7] and natural language processing [8].

The minimization algorithm in [6] was inspired by a partitioning algorithm due to Paige and Tarjan [9], and relies heavily on *bisimulation*; a concept introduced by Milner as a formal tool for investigating transition systems. Intuitively, two states are bisimilar if they can simulate each other, or equivalently, the observable behavior of the two states coincides. Depending on the capacity of the observer, we obtain different types of bisimulation. In all cases we assume that the observer has the capacity to inspect the final reaction to a given input (i.e., the acceptance or rejection of a given tree). The presence of bisimilar states in an automaton indicates redundancy. Thus, identifying bisimilar states allows us to reduce the size of the input automaton, but we are not guaranteed to obtain the smallest possible automaton. In this work we extend the approach of [6] in two ways: (i) we relax the constraints for state equivalence, and (ii) we introduce a new bisimulation relation that can be applied to deterministic (bottom-up) tree automata (dta) [10] with practical effects. As [6], the only previous known minimization algorithm for tree automata, is ineffective on dta and no more effective on fta than the extensions presented in this work, we are able to obtain smaller automata than previously possible.

The two extensions correspond, respectively, to two types of bisimulation: *backward* and *forward* bisimulation [11]. In a forward bisimulation on an automaton M, bisimilar states are restricted to have identical futures (i.e., the observer can inspect what will happen next). The future of a state q is the set of *contexts* (i.e., trees in which there is a unique leaf labeled by the special symbol $\square$) that would be recognized by M, if the (bottom-up) computation starts with the state q at the unique $\square$-labeled node in the context. By contrast, backward bisimulation uses a local condition on the transitions to enforce that the past of any two bisimilar states is equal (i.e. the observer can inspect what has already happened). The past of a state q is the language that would be recognized by the automaton if q were its only final state (cf., left and right congruence [12] for string languages).

Both types of bisimulation yield efficient minimization procedures, which can be applied to arbitrary fta. Further, forward bisimulation minimization is useful on dta. It computes the unique minimal dta recognizing the same language as the input dta (see Theorem 4.7). More importantly, it is shown in Theorem 4.12 that the asymptotic time complexity of our minimization algorithm is $O(rm \log n)$, where r is the maximal rank of the symbols in the input alphabet and m and n are respectively the number of transitions and states of the input automaton. Thus, our algorithm supersedes the currently best minimization algorithm [10] for dta, whose complexity is $O(rmn)$, and coincides with the Hopcroft-algorithm on dfa ($r = 1$ for dfa). Backward bisimulation, though slightly harder to compute, has great practical value as well. Our backward bisimulation is weaker than the bisimulation of [6]. Consequently, the fta obtained by our backward bisimulation minimization algorithm will have at most as many states as the fta obtained by the minimization algorithm of [6]. In addition, the asymptotic time-complexity of our algorithm (see Theorem 3.22), which is $O\big(r^2\, m \log n\big)$, is the same as the one for the minimization algorithm of [6]. Note that in [6] the run time $O(rm' \log n)$ is reported with $m' = rm$.

Finally, there are advantages that support having two types of bisimulation. First, forward and backward bisimulation minimization only yield fta that are minimal with respect to the respective type of bisimulation. Thus applying forward and backward bisimulation minimization in an alternating fashion commonly yields even smaller fta (see Sect. 5). Recently, [13] considered our backward bisimulation minimization followed by one variant of forward bisimulation minimization, which is called "composed bisimulation". Second, in certain domains only one type of bisimulation minimization is effective. For example, backward bisimulation minimization is ineffective on dta because no two different states of a dta have the same past.

Including this introduction, the paper has 6 sections. In Sect. 2, we define basic notions and notations. We then proceed with backward bisimulation and the minimization algorithm based on it. In Sect. 4, we consider forward bisimulation. Finally, in Sect. 5 we demonstrate our algorithms on a typical task in natural language processing and conclude in Sect. 6.

2. Preliminaries

We write $\mathbb{N}$ to denote the set of natural numbers including zero. The set $\{i \mid k \leq i \leq n\}$ is abbreviated to $[k, n]$, the cardinality of a set S is denoted by $|S|$, and the subtraction of elements of set B from set S is denoted by $S \setminus B$. We abbreviate $Q \times Q$ to Q^2 and write $q_1 \cdots q_k \in D_1 \cdots D_k$ instead of $(q_1, \ldots, q_k) \in D_1 \times \cdots \times D_k$.

Let $\mathcal{R}$ and $\mathcal{P}$ be equivalence relations on S. We say that $\mathcal{R}$ is *coarser* than $\mathcal{P}$ (or equivalently: $\mathcal{P}$ is a *refinement* of $\mathcal{R}$), if $\mathcal{P} \subseteq \mathcal{R}$. The *equivalence class* (or *block*) of $s \in S$ with respect to $\mathcal{R}$ is $[s]_{\mathcal{R}} = \{s' \mid (s, s') \in \mathcal{R}\}$. Whenever $\mathcal{R}$ is obvious from the context, we simply write $[s]$ instead of $[s]_{\mathcal{R}}$. It should be clear that $[s]$ and $[s']$ are equal if $(s, s') \in \mathcal{R}$ and disjoint otherwise, so $\mathcal{R}$ induces a partition $(S/\mathcal{R}) = \{[s] \mid s \in S\}$ of S.

A *ranked alphabet* is a finite set of symbols $\Sigma = \bigcup_{k \in \mathbb{N}} \Sigma_{(k)}$ that is partitioned into pairwise disjoint subsets $\Sigma_{(k)}$. The set T_Σ of *trees over* Σ is the smallest language over Σ such that $f\, t_1 \cdots t_k \in T_\Sigma$ for every $f \in \Sigma_{(k)}$ and all $t_1, \ldots, t_k \in T_\Sigma$. To improve readability we write $f[t_1, \ldots, t_k]$ instead of $f\, t_1 \cdots t_k$ unless $k = 0$. Any subset of T_Σ is called a *tree language*.

By $\Sigma(Q)$ we denote the set $\{f(q_1, \ldots, q_k) \mid f \in \Sigma_{(k)}, q_1, \ldots, q_k \in Q\}$ for every ranked alphabet Σ and set Q. A *finite-state tree automaton* (for short: fta) is a tuple $M = (Q, \Sigma, \delta, F)$ where Q is a finite set of *states*, Σ is a ranked alphabet, and δ is a finite set of *transitions* of the form $w \to q$ for some $w \in \Sigma(Q)$ and $q \in Q$. We call an *fta deterministic* and (*complete*) if for every $w \in \Sigma(Q)$ there exists exactly one $q \in Q$ such that $w \to q \in \delta$. Finally, $F \subseteq Q$ is a set of *accepting* states. To indicate that a transition $w \to q$ is in δ, we write $w \xrightarrow{\delta} q$. In the obvious way, δ extends to trees yielding a mapping $\delta\colon T_\Sigma \to \mathfrak{P}(Q)$; i.e., $\delta(t) = \{q \mid f(q_1, \ldots, q_k) \xrightarrow{\delta} q \text{ and } q_i \in \delta(t_i) \text{ for all } i \in [1, k]\}$ for $t = f[t_1, \ldots, t_k]$ in T_Σ. For every $q \in Q$ we denote $\{t \in T_\Sigma \mid q \in \delta(t)\}$ by $\mathcal{L}(M)_q$. The tree language *recognized* by M is $\mathcal{L}(M) = \bigcup_{q \in F} \mathcal{L}(M)_q$. Two fta M_1 and M_2 are *equivalent* if $\mathcal{L}(M_1) = \mathcal{L}(M_2)$. Finally, we say that a state q in Q is *useless* if $\mathcal{L}(M)_q = \emptyset$. For every fta M we can construct an equivalent fta without useless states in time $O(m)$ where $m = |\delta|$ is the number of transitions of M.

Let $\mathcal{R}$ be an equivalence relation on Q. The *aggregated fta (with respect to M and $\mathcal{R}$)*, denoted by $(M/\mathcal{R})$, is the fta $((Q/\mathcal{R}), \Sigma, \delta', F')$ given by $F' = \{[q] \mid q \in F\}$ and

$$\delta' = \{f([q_1], \ldots, [q_k]) \to [q] \mid f(q_1, \ldots, q_k) \xrightarrow{\delta} q\} \enspace .$$

Note that, in general, $\bigcup_{p \in [q]} \mathcal{L}(M)_p \subseteq \mathcal{L}((M/\mathcal{R}))_{[q]}$ for every $q \in Q$.

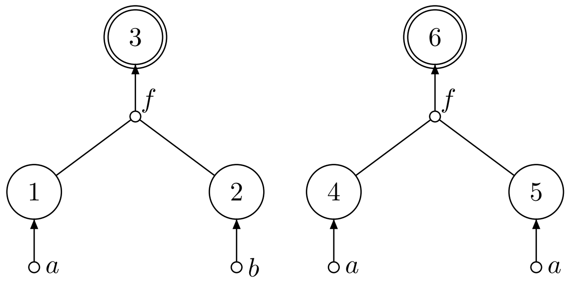

Fig. 1. Example tree automaton N of Example 3.2.

3. Backward Bisimulation

3.1. *Foundation*

We first introduce the notion of *backward bisimulation* for a fta M. This type of bisimulation requires bisimilar states to recognize the same tree language, but it is irrelevant whether the states are final states or not. Clearly, the presence of two backward bisimilar states indicates a redundancy. For the rest of this section, let $M = (Q, \Sigma, \delta, F)$ be a fta.

Definition 3.1 (cf. [11, Definition 4.1]) *An equivalence relation $\mathcal{R}$ on Q is a* backward bisimulation *on M if $f(p_1, \ldots, p_k) \xrightarrow{\delta} p$ implies that for every $i \in [1, k]$ there exists $q_i \in [p_i]$ such that $f(q_1, \ldots, q_k) \xrightarrow{\delta} q$ for every $(p, q) \in \mathcal{R}$, symbol f of $\Sigma_{(k)}$, and sequence $p_1, \ldots, p_k \in Q$.*

Note that in the special case of a nullary symbol $f \in \Sigma_{(0)}$, we obtain that $f() \xrightarrow{\delta} p$ implies $f() \xrightarrow{\delta} q$ for every $(p, q) \in \mathcal{R}$. Let us illustrate backward bisimulation on an example.

Example 3.2 Let $\Sigma = \Sigma_{(2)} \cup \Sigma_{(0)}$ be the ranked alphabet with $\Sigma_{(2)} = \{f\}$ and $\Sigma_{(0)} = \{a, b\}$. We want to recognize the tree language $L = \{f[a, b], f[a, a]\}$. To this end, we first construct fta N_1 and N_2 that recognize $\{f[a, b]\}$ and $\{f[a, a]\}$, respectively. Then we construct N by disjoint union of N_1 and N_2. We obtain the fta $N = ([1, 6], \Sigma, \delta, \{3, 6\})$, which is illustrated in Fig. 1, with

$$a() \xrightarrow{\delta} 1 \qquad b() \xrightarrow{\delta} 2 \qquad f(1, 2) \xrightarrow{\delta} 3 \qquad a() \xrightarrow{\delta} 4 \qquad a() \xrightarrow{\delta} 5 \qquad f(4, 5) \xrightarrow{\delta} 6 \ .$$

Let $\mathcal{P}$ be the equivalence induced by the partition $\{\{1, 4, 5\}, \{2\}, \{3\}, \{6\}\}$. In fact, $\mathcal{P}$ is a backward bisimulation on N. In order to justify this claim, we only need to check the transitions leading to 1, 4, or 5. Trivially, the condition of Definition 3.1 is met for such transitions because $a() \to q$ is in δ and $b() \to q$ is not in δ for every state $q \in \{1, 4, 5\}$.

The aggregated fta $(N/\mathcal{P})$ is $(Q', \Sigma, \delta', F')$ where $Q' = \{[1], [2], [3], [6]\}$, $F' = \{[3], [6]\}$, and

$$a() \xrightarrow{\delta'} [1] \qquad b() \xrightarrow{\delta'} [2] \qquad f([1], [2]) \xrightarrow{\delta'} [3] \qquad f([1], [1]) \xrightarrow{\delta'} [6] \ .$$

We display $(N/\mathcal{P})$ in Fig. 2. □

Next, we show that, for every backward bisimulation $\mathcal{R}$ on M, the fta M and $(M/\mathcal{R})$ are equivalent. We prepare this statement with a key lemma, which states that the state q of M and the state $[q]$ of $(M/\mathcal{R})$ recognize the same tree language. For the rest of this section, let $\mathcal{R}$ be a backward bisimulation on M.

Lemma 3.3 (cf. [11, Theorem 4.2] and [14, Lemma 8]) *For every state q of M we have $\mathcal{L}((M/\mathcal{R}))_{[q]} = \mathcal{L}(M)_q$.*

PROOF. Let $(M/\mathcal{R}) = (Q', \Sigma, \delta', F')$. We already remarked that $\mathcal{L}(M)_q \subseteq \mathcal{L}((M/\mathcal{R}))_{[q]}$ holds in general. We prove the remaining direction for every $t \in T_\Sigma$ by induction. Suppose that $t \in \mathcal{L}((M/\mathcal{R}))_{[q]}$ with $t = f[t_1, \ldots, t_k]$ for some $f \in \Sigma_{(k)}$ and $t_1, \ldots, t_k \in T_\Sigma$. Then $[q] \in \delta'(t)$ and

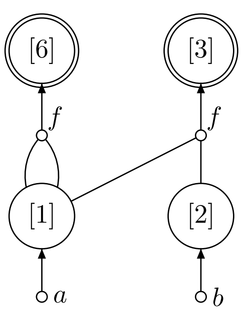

Fig. 2. Aggregated tree automaton $(N/\mathcal{P})$ of Example 3.2.

thus there exist $D_1, \ldots, D_k \in Q'$ such that $f(D_1, \ldots, D_k) \xrightarrow{\delta'} [q]$ and $D_i \in \delta'(t_i)$ for every $i \in [1, k]$. By definition of $(M/\mathcal{R})$, there exist $p, p_1, \ldots, p_k \in Q$ such that $f(p_1, \ldots, p_k) \xrightarrow{\delta} p$, $p \in [q]$, and $p_i \in D_i$ for every $i \in [1, k]$. With the help of Definition 3.1, we conclude that there also exist $q_1, \ldots, q_k \in Q$ such that $f(q_1, \ldots, q_k) \xrightarrow{\delta} q$ and $q_i \in D_i$ for every $i \in [1, k]$ because $(p, q) \in \mathcal{R}$. Finally, by the induction hypothesis, we have $t_i \in \mathcal{L}(M)_{q_i}$ and consequently $q_i \in \delta(t_i)$ for every $i \in [1, k]$ because $t_i \in \mathcal{L}((M/\mathcal{R}))_{[q_i]}$. This yields $q \in \delta(t)$ and $t \in \mathcal{L}(M)_q$ as desired. □

Clearly, the previous lemma shows that backward bisimilar states in M recognize the same tree language. Moreover, we can now show that $(M/\mathcal{R})$ recognizes exactly $\mathcal{L}(M)$.

Corollary 3.4 (cf. [11, Theorem 4.2] and [14, Theorem 2]) $\mathcal{L}((M/\mathcal{R})) = \mathcal{L}(M)$.

PROOF. A tree t is in $\mathcal{L}((M/\mathcal{R}))$ if and only if there exists a state q of M such that $q \in F$ and $t \in \mathcal{L}((M/\mathcal{R}))_{[q]}$. By Lemma 3.3, the latter holds precisely when $t \in \mathcal{L}(M)_q$. Consequently, $t \in \mathcal{L}((M/\mathcal{R}))$ if and only if $t \in \mathcal{L}(M)$. □

Clearly, among all backward bisimulations on M, the coarsest one yields the smallest aggregated fta. Next we show that this smallest aggregated fta admits only the trivial (i.e., identity) backward bisimulation, and thus, cannot be further minimized with the help of backward bisimulations. An fta that admits only the identity as backward bisimulation is called *backward bisimulation minimal.* We now prove that the coarsest backward bisimulation $\mathcal{P}$ on M yields a backward bisimulation minimal fta $(M/\mathcal{P})$ that is equivalent to M.

Theorem 3.5 *For every fta M there exists a coarsest backward bisimulation $\mathcal{P}$ on M, and $(M/\mathcal{P})$ is an equivalent backward bisimulation minimal fta.* □

PROOF. Suppose that a coarsest backward bisimulation $\mathcal{P}$ on M exists. Then any two states that are bisimilar in $(M/\mathcal{P})$, are already bisimilar in M, which together with Theorem 3.4 proves the second part of the statement.

Let $\mathcal{R}$ and $\mathcal{P}$ be backward bisimulations on M. Then there exists a backward bisimulation $\mathcal{R}'$ on M such that $\mathcal{R} \cup \mathcal{P} \subseteq \mathcal{R}'$. From this statement, the existence of the coarsest backward bisimulation easily follows. Let $\mathcal{R}'$ be the smallest equivalence containing $\mathcal{R} \cup \mathcal{P}$. We now show that $\mathcal{R}'$ is a backward bisimulation. Let $(p, q) \in \mathcal{R}'$. Thus there exist $n \in \mathbb{N}$ and

$$(p_1, p_2), (p_2, p_3), \ldots, (p_{n-2}, p_{n-1}), (p_{n-1}, p_n) \in \mathcal{R} \cup \mathcal{P}$$

such that $p_1 = p$ and $p_n = q$. Clearly, every block $D \in (Q/\mathcal{R}')$ is a union of blocks of $(Q/\mathcal{R})$ as well as a union of blocks of $(Q/\mathcal{P})$. Let $f \in \Sigma_{(k)}$ and $q_1, \ldots, q_k \in Q$ be such that $f(q_1, \ldots, q_k) \xrightarrow{\delta} p_1$. We prove by induction that for every $m \in [1, n]$ and for every $i \in [1, k]$ there exists $q_i' \in [q_i]_{\mathcal{R}'}$ such that $f(q_1', \ldots, q_k') \xrightarrow{\delta} p_m$. For $m = 1$, this is trivial. Now let $1 < m \leq n$. By induction hypothesis, for every $i \in [1, k]$ there exists $q'_i \in [q_i]_{\mathcal{R}'}$ such that $f(q'_1, \ldots, q'_k) \xrightarrow{\delta} p_{m-1}$. Since $(p_{m-1}, p_m) \in \mathcal{R} \cup \mathcal{P}$, we have $(p_{m-1}, p_m) \in \mathcal{R}$ or $(p_{m-1}, p_m) \in \mathcal{P}$. Suppose the former; the reasoning is analogous in the latter case. Since $\mathcal{R}$ is a backward bisimulation we have that for every $i \in [1, k]$ there exists $q''_i \in [q'_i]_{\mathcal{R}}$ such that $f(q''_1, \ldots, q''_k) \xrightarrow{\delta} p_m$. Clearly, $q''_i \in [q'_i]_{\mathcal{R}'}$ for every $i \in [1, k]$ because $\mathcal{R} \subseteq \mathcal{R}'$. Finally, $[q'_i]_{\mathcal{R}'} = [q_i]_{\mathcal{R}'}$ and hence $q''_i \in [q_i]_{\mathcal{R}'}$ for every $i \in [1, k]$. This completes the induction, which proves the auxiliary statement. □

Let us conclude this section with a comparison of our notion of backward bisimulation to the notion of bisimulation in [6]. For the reader's convenience, we repeat the main definition of [6].

Definition 3.6 (cf. [6, Sect. 5]) *Let $\mathcal{P}$ be an equivalence relation on Q. We say that $\mathcal{P}$ is an* AKH-bisimulation *on M, if for every $(p, q) \in \mathcal{P}$ we have*

(i) if $p \in F$, then $q \in F$; and

(ii) for every symbol $f \in \Sigma_{(k)}$, index $j \in [1, k+1]$, and sequence $p_1, \ldots, p_{k+1} \in Q$ such that $f(p_1, \ldots, p_k) \xrightarrow{\delta} p_{k+1}$ with $p_j = p$ we have that for every $i \in [1, k+1]$ there exists $q_i \in [p_i]$ such that $f(q_1, \ldots, q_k) \xrightarrow{\delta} q_{k+1}$ and $q_j = q$.

We immediately observe that this notion of bisimulation is closely related to our notion of backward bisimulation. The next lemma expresses this formally.

Lemma 3.7 *Every AKH-bisimulation on M is a backward bisimulation on M.*

PROOF. Clearly, the condition of Definition 3.1 is met by setting j to $k+1$ in Definition 3.6. □

It follows that the coarsest backward bisimulation $\mathcal{R}$ on M must be coarser than the coarsest AKH-bisimulation $\mathcal{P}$ on M. Hence $(M/\mathcal{R})$ has at most as many states as $(M/\mathcal{P})$.

3.2. *Minimization algorithm*

At this point we know that there exists a coarsest backward bisimulation $\mathcal{R}$ on every fta M (Corollary 3.5), and that $(M/\mathcal{R})$ is an equivalent fta of size less or equal to that of M. These results now allow us to define a minimization algorithm for fta that proceeds as follows. The algorithm, which we henceforth refer to as Alg. 3, searches for the coarsest backward bisimulation $\mathcal{R}$ on M by producing increasingly refined equivalence relations $\mathcal{R}_0, \mathcal{R}_1, \mathcal{R}_2, \ldots$ on the state space of M. The first of these is the coarsest possible candidate solution. The relation $\mathcal{R}_{i+1}$ is derived from $\mathcal{R}_i$ by removing pairs of states that prevent $\mathcal{R}_i$ from being a backward bisimulation. The algorithm also produces an auxiliary sequence of relations $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \ldots$ that are used to determine these offending pairs in a time-efficient way. When $\mathcal{P}_i$ eventually coincides with $\mathcal{R}_i$, the relation $\mathcal{R}_i$ is the coarsest backward bisimulation on M. What then remains is to compute the aggregated fta $(M/\mathcal{R}_i)$. Before we discuss these steps in closer detail, it is necessary to extend our notation. From here on, we use m and n to denote the number of transitions and states of M (in other words, $m = |\delta|$ and $n = |Q|$). The maximal rank of any symbol in the input signature Σ of M is $r = \max\{k \mid \Sigma_{(k)} \neq \emptyset\}$.

To obtain a time-efficient algorithm, we apply a counting argument by Paige and Tarjan. The argument goes as follows: If we already know that there are n f-transitions that lead from elements of the word $D_1 \cdots D_k$ to the state q, and we count that there are m f-transitions that lead from elements of the word $C_1 \cdots C_k$ to q, where $C_i \subseteq D_i$ for every $i \in [1, k]$, then it is immediate that there are $n - m$ f-transitions that lead from elements of $(D_1 \times \cdots \times D_k) \setminus (C_1 \times \cdots \times C_k)$ to the state q. When we, in Alg. 3, need to test the implication of Definition 3.1, we can thus reduce the number of transitions that must be inspected explicitly, provided that we keep a record of the number of transitions that we counted in previous iterations. For this purpose, we introduce the *observation* mapping obs_q, which, given a left-hand side $f(D_1, \ldots, D_k) \in \Sigma(\mathfrak{P}(Q))$, tells us the number of f-transitions that lead from elements of $D_1 \cdots D_k$ to the state q.

Definition 3.8 *For every $q \in Q$, the mapping $\mathrm{obs}_q : \Sigma(\mathfrak{P}(Q)) \to \mathbb{N}$ is given by*

$$\mathrm{obs}_q(f(D_1, \ldots, D_k)) = |\{q_1 \cdots q_k \in D_1 \cdots D_k \mid f(q_1, \ldots, q_k) \xrightarrow{\delta} q\}|$$

for every $f(D_1, \ldots, D_k) \in \Sigma(\mathfrak{P}(Q))$.

As we will shortly see, we discard (q, q') from our maintained set of bisimilar states should $\mathrm{obs}_q(w)$ and $\mathrm{obs}_{q'}(w)$ disagree in the sense that one counter is positive whereas the other is zero for some $w \in \Sigma(\mathfrak{P}(Q))$.

To prove correctness we also need a vocabulary that lets us express how the rules of an aggregated fta $(M/\mathcal{R})$ relate to the rules of M. Intuitively, L_i is the set of left-hand sides that are possible in an fta with state space $(Q/\mathcal{P}_i)$ and input alphabet Σ. In each iteration of Alg. 3, a pivot block B is selected. Since we often address those entries in a set of left-hand sides L in which B occur, we abbreviate this subset by $L(B)$.

Definition 3.9 *Let B and D be subsets of Q, $i \in \mathbb{N}$ be an index, and $L \subseteq \Sigma(\mathfrak{P}(Q))$ be a language. We use*

– *L_i to abbreviate the set $\Sigma((Q/\mathcal{P}_i))$,*

– *$L(B)$ to abbreviate the set*

$$\{f(D_1, \ldots, D_k) \in L \mid D_i = B \text{ for some } i \in [1, k]\} \enspace ,$$

– *$L(B, \neg D)$ to abbreviate the set*

$$\{f(D_1, \ldots, D_k) \in L(B) \mid D_i \neq D \text{ for every } i \in [1, k]\} \enspace ,$$

– *L^w for some $w = f(D_1, \ldots, D_k) \in \Sigma(\mathfrak{P}(Q))$ to abbreviate the set of elements in L of the form $f(C_1, \ldots, C_k)$ where $C_i \subseteq D_i$ for every $i \in [1, k]$.*

Let us now give an informal description of the mappings `cut`, `split` and `splitn` that appear in Alg. 3, before we state their definitions. The first of these, `cut`, takes as argument a block $B \subseteq Q$ and returns a subset of $Q \times Q$. Subtracting $\mathtt{cut}(B)$ from an equivalence relation $\mathcal{R}$ on Q ensures that B is a separate block in $(Q/(\mathcal{R} \setminus \mathtt{cut}(B)))$ provided, of course, that $B^2 \subseteq \mathcal{R}$. The mapping `split` takes as argument a language L of left-hand sides and returns those pairs of states that can be proven not to be bisimilar by inspecting only transitions with left-hand sides in L. The mapping `splitn` implements the previously discussed counting argument by Paige and Tarjan; it takes left-hand-side languages L and L' and returns all those pairs of destintation states where one state may be reached from some left-hand-sides in $L \setminus L'$ and the other may not.

Definition 3.10 *Let B be a subset of Q and $L, L' \subseteq \Sigma(\mathfrak{P}(Q))$ be languages. We write*

– *$\mathtt{cut}(B)$ for the subset $(Q^2 \setminus B^2) \setminus (Q \setminus B)^2$ of $Q \times Q$;*

– *$\mathtt{split}(L)$ for the set of all (q, q') in $Q \times Q$ for which there exists $w \in L$ such that exactly one of $\mathrm{obs}_q(w)$ and $\mathrm{obs}_{q'}(w)$ is zero; and*

– *$\mathtt{splitn}(L, L')$ for the set of all (q, q') in $Q \times Q$ such that there exists $w \in L$ such that*

$$\mathrm{obs}_p(w) = \sum_{w' \in (L')^w} \mathrm{obs}_p(w')$$

holds for either $p = q$ or $p = q'$ but not both.

Let us briefly discuss how the sets L_0, L_1, L_2, … that are generated by Alg. 3 relate to each other. The set L_0 is equal to $\Sigma(\{Q\})$. Every $f(D_1, \ldots, D_k)$ in the set L_{i+1} is in either already in L_i or $w = f(D'_1, \ldots, D'_k)$ is in L_i where $D'_j = S_i$ if $D_j \in \{B_i, S_i \setminus B_i\}$ and $D'_j = D_j$ otherwise for every $j \in [1, k]$. Note that in the latter case $f(D_1, \ldots, D_k) \in (\mathrm{L}_i)^w$.

Example 3.11 The execution of Alg. 3 is traced on the fta N of Example 3.2. In the initialization, State 2 can be separated from the block $[1, 6]$ since only $\mathrm{obs}_2(b())$ is non-zero (and $b() \in \mathrm{L}_0$).

```
input:    a fta $M = (Q, \Sigma, \delta, F)$;

initially:
    $\mathcal{P}_0$    := $Q \times Q$;
    $\mathcal{R}_0$    := $\mathcal{P}_0 \setminus$ split($\mathtt{L}_0$);
    $i$     := 0;

while $\mathcal{R}_i \neq \mathcal{P}_i$:
    choose $S_i \in (Q/\mathcal{P}_i)$ and $B_i \in (Q/\mathcal{R}_i)$ such that
            $B_i \subset S_i$ and $|B_i| \leq |S_i|/2$;
    $\mathcal{P}_{i+1}$  := $\mathcal{P}_i \setminus$ cut($B_i$);
    $\mathcal{R}_{i+1}$  := $(\mathcal{R}_i \setminus$ split($\mathtt{L}_{i+1}(B_i)$)$) \setminus$ splitn($\mathtt{L}_i(S_i)$, $\mathtt{L}_{i+1}(B_i)$);
    $i$     := $i + 1$;

return:   the fta $(M/\mathcal{R}_i)$;
```

Alg. 3. A minimization algorithm for finite-state tree automata

Similarly, states 3 and 6 differ from 1, 4, and 5, as $\mathrm{obs}_3(f(Q,Q))$ and $\mathrm{obs}_6(f(Q,Q))$ are both non-zero. When the initialization is complete, we thus have

$$\mathcal{P}_0 = Q \times Q \text{ and } \mathcal{R}_0 = \{1,4,5\}^2 \cup \{2\}^2 \cup \{3,6\}^2 \ .$$

In the first iteration, we let $S_0 = Q$ and $B_0 = \{2\}$. The algorithm can now use the left-hand side $w = f(\{1,3,4,5,6\},\{2\})$ in $\mathrm{L}_1(B_0)$ to distinguish between state 3 and state 6, as $\mathrm{obs}_3(w) > 0$ whereas $\mathrm{obs}_6(w) = 0$. The next pair of equivalence relations is then:

$$\mathcal{P}_1 = \{2\}^2 \cup \{1,3,4,5,6\}^2 \text{ and } \mathcal{R}_1 = \{1,4,5\}^2 \cup \{2\}^2 \cup \{3\}^2 \cup \{6\}^2 \ .$$

As the states in $\{1,4,5\}$ do not appear at the left-hand side of any transition, this block will not be further divided. However, another two iterations are needed before $\mathcal{P}_i$ equals $\mathcal{R}_i$ and the algorithm terminates. □

Our next task is to verify that Alg. 3 computes the coarsest backward bisimulation on M as claimed. For this, we use the notations introduced in the outline above.

Lemma 3.12 *The relation $\mathcal{R}_i$ is a refinement of $\mathcal{P}_i$ for all $i \in \{0,1,2,\dots\}$.*

PROOF. The proof is by induction on i. The base case is satisfied by the initialization of $\mathcal{P}_0$ to $Q \times Q$. For the induction step, we proceed as follows. By definition, $\mathcal{R}_{i+1} \subseteq \mathcal{R}_i$ and $\mathcal{P}_{i+1} = \mathcal{P}_i \setminus \mathtt{cut}(B_i)$. Since $B_i \in (Q/\mathcal{R}_i)$, we also have the equality $\mathcal{R}_i \cap \mathtt{cut}(B_i) = \emptyset$, and by the induction hypothesis, the inclusion $\mathcal{R}_i \subseteq \mathcal{P}_i$. It follows that

$$\mathcal{R}_{i+1} \ \subseteq \ \mathcal{R}_i \ = \ \mathcal{R}_i \setminus \mathtt{cut}(B_i) \ \subseteq \ \mathcal{P}_i \setminus \mathtt{cut}(B_i) \ = \ \mathcal{P}_{i+1} \ . \qquad \square$$

Lemma 3.12 thus assures that $\mathcal{R}_i$ is a *proper* refinement of $\mathcal{P}_i$, for all $i \in \{0,\dots,t-1\}$ where t is the value of i at termination. This means that up to the termination point t we can always find blocks $B_i \in (Q/\mathcal{R}_i)$ and $S_i \in (Q/\mathcal{P}_i)$ such that B_i is contained in S_i, and the size of B_i is at most half the size of S_i. The check of the termination criterion can hence be combined with the choice of S_i and B_i, as we can only fail to choose these blocks if $\mathcal{R}_i$ and $\mathcal{P}_i$ are equal. Termination in less than $|Q|$ iterations is guaranteed by Lemma 3.13.

Lemma 3.13 *There exists a $t < |Q|$ such that $\mathcal{R}_t = \mathcal{P}_t$.*

PROOF. Clearly, the algorithm only terminates if $\mathcal{R}_t$ and $\mathcal{P}_t$ coincide for some t in $\mathbb{N}$. Up until termination, i.e. for all i less than t, we have that

$$|(Q/\mathcal{R}_i)| > |(Q/\mathcal{P}_i)| \quad \text{and} \quad |(Q/\mathcal{P}_{i+1})| > |(Q/\mathcal{P}_i)|$$

hold by Lemma 3.12. The size of both $(Q/\mathcal{R}_i)$ and $(Q/\mathcal{P}_i)$ is bounded from above by $|Q|$. Should the algorithm reach iteration $|Q| - 1$ before terminating, we have by necessity that both $|(Q/\mathcal{P}_{|Q|-1})|$ and $|(Q/\mathcal{R}_{|Q|-1})|$ are equal to $|Q|$, so $\mathcal{R}_{|Q|-1}$ and $\mathcal{P}_{|Q|-1}$ coincide. Consequently, there exists an integer t less than $|Q|$ such that $\mathcal{R}_t$ and $\mathcal{P}_t$ are equal. □

As mentioned earlier, Alg. 3 constructs in parallel two sequences of equivalence relations $(\mathcal{R}_i)_{i\in\mathbb{N}}$ and $(\mathcal{P}_i)_{i\in\mathbb{N}}$. The former represents the current hypothesis, and the latter has an auxiliary function in that it directs our search. We say that the relation $\mathcal{R}_i$ is stable with respect to $\mathcal{P}_i$ if, for every pair $(q, q') \in \mathcal{R}_i$, the observations made about q agree with those about q', provided that we restrict our view to the context of $\mathcal{P}_i$.

Definition 3.14 *Let $\mathcal{R}$ and $\mathcal{P}$ be two equivalence relations on Q such that $\mathcal{P}$ is coarser than $\mathcal{R}$. We say that $\mathcal{R}$* is stable with respect to *$\mathcal{P}$ if, for every (q, q') in $\mathcal{R}$ and $w \in \Sigma((Q/\mathcal{P}))$,*

$$\mathrm{obs}_q(w) = 0 \textit{ if and only if } \mathrm{obs}_{q'}(w) = 0 \ .$$

We say that $\mathcal{R}$ is stable *if it is stable with respect to itself.*

Note that every stable equivalence relation $\mathcal{R}$ is a backward bisimulation on M. Let us now make two remarks concerning Definition 3.9 that will help us understand the relationship between a left-hand side in L_i and those left-hand sides in L_{i+1} that are descendant from w.

Remark 3.15 *For every $i \in [0, t-1]$ and $w \in \mathrm{L}_i(S_i)$, it holds that*

$$\mathrm{obs}_q(w) = \sum_{w' \in \mathrm{L}^w_{i+1}(B_i)} \mathrm{obs}_q(w') + \sum_{w' \in \mathrm{L}^w_{i+1}(S_i \setminus B_i, \neg B_i)} \mathrm{obs}_q(w') \ .$$

Moreover, there is a unique $w' \in \mathrm{L}^w_{i+1}(S_i \setminus B_i, \neg B_i)$.

We are now equipped to state and prove Lemma 3.16.

Lemma 3.16 *The relation $\mathcal{R}_i$ is stable with respect to $\mathcal{P}_i$, for all $i \in [0, t]$.*

PROOF. By Lemma 3.12, the relation $\mathcal{P}_i$ is coarser than $\mathcal{R}_i$. The remaining proof is by induction on i. The base case follows from the definitions of $\mathcal{R}_0$ and $\mathcal{P}_0$. Now, let $(q, q') \in \mathcal{R}_{i+1}$. We show that $\mathrm{obs}_{q'}(w) = 0$ if $\mathrm{obs}_q(w) = 0$ for every $w \in \mathrm{L}_{i+1}$. Depending on w, there are three cases, and we examine each case.

First, let $w \in \mathrm{L}_i$. Since $(q, q') \in \mathcal{R}_{i+1}$ we have $(q, q') \in \mathcal{R}_i$ because $\mathcal{R}_i$ is coarser than $\mathcal{R}_{i+1}$. Supporting ourselves on the induction hypothesis, we have that $\mathrm{obs}_{q'}(w) = 0$ if $\mathrm{obs}_q(w) = 0$. Second, let $w \in \mathrm{L}_{i+1}(B_i)$, and here the desired equality follows from the fact that (q, q') is not in $\mathtt{split}(\mathrm{L}_{i+1}(B_i))$. Third, let $w \in \mathrm{L}_{i+1}(S_i \setminus B_i, \neg B_i)$. Let $w = f(D_1, \ldots, D_k)$ for some $f \in \Sigma_{(k)}$ and $D_1, \ldots, D_k \in (Q/\mathcal{P}_{i+1})$. Moreover, let $w' = f(D'_1, \ldots, D'_k)$ where $D'_i = S_i$ if $D_i = S_i \setminus B_i$ and $D'_i = D_i$ otherwise for every $i \in [1, k]$. Note that w is the unique element of $\mathrm{L}^{w'}_{i+1}(S_i \setminus B_i, \neg B_i)$ and according to Remark 3.15

$$\mathrm{obs}_q(w') = \sum_{w'' \in \mathrm{L}^{w'}_{i+1}(B_i)} \mathrm{obs}_q(w'') + \mathrm{obs}_q(w) = \sum_{w'' \in \mathrm{L}^{w'}_{i+1}(B_i)} \mathrm{obs}_q(w'') \ .$$

Since $(q, q') \in \mathcal{R}_{i+1}$ we have $(q, q') \notin \mathtt{splitn}(\mathrm{L}_i(S_i), \mathrm{L}_{i+1}(B_i))$. Consequently,

$$\mathrm{obs}_{q'}(w') = \sum_{w'' \in \mathrm{L}^{w'}_{i+1}(B_i)} \mathrm{obs}_{q'}(w'')$$

and by Remark 3.15 this yields $\mathrm{obs}_{q'}(w) = 0$. □

The next lemma serves to simplify the proof of Lemma 3.18, which states that if a backward bisimulation $\mathcal{R}$ is a refinement of the initial hypothesis $\mathcal{R}_0$, then $\mathcal{R}$ is also a refinement of every hypothesis $\mathcal{R}_i$, $i \in [0, t]$, that follows.

Lemma 3.17 *Let $\mathcal{R}$ be a backward bisimulation, and let $w \in \Sigma(P)$ where*

$$P = \{ \bigcup_{B \in S} B \mid S \subseteq (Q/\mathcal{R}) \} \enspace .$$

Then $\mathrm{obs}_{q'}(w) = 0$ if $\mathrm{obs}_q(w) = 0$ for every $(q, q') \in \mathcal{R}$.

PROOF. Suppose that $\mathrm{obs}_q(w) = 0$ with $w = f(D_1, \ldots, D_k)$. Consequently, there do not exist $q_1 \cdots q_k \in D_1 \cdots D_k$ such that $f(q_1, \ldots, q_k) \xrightarrow{\delta} q$. Since each D_i is a union of blocks of $(Q/\mathcal{R})$, we obtain that $\mathrm{obs}_q(w') = 0$ for every $w' = f(C_1, \ldots, C_k)$ with $C_i \in (D_i/\mathcal{R})$ for every $i \in [1, k]$. Due to the facts that $\mathcal{R}$ is a backward bisimulation and $(q, q') \in \mathcal{R}$ we can conclude that $\mathrm{obs}_{q'}(w') = 0$ for every such w'. This clearly yields $\mathrm{obs}_{q'}(w) = 0$. □

Lemma 3.18 *Every backward bisimulation $\mathcal{R}$ on M is a refinement of $\mathcal{R}_i$ for every $i \in [0, t]$.*

PROOF. The proof is by induction on i, and the base case is easily checked. To cover the induction step, we show that if $(q, q') \in \mathcal{R}$, then $(q, q') \in \mathcal{R}_{i+1}$. This is done by examining how the minimization algorithm obtains $\mathcal{R}_{i+1}$ from $\mathcal{R}_i$. By the induction hypothesis we have $(q, q') \in \mathcal{R}_i$. To have $(q, q') \in \mathcal{R}_{i+1}$, it must hold that $(q, q') \notin \mathtt{split}(\mathrm{L}_{i+1}(B_i))$ and hence $\mathrm{obs}_q(w) = 0$ if and only if $\mathrm{obs}_{q'}(w) = 0$ for every left-hand side $w = f(D_1, \ldots, D_k)$ in $\mathrm{L}_{i+1}(B_i)$. Since D_i is the union of blocks in $(Q/\mathcal{R}_i)$ and hence a union of blocks in $(Q/\mathcal{R})$, for all $i \in [1, k]$, this condition is satisfied by Lemma 3.17.

Finally, we have to prove that $(q, q') \notin \mathtt{splitn}(\mathrm{L}_i(S_i), \mathrm{L}_{i+1}(B_i))$. Let $w \in \mathrm{L}_i(S_i)$ and w'' be the unique element of $\mathrm{L}^w_{i+1}(S_i \setminus B_i, \neg B_i)$. For every $p \in \{q, q'\}$ we have

$$\mathrm{obs}_p(w) = \sum_{w' \in \mathrm{L}^w_{i+1}(B_i)} \mathrm{obs}_p(w') \quad \text{if and only if} \quad \sum_{w' \in \mathrm{L}^w_{i+1}(S_i \setminus B_i, \neg B_i)} \mathrm{obs}_p(w') = 0$$

by Remark 3.15. The latter holds precisely when $\mathrm{obs}_p(w'') = 0$ because w'' is the only element of $\mathrm{L}^w_{i+1}(S_i \setminus B_i, \neg B_i)$. It remains to show that $\mathrm{obs}_q(w'') = 0$ if and only if $\mathrm{obs}_{q'}(w'') = 0$. This holds by Lemma 3.17 because (i) $S_i \setminus B_i$ is a union of blocks of $\mathcal{R}_i$ and thus a union of blocks of $\mathcal{R}$ and (ii) all other blocks $D \in (Q/\mathcal{P}_{i+1}) \setminus \{B_i, S_i \setminus B_i\}$ are blocks of $(Q/\mathcal{P}_i)$ and thus a union of blocks of $\mathcal{R}_i$ and a union of blocks of $\mathcal{R}$. □

Now we collect the separate results in a final correctness theorem. In conjunction with Theorem 3.5 it shows that Alg. 3 really computes a backward bisimulation minimal fta $(M/\mathcal{R}_t)$.

Theorem 3.19 *$\mathcal{R}_t$ is the coarsest backward bisimulation on M.*

PROOF. Lemma 3.13 guarantees that Alg. 3 terminates and Lemma 3.16 shows that $\mathcal{R}_t$ is stable with respect to $\mathcal{P}_t$. Since $\mathcal{R}_t = \mathcal{P}_t$, the equivalence relation $\mathcal{R}_t$ is stable and hence a backward bisimulation on M (see Definition 3.1). Now let $\mathcal{P}$ be an arbitrary backward bisimulation on M. Obviously, $\mathcal{P}$ is a refinement of $\mathcal{R}_t$ by Lemma 3.18, which proves that $\mathcal{R}_t$ is the coarsest backward bisimulation on M. □

Let us now analyze the running time of Alg. 3 on M. In the complexity calculations, we write δ_L, where $L \subseteq \Sigma(\mathfrak{P}(Q))$, for the subset of δ that contains entries of the form $f(q_1, \ldots, q_k) \to q$, where $q_1 \cdots q_k \in B_1 \cdots B_k$ for some $f(B_1, \ldots, B_k) \in L$ and $q \in Q$. Our computation model is the random access machine [15], which supports indirect addressing, and thus allows the use of pointers. This means that we can represent each block in a partition $(Q/\mathcal{R})$ as a record of two-way pointers to its elements, and that we can link each state to its occurrences in the transition table. Given a state q and a block B, we can then determine $[q]_{\mathcal{R}}$ in constant time, and δ_L, where $L \subseteq \Sigma(\mathfrak{P}(Q))$, in time proportional to the number of entries.

To avoid pairwise comparison between states, we hash each state q in Q using obs_q as key, and then inspect which states are mapped to the same positions in the hash table. Since a random access machine has unlimited memory, we can always implement a collision-free hash, for instance, by interpreting the binary representation of obs_q as a memory address. The time required to hash a state q is consequently proportional to the size of the representation of obs_q.

The overall time complexity of the algorithm is

$$O\Big(\textsc{Init} + \sum_{i=0}^{t-1} (\textsc{Select}_i + \textsc{Cut}_i + \textsc{Split}_i + \textsc{Splitn}_i) + \textsc{Aggregate}\Big) \enspace ,$$

where

– $\textsc{Init}$ is the complexity of the initialization phase;
– $\textsc{Select}_i$ is the complexity of the choice of S_i and B_i;
– $\textsc{Cut}_i$ is the complexity of the computation of $\mathcal{P}_i \setminus \mathtt{cut}(B_i)$;
– $\textsc{Split}_i$ is the complexity of the computation of $\mathcal{R}_i \setminus \mathtt{split}(\mathrm{L}_{i+1}(B_i))$;
– $\textsc{Splitn}_i$ is the complexity of the subtraction of $\mathtt{splitn}(\mathrm{L}_i(S_i), \mathrm{L}_{i+1}(B_i))$; and
– $\textsc{Aggregate}$ is the complexity of the construction of the aggregated automaton $(M/\mathcal{R}_t)$.

The next lemma shows the complexities of the mentioned parts of Alg. 3.

Lemma 3.20 $\textsc{Init}$ *and* $\textsc{Aggregate}$ *are in* $O(rm + n)$*, whereas, for every* i *in* $[0, t-1]$*,*
- $\textsc{Select}_i$ *is in* $O(1)$*,*
- $\textsc{Cut}_i$ *is in* $O(|B_i|)$*, and*
- $\textsc{Split}_i$ *and* $\textsc{Splitn}_i$ *are in* $O\big(r\,|\delta_{\mathrm{L}_{i+1}(B_i)}|\big)$.

The next lemma is based on an observation by Hopcroft [1].

Lemma 3.21 *For each* $q \in Q$ *we have* $|\{B_i \mid i \in [0, t-1] \text{ and } q \in B_i\}| \leq \log_2 n$.

PROOF. Let B_i and B_j, where $i < j$, be two blocks that both include the state q. Since $\mathcal{R}_j$ is a refinement of $\mathcal{R}_i$, we have that B_j is a subset of B_i. We know then that $|B_j|$ is less or equal to $|B_i|/2$, or else B_j would violate the selection criteria for the B-blocks. If we order the B-blocks in which q occurs in descending order (with respect to their cardinality), we have that each block in the list is at most half the size of its predecessor. The first block in which q occurs cannot be larger than n, and the last block cannot be smaller than a singleton. Hence, the q is included in at most $\log_2 n$ distinct B-blocks. □

We are now ready to compute the overall complexity of Alg. 3.

Theorem 3.22 *The backward minimization algorithm is in* $O\big(r^2\, m \log n\big)$.

PROOF. By Lemma 3.20 the time complexity of the algorithm can be written as

$$O\Big((rm + n) + \sum_{i=0}^{t-1} (1 + |B_i| + r\,|\delta_{\mathrm{L}_{i+1}(B_i)}| + r\,|\delta_{\mathrm{L}_{i+1}(B_i)}|) + (rm + n)\Big) \enspace .$$

Omitting the smaller terms and simplifying, we obtain

$$O\left(r \sum_{i=0}^{t-1} |\delta_{\mathrm{L}_{i+1}(B_i)}|\right) \enspace .$$

According to Lemma 3.21, no state occurs in more than $\log_2 n$ distinct B-blocks, so no transition in δ will contribute by more than $r \log_2 n$ to the total sum. As there are m transitions, the overall time complexity of the algorithm is $O(r^2 m \log n)$. $\square$

Recall from Lemma 3.7 that every bisimulation in the sense of [6] is also a backward bisimulation (but the opposite is not true). Since Alg. 3 for minimization via backward bisimulation is computationally as efficient as the algorithm of [6] (see Theorem 3.22 and [6, Sect. 3]), Alg. 3 supersedes the algorithm of [6].

4. Forward Bisimulation

4.1. *Foundation*

In this section, we consider a computationally simpler notion of bisimulation. Minimization via forward bisimulation will generalize classical minimization of deterministic tree automata and actually coincide with it on deterministic tree automata (see Theorem 4.7). In addition, the two minimization procedures greatly increase their potential when they are used together in an alternating fashion (for practical experiments, see Sect. 5). Recently, [13] considered our backward bisimulation minimization followed by a slight variant of forward bisimulation minimization, which they called "composed bisimulation". We note that the fta obtained by their composed bisimulation minimization might be slightly different (better or worse) from the one obtained by executing our two minimization procedures in the mentioned order. However, in their evaluation [13] no essential difference presented itself. Moreover, their obtained fta also have the disadvantageous properties that our obtained fta have (e.g., the resulting fta might not be backward bisimulation minimal). On deterministic tree automata, composed bisimulation minimization will coincide with our forward bisimulation minimization.

As before, let $M = (Q, \Sigma, \delta, F)$ be a fta for the rest of this section.

Definition 4.1 *We say that an equivalence relation $\mathcal{R}$ on Q is a* forward bisimulation on M *if for every (p, q) in $\mathcal{R}$ we have*

(i) if $p \in F$, then $q \in F$; and

(ii) for every $f \in \Sigma_{(k)}$, $i \in [1, k]$, and $p', q_1, \ldots, q_k \in Q$ with $f(q_1, \ldots, q_{i-1}, p, q_{i+1}, \ldots, q_k) \xrightarrow{\delta} p'$ there exists $q' \in [p']$ such that $f(q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_k) \xrightarrow{\delta} q'$.

Note that Condition (ii) in Definition 4.1 is automatically fulfilled for all nullary symbols. Let us continue Example 3.2.

Example 4.2 Recall the aggregated fta from Example 3.2. An isomorphic fta N is given by $([1, 4], \Sigma, \delta, \{3, 4\})$ with

$$a() \xrightarrow{\delta} 1 \qquad b() \xrightarrow{\delta} 2 \qquad f(1, 2) \xrightarrow{\delta} 3 \qquad f(1, 1) \xrightarrow{\delta} 4 \ .$$

We have seen in Example 3.11 that N is backward bisimulation minimal. Let us consider the equivalence relation $\mathcal{P}$ induced by the partition $\{\{1\}, \{2\}, \{3, 4\}\}$. We claim that $\mathcal{P}$ is a forward bisimulation on N. Condition (i) of Definition 4.1 is met, and since $(1, 2) \notin \mathcal{P}$ and the states 3 and 4 only appear on the right hand side of rules, also Condition (ii) holds.

The aggregated fta $(N/\mathcal{P})$, displayed in Fig. 4, is $(Q', \Sigma, \delta', F')$ with $Q' = \{[1], [2], [3]\}$, $F' = \{[3]\}$, and

$$a() \xrightarrow{\delta'} [1] \qquad b() \xrightarrow{\delta'} [2] \qquad f([1], [2]) \xrightarrow{\delta'} [3] \qquad f([1], [1]) \xrightarrow{\delta'} [3] \ . \qquad \square$$

For the rest of this section, let $\mathcal{R}$ be a forward bisimulation on M. In the forward case, a collapsed state of $(M/\mathcal{R})$ functions like the combination of its constituents in M (cf. Sect. 3). In particular, bisimilar states need not recognize the same tree language.

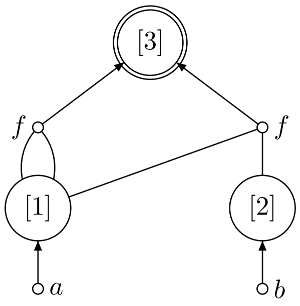

Fig. 4. Aggregated tree automaton $(N/\mathcal{P})$ of Example 4.2.

Lemma 4.3 (cf. [11, Theorem 3.1]) *$\mathcal{L}((M/\mathcal{R}))_{[q]} = \bigcup_{p \in [q]} \mathcal{L}(M)_p$ for every $q \in Q$.*

PROOF. We have already seen that $\bigcup_{p \in [q]} \mathcal{L}(M)_p \subseteq \mathcal{L}((M/\mathcal{R}))_{[q]}$ holds for every equivalence relation $\mathcal{R}$. For the remaining direction, let $(M/\mathcal{R}) = (Q', \Sigma, \delta', F')$. We prove the statement by induction for every $t \in T_\Sigma$. Suppose that $t = f[t_1, \ldots, t_k]$ for some $f \in \Sigma_{(k)}$ and $t_1, \ldots, t_k \in T_\Sigma$. By $t \in \mathcal{L}((M/\mathcal{R}))_{[q]}$ we have $[q] \in \delta'(t)$. The latter implies that there exist $D_1, \ldots, D_k \in Q'$ such that $f(D_1, \ldots, D_k) \xrightarrow{\delta'} [q]$ and $D_i \in \delta'(t_i)$ for every $i \in [1, k]$. With the help of the induction hypothesis, we obtain that there exist $q_1, \ldots, q_k \in Q$ such that $q_i \in D_i$ and $q_i \in \delta(t_i)$ for every $i \in [1, k]$. By construction of $(M/\mathcal{R})$, there also exist $p, p_1, \ldots, p_k \in Q$ with $p \in [q]$ such that $f(p_1, \ldots, p_k) \xrightarrow{\delta} p$ and $p_i \in [q_i]$ for every $i \in [1, k]$. It follows that $f(q_1, p_2, \ldots, p_k) \xrightarrow{\delta} p$, and consequently, $f(q_1, \ldots, q_k) \xrightarrow{\delta} p$ by Definition 4.1. This yields that $p \in \delta(t)$ and $t \in \bigcup_{p \in [q]} \mathcal{L}(M)_p$. $\square$

With the above lemma, it is now easy to prove that $(M/\mathcal{R})$ and M are equivalent provided that $\mathcal{R}$ is a forward bisimulation on M. At this point, we will also use Condition (i) of Definition 4.1.

Theorem 4.4 (cf. [11, Corollary 3.4]) *$\mathcal{L}((M/\mathcal{R})) = \mathcal{L}(M)$.*

PROOF. Let $t \in T_\Sigma$. We have $t \in \mathcal{L}((M/\mathcal{R}))$ if and only if there exists a state q of M such that $q \in F$ and $t \in \mathcal{L}((M/\mathcal{R}))_{[q]}$. By Definition 4.1 and Lemma 4.3, the latter holds if and only if there exists a state p of M such that $p \in F$ and $t \in \mathcal{L}(M)_p$. Clearly, this is exactly the case when $t \in \mathcal{L}(M)$. $\square$

As before, the coarsest of all forward bisimulations on M naturally yields the smallest aggregated fta. Any fta that admits only the identity as forward bisimulation is called *forward bisimulation minimal.* Such an fta cannot be reduced further with the help of some forward bisimulation.

Theorem 4.5 *For every fta M there exists a coarsest forward bisimulation $\mathcal{P}$ on M, and $(M/\mathcal{P})$ is an equivalent forward bisimulation minimal fta.*

PROOF. The latter statement is again clear (using Theorem 4.4). For the former statement, let $\mathcal{R}$ and $\mathcal{P}$ be forward bisimulations on M. We prove that there exists a forward bisimulation $\mathcal{R}'$ on M such that $\mathcal{R} \cup \mathcal{P} \subseteq \mathcal{R}'$. Let $\mathcal{R}'$ be the smallest equivalence containing $\mathcal{R} \cup \mathcal{P}$. We now show that $\mathcal{R}'$ is a forward bisimulation. Let $(p, q) \in \mathcal{R}'$. Thus there exist an integer $n \in \mathbb{N}$ and

$$(p_1, p_2), (p_2, p_3), \ldots, (p_{n-2}, p_{n-1}), (p_{n-1}, p_n) \in \mathcal{R} \cup \mathcal{P}$$

such that $p_1 = p$ and $p_n = q$. It is immediately clear that $q \in F$ whenever $p \in F$. Now to Condition (ii) of Definition 4.1. Let $f \in \Sigma_{(k)}$, $i \in [1, k]$, and $p', q_1, \ldots, q_k \in Q$ be such that $f(q_1, \ldots, q_{i-1}, p, q_{i+1}, \ldots, q_k) \xrightarrow{\delta} p'$. We will prove that there exists $q' \in [p']_{\mathcal{R}'}$ such that $f(q_1, \ldots, q_{i-1}, p_m, q_{i+1}, \ldots, q_k) \xrightarrow{\delta} q'$ for every $m \in [1, n]$. This is trivial for $m = 1$. Now let $m > 1$ and suppose that there exists $q' \in [p']_{\mathcal{R}'}$ such that $f(q_1, \ldots, q_{i-1}, p_{m-1}, q_{i+1}, \ldots, q_k) \xrightarrow{\delta} q'$. Moreover, suppose that $(p_{m-1}, p_m) \in \mathcal{R}$; the case that $(p_{m-1}, p_m) \in \mathcal{P}$ is handled analogously. By Definition 4.1, there exists $q'' \in [q']_{\mathcal{R}}$ such that $f(q_1, \ldots, q_{i-1}, p_m, q_{i+1}, \ldots, q_k) \xrightarrow{\delta} q''$. Since $\mathcal{R} \subseteq \mathcal{R}'$, it follows that $q'' \in [p']_{\mathcal{R}'}$. This completes the induction and proves the auxiliary statement. □

Finally, we relate forward bisimulation with classical minimization of deterministic tree automata. Let us first recall the required notions.

Definition 4.6 *The fta M is a dta (*deterministic *and* complete*), if for every $w \in \Sigma(Q)$ there exists exactly one $q \in Q$ such that $w \xrightarrow{\delta} q$.*

It is an easy exercise to verify that the fta $(M/\mathcal{R})$ is deterministic whenever M is so. Moreover, there exists a unique (up to isomorphism) minimal (with respect to the number of states) dta N that is equivalent to M [16,17]. The next theorem shows that N is isomorphic to $(M/\mathcal{R})$ where $\mathcal{R}$ is the coarsest forward bisimulation on M.

Theorem 4.7 *Let M be a dta without useless states, and let $\mathcal{R}$ be the coarsest forward bisimulation on M. Then $(M/\mathcal{R})$ is an equivalent minimal dta.*

PROOF. Let $M' = (Q', \Sigma, \delta', F')$ be the unique (up to isomorphism) equivalent minimal dta. We prove that there exists a forward bisimulation $\mathcal{R}$ on M such that $(M/\mathcal{R})$ and M' are isomorphic. By minimality of M' such a bisimulation must be the coarsest forward bisimulation on M.

We define the relation $\imath = \{(q, q') \in Q \times Q' \mid \mathcal{L}(M)_q \cap \mathcal{L}(M')_{q'} \neq \emptyset\}$. Since M has no useless states we have that $\mathcal{L}(M)_q \neq \emptyset$ for every $q \in Q$. Moreover, for every $q \in Q$ there exists $q' \in Q'$ such that $\mathcal{L}(M)_q \subseteq \mathcal{L}(M')_{q'}$. Hence for every $q \in Q$ there exists $q' \in Q'$ such that $(q, q') \in \imath$. Moreover, since M' is a dta, for every tree $t \in T_\Sigma$, there exists exactly one $q' \in Q'$ such that $t \in \mathcal{L}(M')_{q'}$, and thus, for every $q \in Q$ there exists at most one $q' \in Q'$ such that $(q, q') \in \imath$. Thus $\imath \colon Q \to Q'$. Now suppose that there exists $q' \in Q'$ so that there exists no $q \in Q$ with $\imath(q) = q'$. Clearly, $\mathcal{L}(M')_{q'} = \emptyset$ which contradicts to the minimality of M'. Thus $\imath$ is surjective.

Let $\mathcal{R} = \ker(\imath)$, which, by definition, is an equivalence relation. We first prove Condition (i) of Definition 4.1. Let $(p, q) \in \mathcal{R}$. We have to prove that $q \in F$ if $p \in F$. Since M has no useless states, there exist trees t and u in T_Σ such that $t \in \mathcal{L}(M)_p$ and $u \in \mathcal{L}(M)_q$. Suppose that $p \in F$. Then $t \in \mathcal{L}(M)$, and consequently, $t \in \mathcal{L}(M')$. Since $\mathcal{L}(M)_p \subseteq \mathcal{L}(M')_{\imath(p)}$ and $\mathcal{L}(M)_q \subseteq \mathcal{L}(M')_{\imath(q)}$, we obtain that $\imath(p) = \imath(q) \in F'$. Thus, also $u \in \mathcal{L}(M')$ and $u \in \mathcal{L}(M)$. This finally yields $q \in F$.

Now to Condition (ii). Let $f \in \Sigma_{(k)}$ be a symbol, $i \in [1, k]$ be an index, and $p', q_1, \ldots, q_k \in Q$ be states such that $f(q_1, \ldots, q_{i-1}, p, q_{i-1}, \ldots, q_k) \xrightarrow{\delta} p'$. By determinism, there exists a unique state $q' \in Q$ such that $f(q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_k) \xrightarrow{\delta} q'$. Thus it remains to show that $(p', q') \in \mathcal{R}$. We observe that $\imath(p') = \imath(q')$ if and only if there exists a state $r \in Q'$ such that $\mathcal{L}(M)_{p'} \cap \mathcal{L}(M')_r$ and $\mathcal{L}(M)_{q'} \cap \mathcal{L}(M')_r$ are nonempty. By assumption, $(p, q) \in \mathcal{R}$ and thus there exists a state $r' \in Q'$ and trees s and s' of T_Σ such that $s \in \mathcal{L}(M)_p \cap \mathcal{L}(M')_{r'}$ and $s' \in \mathcal{L}(M)_q \cap \mathcal{L}(M')_{r'}$. Further, since M has no useless states, for every $j \in [1, k]$ there exists a tree $s_j \in T_\Sigma$ such that $s_j \in \mathcal{L}(M)_{q_j}$. Since M' is deterministic, we obtain that there exists a state $r \in Q'$ such that

$$\{f[s_1, \ldots, s_{i-1}, s, s_{i+1}, \ldots, s_k], f[s_1, \ldots, s_{i-1}, s', s_{i+1}, \ldots, s_k]\} \subseteq \mathcal{L}(M')_r \ .$$

Clearly, $f[s_1, \ldots, s_{i-1}, s, s_{i+1}, \ldots, s_k] \in \mathcal{L}(M)_{p'}$ and $f[s_1, \ldots, s_{i-1}, s', s_{i+1}, \ldots, s_k] \in \mathcal{L}(M)_{q'}$. This proves that the intersections $\mathcal{L}(M)_{p'} \cap \mathcal{L}(M')_r$ and $\mathcal{L}(M)_{q'} \cap \mathcal{L}(M')_r$ are nonempty, and thus $(p', q') \in \mathcal{R}$. Hence Condition (ii) of Definition 4.1 is fulfilled and $\mathcal{R}$ is a forward bisimulation on M.

```
input:   a fta $M = (Q, \Sigma, \delta, F)$;

initially:
    $\mathcal{P}_0$     := $Q \times Q$;
    $\mathcal{R}_0$     := $((Q \setminus F)^2 \cup F^2) \setminus$ splitf$(Q)$;
    $i$      := 0;

while $\mathcal{R}_i \neq \mathcal{P}_i$:
    choose $S_i \in (Q/\mathcal{P}_i)$ and $B_i \in (Q/\mathcal{R}_i)$ such that
            $B_i \subset S_i$ and $|B_i| \leq |S_i|/2$;
    $\mathcal{P}_{i+1}$  := $\mathcal{P}_i \setminus$ cut$(B_i)$;
    $\mathcal{R}_{i+1}$  := $\big(\mathcal{R}_i \setminus$ splitf$(B_i)\big) \setminus$ splitfn$(S_i, B_i)$;
    $i$      := $i + 1$;

return:  the fta $(M/\mathcal{R}_i)$;
```

Alg. 5. A minimization algorithm based on forward bisimulation

It remains to prove that the aggregated fta $(M/\mathcal{R})$ is isomorphic to M'. Clearly, $(M/\mathcal{R})$ is a dta, has $|Q'|$ states, and is equivalent to M by Theorem 4.4. Thus $(M/\mathcal{R})$ is a minimal dta recognizing $\mathcal{L}(M)$ and by the uniqueness of such a dta isomorphic to M'. $\square$

4.2. *Minimization algorithm*

We now consider an algorithm that minimizes with respect to forward bisimulation. As in Sect. 3 this requires us to extend our notation. We denote by $\Sigma_\square(Q)$ the of set of *contexts* of $\Sigma(Q \cup \{\square\})$: the subset of elements of $\Sigma(Q \cup \{\square\})$ that contain the special symbol $\square \notin Q$ exactly once. We denote by $c[\![q]\!]$, where $c \in \Sigma_\square(Q)$ and $q \in Q$, the element of $\Sigma(Q)$ that is obtained by substituting q for the unique occurrence of $\square$ in c.

Definition 4.8 *For each state q in Q, the map* $\mathrm{obs}^q \colon \Sigma_\square(Q) \times \mathfrak{P}(Q) \to \mathbb{N}$ *is defined by*

$$\mathrm{obs}^q(c, D) = |\{q' \in D \mid c[\![q]\!] \xrightarrow{\delta} q'\}|$$

for every context $c \in \Sigma_\square(Q)$ and set $D \subseteq Q$ of states.

The mapping obs^q is similar to the mapping obs_q of Sect. 3.2 in that it is a local observation of the properties of q. The difference between them is that $\mathrm{obs}^q(c, D)$ is now the number of transitions from $c[\![q]\!]$ to a state of D. In contrast, obs_q looked from the other side of the rule.

Definition 4.9 *Let D and D' be subsets of Q.*

– *We write* splitf(D) *for the set of all pairs (p, q) in $Q \times Q$, for which there exists $c \in \Sigma_\square(Q)$ such that exactly one of* $\mathrm{obs}^p(c, D)$ *and* $\mathrm{obs}^q(c, D)$ *is zero.*

– *Similarly, we write* splitfn(D, D') *for the set of all pairs (q, q') in $Q \times Q$, for which there exists $c \in \Sigma_\square(Q)$ such that* $\mathrm{obs}^p(c, D) = \mathrm{obs}^p(c, D')$ *holds for either $p = q$ or $p = q'$ but not both.*

The second minimization algorithm is Alg. 5. This algorithm can be obtained from Alg. 3 by altering the way the family of relations $(\mathcal{R}_i)_{i \geq 0}$ is computed. The next example underlines the difference between the two algorithms.

Example 4.10 Let us trace the execution of Alg. 5 on the fta N from Example 4.2. In the initialization of $\mathcal{R}_0$, states 3 and 4 are separated out because they are both accepting. State 1 can also be distinguished as only $\mathrm{obs}^1(f(\square, 2), Q)$ is non-zero. This yields the equivalence relations

$$\mathcal{P}_0 = Q \times Q \text{ and } \mathcal{R}_0 = \{1\}^2 \cup \{2\}^2 \cup \{3, 4\}^2 \ .$$

As neither state 3 nor state 4 appear on a left-hand side of any transition, they will not be separated. The algorithm thus terminates and outputs $(M/\mathcal{R}_0)$ after a second iteration, during which $\mathcal{P}_0$ was refined to coincide with $\mathcal{R}_0$. □

Partial correctness and termination after $t < |Q|$ iterations of Alg. 5 are proved analogously to the case of backward bisimulation. For this reason we omit the explicit proofs (which can be found in [18]) and proceed immediately to the main result of this section.

Theorem 4.11 *$\mathcal{R}_t$ is the coarsest forward bisimulation on M.*

The time complexity of the forward bisimulation algorithm is computed using the same assumptions and notations as in Sect. 3. Although the computations are quite similar, they differ in that when the backward algorithm would examine every transition in δ of the form $f(q_1, \ldots, q_k) \to q$, where $q_j \in B_i$ for some $j \in [1, k]$, the forward algorithm considers only those transitions that are of the form $f(q_1, \ldots, q_k) \to q$, where $q \in B_i$. Since the latter set is on average a factor r smaller, we are able to obtain a proportional speed-up of the algorithm.

Theorem 4.12 *Algorithm 5 runs in time $O(rm \log n)$.*

5. Implementation

In this section, we present experimental results obtained by applying prototype implementations of Algorithms 3 and 5 to the problem of *language modeling* in the natural language processing domain [19]. A language model is a formalism for determining whether a given sentence is in a particular language. Language models are particularly useful in many applications of natural language and speech processing such as translation, transliteration, speech recognition, character recognition, etc., where transformation system output must be verified to be an appropriate sentence in the domain language. Recent research in natural language processing has focused on using tree-based models to capture syntactic dependencies in applications such as machine translation [20,21]. Thus, the problem is elevated to determining whether a given syntactic tree is in a language. Language models are naturally representable as finite-state automata. For efficiency and data sparsity reasons, whole sentences are not typically stored, but rather a sliding window of partial sentences is verified. In the string domain this is known as *n-gram* language modeling. We instead model *n-subtrees*, fixed-size pieces of a syntactic tree.

We prepared a data set by collecting 3-subtrees (i.e., all subtrees of height 3) from sentences taken from the Penn Treebank corpus of syntactically bracketed English news text [22]. An initial fta was constructed by representing each 3-subtree in a single path. We then wrote an implementation of the forward and backward minimization algorithms in Perl and applied them to data sets of various sizes of 3-subtrees. To illustrate that the two algorithms perform different minimizations, we then ran the forward algorithm on the result from the backward algorithm, and vice-versa. As Tables 1 and 2 show, the combination of both algorithms reduces the automata nicely, to less than half the size (in the sum of rules and states) of the original.

Tables 1 and 2 also include the state and rule count of the same automata after minimization with respect to AKH-bisimulation [6]. As these figures testify, the conditions placed on an AKH-bisimulation are much more restrictive than those met by a backward bisimulation. In fact, Definition 3.6 is obtained from Definition 3.1 if the two-way implication in Definition 3.1 is required to hold for *every* position in a transition rule (i.e. not just the last), while insisting that the sets of accepting and rejecting states are respected.

Example 5.1 Consider the nfa M_0 in Fig. 7 that recognizes the language $L = \{a, b, c, d, e\}^2$. The nfa M_0 is backward bisimulation minimal, but an application of Alg. 5 yields the nfa M_1 with the state set

$$\{\{q_1, q_2\}, \{q_3\}, \ldots, \{q_9\}, \{p_0\}, \{p_1\}\} \ .$$

TREES	ORIGINAL	BACKWARD	FORWARD	AKH	FWD, BWD	BWD, FWD
58	353	252	286	353	185	180
161	953	576	749	953	378	356
231	1373	781	1075	1373	494	468
287	1726	947	1358	1726	595	563

Table 1
State set reduction after minimization.

TREES	ORIGINAL	BACKWARD	FORWARD	AKH	FWD, BWD	BWD, FWD
58	353	252	341	353	240	235
161	953	576	905	953	534	512
231	1373	781	1299	1373	718	691
287	1726	947	1637	1726	874	842

Table 2
Rule set reduction after minimization.

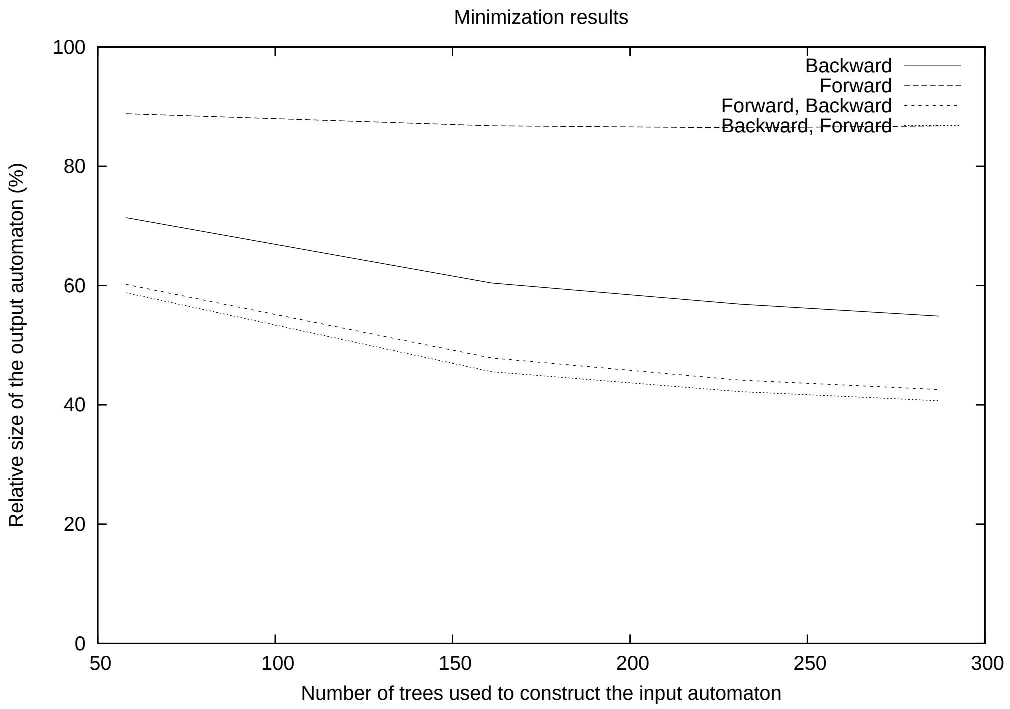

Fig. 6. The compression rate obtained by applying various kinds of bisimulation minimization. As AKH bisimulation did not affect the size of the input automaton, the corresponding test series is not reported in the figure.

Now, Alg. 3 discovers that states $\{q_1, q_2\}$ and $\{q_3\}$ are bisimilar, letting us form the even smaller automaton M_2. This turn-wise application of backward and forward bisimulation minimization can be continued until an automaton M_8 with three states is obtained. As this is the minimal number of states needed to recognize L, convergence is surely reached. □

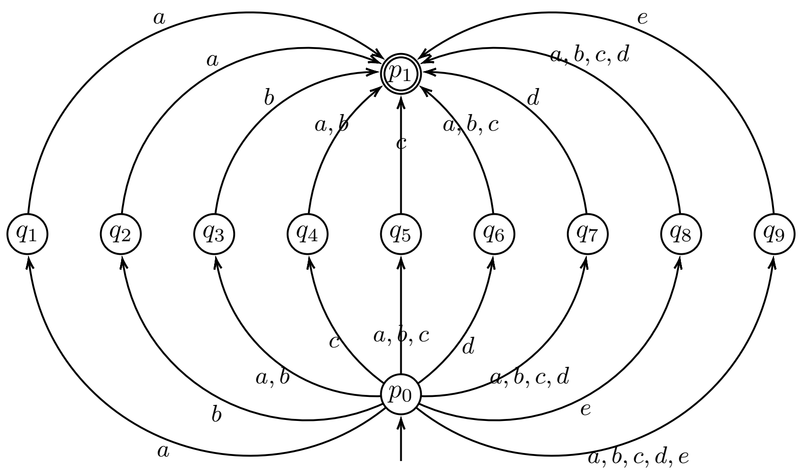

Fig. 7. The fta that is subject for minimization in Example 5.1.

6. Conclusion

We have introduced a general algorithm for bisimulation minimization of tree automata and discussed its operation under forward and backward bisimulation. The algorithm has attractive runtime properties and is useful for applications that desire a compact representation of large finite-state tree automata. We plan to include a refined implementation of this algorithm in a future version of the tree automata toolkit Tiburon, which is described in [23].

Acknowledgments

The authors acknowledge the support and advice of Frank Drewes and Kevin Knight. We thank Lisa Kaati for providing data and information relevant to the details of [6]. We would also like to thank the referees for their constructive comments.

References

[1] J. E. Hopcroft, An $n \log n$ algorithm for minimizing states in a finite automaton, in: Theory of Machines and Computations, Academic Press, 1971, pp. 189–196.

[2] A. R. Meyer, L. J. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, in: Proc. 13th Annual Symp. Foundations of Computer Science, IEEE Computer Society, 1972, pp. 125–129.

[3] G. Gramlich, G. Schnitger, Minimizing nfas and regular expressions, in: Proc. 22nd Int. Symp. Theoretical Aspects of Computer Science, Vol. 3404 of LNCS, Springer Verlag, 2005, pp. 399–411.

[4] G. Gramlich, G. Schnitger, Minimizing nfa's and regular expressions, Journal of Computer and System Sciences 73 (6) (2007) 908–923.

[5] H. Gruber, M. Holzer, Inapproximability of nondeterministic state and transition complexity assuming P $\neq$ NP, in: Proc. 11th Int. Conf. Developments in Language Theory, Vol. 4588 of LNCS, Springer Verlag, 2007, pp. 205–216.

[6] P. A. Abdulla, L. Kaati, J. Högberg, Bisimulation minimization of tree automata, in: Proc. 11th Int. Conf. Implementation and Application of Automata, Vol. 4094 of LNCS, Springer Verlag, 2006, pp. 173–185.

[7] P. A. Abdulla, B. Jonsson, P. Mahata, J. d'Orso, Regular tree model checking, in: Proc. 14th Int. Conf. Computer Aided Verification, Vol. 2404 of LNCS, Springer Verlag, 2002, pp. 555–568.

[8] K. Knight, J. Graehl, An overview of probabilistic tree transducers for natural language processing, in: Proc. 6th Int. Conf. Computational Linguistics and Intelligent Text Processing, Vol. 3406 of LNCS, Springer Verlag, 2005, pp. 1–24.

[9] R. Paige, R. Tarjan, Three partition refinement algorithms, SIAM Journal on Computing 16 (6) (1987) 973–989.

[10] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi, Tree automata: Techniques and applications, Available on: `http://www.grappa.univ-lille3.fr/tata` (1997).

[11] P. Buchholz, Bisimulation relations for weighted automata, Theoretical Computer Science 393 (1–3) (2008) 109–123.

[12] S. Yu, Regular languages, in: Word, Language, Grammar, Vol. 1 of Handbook of Formal Languages, Springer Verlag, 1997, Ch. 2, pp. 41–110.

[13] P. A. Abdulla, A. Bouajjani, L. Holík, L. Kaati, T. Vojnar, Composed bisimulation for tree automata, in: Proc. 13th Int. Conf. Implementation and Applications of Automata, Vol. 5148 of LNCS, Springer Verlag, 2008, pp. 212–222.

[14] P. A. Abdulla, J. Högberg, L. Kaati, Bisimulation minimization of tree automata, International Journal of Foundations of Computer Science 18 (4) (2007) 699–713.

[15] C. H. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.

[16] F. Gécseg, M. Steinby, Tree Automata, Akadémiai Kiadó, 1984.

[17] F. Gécseg, M. Steinby, Tree languages, in: Handbook of Formal Languages, Vol. 3, Springer Verlag, 1997, Ch. 1, pp. 1–68.

[18] J. Högberg, A. Maletti, J. May, Backward and forward bisimulation minimisation of tree automata, Tech. Rep. ISI-TR-633, University of Southern California (2007).

[19] F. Jelinek, Continuous speech recognition by statistical methods, Proc. IEEE 64 (4) (1976) 532–557.

[20] M. Galley, M. Hopkins, K. Knight, D. Marcu, What's in a translation rule?, in: Proc. 2004 Human Language Technology Conf. of the North American Chapter of the Association for Computational Linguistics, 2004, pp. 273–280.

[21] K. Yamada, K. Knight, A syntax-based statistical translation model, in: Proc. 39th Meeting of the Association for Computational Linguistics, Morgan Kaufmann, 2001, pp. 523–530.

[22] M. P. Marcus, M. A. Marcinkiewicz, B. Santorini, Building a large annotated corpus of english: The Penn treebank, Computational Linguistics 19 (2) (1993) 313–330.

[23] J. May, K. Knight, Tiburon: A weighted tree automata toolkit, in: Proc. 11th Int. Conf. Implementation and Application of Automata, Vol. 4094 of LNCS, Springer Verlag, 2006, pp. 102–113.