Backward and Forward Bisimulation Minimisation of Tree Automata

Johanna Högberg[1], Andreas Maletti[2], and Jonathan May[3]

[1] Dept. of Computing Science, Umeå University, S–90187 Umeå, Sweden
johanna@cs.umu.se
[2] Faculty of Computer Science, Technische Universität Dresden, D–01062 Dresden, Germany
maletti@tcs.inf.tu-dresden.de
[3] Information Sciences Institute, University of Southern California, Marina Del Rey, CA 90292
jonmay@isi.edu

Abstract. We improve an existing bisimulation minimisation algorithm for tree automata by introducing backward and forward bisimulations and developing minimisation algorithms for them. Minimisation via forward bisimulation is also effective for deterministic automata and faster than the previous algorithm. Minimisation via backward bisimulation generalises the previous algorithm and is thus more effective but just as fast. We demonstrate implementations of these algorithms on a typical task in natural language processing.

Keywords: bisimulation, tree automata, minimisation, natural language processing.

1 Introduction

Automata minimisation has a long and studied history. For deterministic finite (string) automata (dfa) efficient algorithms exist. The well-known algorithm by Hopcroft [1] runs in time $O(n \log n)$ where n is the number of states of the input automaton. The situation is worse for non-deterministic finite automata (nfa). The minimisation problem for nfa is PSPACE-complete [2] and cannot even be efficiently approximated within the factor $o(n)$ unless P = PSPACE [3]. The problem must thus be restricted to allow algorithms of practical value, and one possibility is to settle for a partial minimisation. This was done in [4] for *non-deterministic tree automata* (nta), which are a generalisation of nfa that recognise tree languages and are used in applications such as model checking [5] and natural language processing [6].

The minimisation algorithm in [4] was inspired by a partitioning algorithm due to Paige and Tarjan [7], and relies heavily on *bisimulation*; a concept introduced by R. Milner as a formal tool for investigating transition systems. Intuitively, two states are bisimilar if they can simulate each other, or equivalently, the observable behaviour of the two states must coincide. Depending on the capacity of the

Jan Holub and Jan Žd'árek (Eds.): CIAA 2007, LNCS 4783, pp. 109–121, 2007.

observer, we obtain different types of bisimulation. In all cases we assume that the observer has the capacity to observe the final reaction to a given input (i.e., the given tree is either accepted or rejected), so the presence of bisimilar states in an automaton indicates redundancy. 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 [4] in two ways: (i) we relax the constraints for state equivalence, and (ii) we introduce a new bisimulation relation that (with effect) can be applied to deterministic (bottom-up) tree automata (dta) [8]. Note that [4] is ineffective on dta. Thus we are able to find smaller automata than previously possible.

The two ways correspond, respectively, to two types of bisimulation: *backward* and *forward* bisimulation [9]. 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 labelled by the special symbol $\Box$) that would be recognised by M, if the (bottom-up) computation starts with the state q at the unique $\Box$-labelled 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 observe what already happened). The past of a state q is the language that would be recognised by the automaton if q were its only final state.

Both types of bisimulation yield efficient minimisation procedures, which can be applied to arbitrary nta. Further, forward bisimulation minimisation is useful on dta. It computes the unique minimal dta recognising the same language as the input dta (see Theorem 29). More importantly, it is shown in Theorem 27 that the asymptotic time-complexity of our minimisation algorithm is $O(rm \log n)$, where r is the maximal rank of the symbols in the input alphabet, m is the size of the transition table, and n is the number of states. Thus our algorithm supersedes the currently best minimisation algorithm [8] for dta, whose complexity is $O(rmn)$. Backward bisimulation, though slightly harder to compute, has great practical value as well. Our backward bisimulation is weaker than the bisimulation of [4]. Consequently, the nta obtained by our backward bisimulation minimisation algorithm will have at most as many states as the automata obtained by the minimisation algorithm of [4]. In addition, the asymptotic time-complexity of our algorithm (see Theorem 15), which is $O\big(r^2\, m \log n\big)$, is the same as the one for the minimisation algorithm of [4]. In [4] 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 minimisation only yield nta that are minimal with respect to the respective type of bisimulation. Thus applying forward and backward bisimulation minimisation in an alternating fashion commonly yields even smaller nta (see Sect. 5). Second, in certain domains only one type of bisimulation minimisation is effective. For example, backward bisimulation minimisation is ineffective on dta because no two 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 minimisation and the 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 $\{k, k+1, \ldots, n\}$ is abbreviated to $[k, n]$, and the cardinality of a set S is denoted by $|S|$. We abbreviate $Q \times Q$ as Q^2, and the inclusion $q_i \in D_i$ for all $i \in [1, k]$ as $q_1 \cdots q_k \in D_1 \cdots D_k$.

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 an element s in S with respect to $\mathcal{R}$ is the set $[s]_{\mathcal{R}} = \{s' \mid (s, s') \in \mathcal{R}\}$. 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 and s' are in relation $\mathcal{R}$, 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 $\Sigma = \bigcup_{k \in \mathbb{N}} \Sigma_{(k)}$ which is partitioned into pairwise disjoint subsets $\Sigma_{(k)}$. The set T_Σ of *trees over* Σ is the smallest language over Σ such that $f\, t_1 \cdots t_k$ is in T_Σ for every f in $\Sigma_{(k)}$ and all $t_1, \ldots, t_k$ in T_Σ. To improve readability we write $f[t_1, \ldots, t_k]$ instead of $f\, t_1 \cdots t_k$ unless k is zero. Any subset of T_Σ is called a *tree language*.

A *non-deterministic tree automaton* (for short: nta) 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 $f(q_1, \ldots, q_k) \to q_{k+1}$ for some symbol f in $\Sigma_{(k)}$ and $q_1, \ldots, q_{k+1} \in Q$. Finally, $F \subseteq Q$ is a set of *accepting* states. To indicate that a transition $f(q_1, \ldots, q_k) \to q_{k+1}$ is in δ, we write $f(q_1, \ldots, q_k) \xrightarrow{\delta} q_{k+1}$. In the obvious way, δ extends to trees yielding a mapping $\delta\colon T_\Sigma \to \mathfrak{P}(Q)$; i.e., $\delta(t) = \{q \mid f(q_1, \ldots, q_k) \xrightarrow{\delta} q$ and $q_i \in \delta(t_i)$ 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 *recognised* by M is $\mathcal{L}(M) = \bigcup_{q \in F} \mathcal{L}(M)_q$. Finally, we say that a state q in Q is *useless* if $\mathcal{L}(M)_q = \emptyset$.

3 Backward Bisimulation

Foundation. We first introduce the notion of *backward bisimulation* for a nta M. This type of bisimulation requires bisimilar states to recognise the same tree language. Next, we show how to collapse a block of bisimilar states into just a single state to obtain a potentially smaller nta M'. The construction is such that M' recognises exactly $\mathcal{L}(M)$. Finally, we show that there exists a coarsest backward bisimulation on M, which leads to the smallest collapsed nta.

Definition 1 (cf. [9, Definition 4.1]). *Let $M = (Q, \Sigma, \delta, F)$ be a nta, and let $\mathcal{R}$ be an equivalence relation on Q. We say that $\mathcal{R}$ is a* backward bisimulation *on M if for every $(p, q) \in \mathcal{R}$, symbol f of $\Sigma_{(k)}$, and sequence $D_1, \dots, D_k \in (Q/\mathcal{R})$*

$$\bigvee_{p_1 \cdots p_k \in D_1 \cdots D_k} f(p_1, \dots, p_k) \xrightarrow{\delta} p \quad \iff \quad \bigvee_{q_1 \cdots q_k \in D_1 \cdots D_k} f(q_1, \dots, q_k) \xrightarrow{\delta} q \ .$$

Example 2. Suppose we want to recognise the tree language $L = \{f[a, b], f[a, a]\}$ over the ranked alphabet $\Sigma = \Sigma_{(2)} \cup \Sigma_{(0)}$ with $\Sigma_{(2)} = \{f\}$ and $\Sigma_{(0)} = \{a, b\}$. We first construct nta N_1 and N_2 that recognise only $f[a, b]$ and $f[a, a]$, respectively. Then we construct N by disjoint union of N_1 and N_2. In this manner we could obtain the nta $N = ([1, 6], \Sigma, \delta, \{3, 6\})$ 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 $\mathcal{P} = \{1, 4, 5\}^2 \cup \{2\}^2 \cup \{3\}^2 \cup \{6\}^2$. We claim that $\mathcal{P}$ is a backward bisimulation on N. In fact, we only need to check the transitions leading to 1, 4, or 5 in order to justify the claim. Trivially, the condition of Definition 1 is met for such transitions because (i) $a() \to q$ is in δ and (ii) $b() \to q$ is not in δ for every state $q \in \{1, 4, 5\}$. □

Next we describe how a nta $M = (Q, \Sigma, \delta, F)$ may be collapsed with respect to an equivalence relation $\mathcal{R}$ on Q. In particular, we will invoke this construction for some $\mathcal{R}$ that is a backward (in the current section) or forward (in Sect. 4) bisimulation on M.

Definition 3 (cf. [9, Definition 3.3]). *Let $M = (Q, \Sigma, \delta, F)$ be a nta, and let $\mathcal{R}$ be an equivalence relation on Q. The* aggregated nta (with respect to M and $\mathcal{R}$)*, denoted by $(M/\mathcal{R})$, is the nta $((Q/\mathcal{R}), \Sigma, \delta', F')$ given by $F' = \{[q] \mid q \in F\}$ and*

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

The nta $(M/\mathcal{R})$ has as many states as there are equivalence classes with respect to $\mathcal{R}$. Thus $(M/\mathcal{R})$ cannot have more states than M.

Example 4. Let N be the nta and $\mathcal{P}$ the backward bisimulation of Example 2. According to Definition 3, the aggregated nta $(N/\mathcal{P})$, which should recognise the language $\{f[a, b], f[a, a]\}$, is $(Q', \Sigma, \delta', F')$ where $Q' = \{[1], [2], [3], [6]\}$ and $F' = \{[3], [6]\}$ and

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

For the rest of this section, we let $M = (Q, \Sigma, \delta, F)$ be an arbitrary but fixed nta and $\mathcal{R}$ be a backward bisimulation on M. Next we prepare Corollary 6, which follows from Lemma 5. This corollary shows that M and $(M/\mathcal{R})$ recognise the same tree language. The linking property is that the states q and $[q]$ (in their respective nta) recognise the same tree language. In fact, this also proves that bisimilar states in M recognise the same tree language.

Lemma 5 (cf. [9, Theorem 4.2]). *For any state q of M, $\mathcal{L}((M/\mathcal{R}))_{[q]} = \mathcal{L}(M)_q$.* □

Corollary 6 (cf. [9, Theorem 4.2]). $\mathcal{L}((M/\mathcal{R})) = \mathcal{L}(M)$. □

Clearly, among all backward bisimulations on M the coarsest one yields the smallest aggregated nta. Further, this nta admits only the trivial backward bisimulation.

Theorem 7. *There exists a coarsest backward bisimulation $\mathcal{P}$ on M, and the identity is the only backward bisimulation on $(M/\mathcal{P})$.*

Minimisation algorithm. We now present a minimisation algorithm for nta that draws on the ideas presented. Algorithm 1 searches for the coarsest backward bisimulation $\mathcal{R}$ on M by producing increasingly refined equivalence relations $\mathcal{R}_0, \mathcal{R}_1, \mathcal{R}_2, \ldots$. 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 find these offending pairs. When $\mathcal{P}_i$ eventually coincides with $\mathcal{R}_i$, the relation $\mathcal{R}_i$ is the coarsest backward bisimulation on M.

Before we discuss the algorithm, its correctness, and its time complexity, we extend our notation.

Definition 8. *For every $q \in Q$ and $k \in \mathbb{N}$ let $\mathrm{obs}_q^k : \Sigma_{(k)} \times \mathfrak{P}(Q)^k \to \mathbb{N}$ be the mapping given by*

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

for every $f \in \Sigma_{(k)}$ and $D_1 \cdots D_k \in \mathfrak{P}(Q)^k$ □

Intuitively, $\mathrm{obs}_q^k(f, D_1 \cdots D_k)$, the *observation*, is the number of f-transitions that lead from blocks $D_1, \ldots, D_k$ to q, and thus a local observation of the properties of q (cf. Definition 1). As we will shortly see, we discard (q, q') from our maintained set of bisimilar states should obs_q^k and $\mathrm{obs}_{q'}^k$ disagree in the sense that one is positive whereas the other is zero.

Definition 9. *Let B be a subset of Q, $i \in \mathbb{N}$, and $L, L' \subseteq \mathfrak{P}(Q)^*$ be languages.*

- *Let $r = \max\{k \mid \Sigma_{(k)} \neq \emptyset\}$.*
- *The notation L_i will abbreviate $(Q/\mathcal{P}_i)^0 \cup \cdots \cup (Q/\mathcal{P}_i)^r$.*
- *We use $L(B)$ to abbreviate $\{D_1 \cdots D_k \in L \mid D_i = B \text{ for some } i \in [1, k]\}$.*
- *We write $\mathrm{cut}(B)$ for the subset $(Q^2 \setminus B^2) \setminus (Q \setminus B)^2$ of $Q \times Q$.*
- *We write $\mathrm{split}(L)$ for the set of all (q, q') in $Q \times Q$ for which there exist $f \in \Sigma_{(k)}$ and a word $w \in L$ of length k such that exactly one of $\mathrm{obs}_q^k(f, w)$ and $\mathrm{obs}_{q'}^k(f, w)$ is zero.*

```
input:   a nta M = (Q, Σ, δ, F);
initially:
    P_0   := Q × Q;
    R_0   := P_0 \ split(L_0);
    i     := 0;
while R_i ≠ P_i:
    choose S_i ∈ (Q/P_i) and B_i ∈ (Q/R_i) such that
           B_i ⊂ S_i and |B_i| ≤ |S_i| /2;
    P_{i+1} := P_i \ cut(B_i);
    R_{i+1} := (R_i \ split(L_{i+1}(B_i))) \ splitn(L_i(S_i), L_{i+1}(B_i));
    i       := i + 1;
return:  the nta (M/R_i);
```

Alg. 1. A minimisation algorithm for non-deterministic tree automata

– *Finally, we write* $\operatorname{splitn}(L, L')$ *for the set of all* (q, q') *in* $Q \times Q$ *such that there exist a symbol* f *in* $\Sigma_{(k)}$ *and a word* $D_1 \cdots D_k \in L$ *of length* k *such that*

$$\operatorname{obs}_p^k(f, D_1 \cdots D_k) = \sum_{\substack{C_1 \cdots C_k \in L', \\ \forall i \in [1,k] \colon C_i \subseteq D_i}} \operatorname{obs}_p^k(f, C_1 \cdots C_k)$$

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. 1 relate to each other. The set L_0 contains a single word of length k, for each $k \in [0, r]$, namely Q^k. Every word w of length k in the set L_{i+1} is in either in L_i, or of the form $D_1 \cdots D_k$, where $D_j \in \{B_i, S_i \setminus B_i\}$ for some $j \in [1, k]$ and $D_l \in (Q/\mathcal{P}_{i+1})$ for every $l \in [1, k]$.

Example 10. We trace the execution of the minimisation algorithm on the automaton N of Example 2. Let us start with the initialisation. State 2 can be separated from $[1, 6]$ since only obs_2^0 is non-zero for the symbol b and the empty word $\epsilon \in \mathrm{L}_0$. Similarly, states 3 and 6 differ from 1, 4, and 5, as obs_3^2 and obs_6^2 are both non-zero for the symbol f and word QQ. Thus $\mathcal{P}_0 = Q \times Q$ 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 symbol f and word $w = (Q \setminus \{2\})\{2\}$ in $\mathrm{L}_1(B_1)$ to distinguish between state 3 and state 6, as $\operatorname{obs}_3^2(f, w) > 0$ whereas $\operatorname{obs}_6^2(f, w) = 0$. The next pair of relations is then:

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

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. Two more iterations are needed before $\mathcal{P}_3$ equals $\mathcal{R}_3$. □

Next we establish that the algorithm really computes the coarsest backward bisimulation on M. We use the notations introduced in the algorithm.

Lemma 11. *The relation $\mathcal{R}_i$ is a refinement of $\mathcal{P}_i$, for all $i \in \{0, 1, 2, \dots\}$.* □

Lemma 11 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. 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 of that of S_i. This means that checking the termination criterion can be combined with the choice of S_i and B_i, because we can only fail to choose these blocks if $\mathcal{R}$ and $\mathcal{P}$ are equal. Lemma 11 also guarantees that the algorithm terminates in less than n iterations.

Theorem 12. *$\mathcal{R}_t$ is the coarsest backward bisimulation on M.* □

Let us now analyse the running time of the minimisation algorithm on M. We use n and m to denote the size of the sets Q and δ, respectively. In the complexity calculations, we write δ_L, where $L \subseteq \mathfrak{P}(Q)^*$, for the subset of δ that contains entries of the form $f(q_1, \dots, q_k) \to q$, where $f \in \Sigma_{(k)}$, $q \in Q$, and $q_1 \cdots q_k$ is in $B_1 \cdots B_k$ for some $B_1 \cdots B_k \in L$. Our computation model is the random access machine [10], 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 $L \subseteq \mathfrak{P}(Q)^*$, in time proportional to the number of entries.

To avoid pairwise comparison between states, we hash each state q in Q using $(\mathrm{obs}_q^k)_{k\in[0,r]}$ as key, and then inspect which states end up at the same positions in the hash table. Since a random access machine has unlimited memory, we can always implement a collision free hash h; i.e., by interpreting the binary representation of $(\mathrm{obs}_q^k)_{k\in[0,r]}$ as a memory address, and the time required to hash a state q is then proportional to the size of the representation of $(\mathrm{obs}_q^k)_{k\in[0,r]}$.

The overall time complexity of the algorithm is

$$O\Big(\textsc{Init} + \sum\nolimits_{i\in[0,t-1]} (\textsc{Select}_i + \textsc{Cut}_i + \textsc{Split}_i + \textsc{Splitn}_i) + \textsc{Aggregate}\Big) \ ,$$

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

Lemma 13. Init *and* 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 $O\big(r\,|\delta_{\mathrm{L}_{i+1}(B_i)}|\big)$.* □

Lemma 14. *For each $q \in Q$ we have $|\{B_i \mid i \in [0, t-1] \text{ and } q \in B_i\}| \leq \log n$.* □

Theorem 15. *The backward minimisation algorithm is in* $O\big(r^2\, m \log n\big)$. □

Proof. By Lemma 13 the time complexity of the algorithm can be written as

$$O\Big((rm+n) + \sum\nolimits_{i\in[0,t-1]} (1 + |B_i| + r\,|\delta_{\mathrm{L}_{i+1}(B_i)}| + r\,|\delta_{\mathrm{L}_{i+1}(B_i)}|) + (rm+n)\Big)\ .$$

Omitting the smaller terms and simplifying, we obtain $O\left(r \sum_{i\in[0,t-1]} |\delta_{\mathrm{L}_{i+1}(B_i)}|\right)$. According to Lemma 14, no state occurs in more than $\log n$ distinct B-blocks, so no transition in δ will contribute by more than $r \log n$ to the total sum. As there are m transitions, the overall time complexity of the algorithm is $O(r^2 m \log n)$. □

We next compare the presented backward bisimulation to the bisimulation of [4].

Definition 16 (cf. [4, 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) $p \in F$ if and only if $q \in F$; and (ii) for every symbol f in $\Sigma_{(k)}$, index $i \in [1,n]$, and sequence $D_1, \dots, D_n$ of blocks in $(Q/\mathcal{P})$*

$$\bigvee_{\substack{p_1\cdots p_n \in D_1\cdots D_n,\\ p_i = p}} f(p_1,\dots,p_k) \xrightarrow{\delta} p_n \iff \bigvee_{\substack{q_1\cdots q_n \in D_1\cdots D_n,\\ q_i = q}} f(q_1,\dots,q_k) \xrightarrow{\delta} q_n$$

where $n = k+1$. □

Lemma 17. *Every AKH-bisimulation on M is a backward bisimulation on M.* □

The coarsest backward bisimulation $\mathcal{R}$ on M is coarser than the coarsest AKH-bisimulation $\mathcal{P}$ on M. Hence $(M/\mathcal{R})$ has at most as many states as $(M/\mathcal{P})$. Since our algorithm for minimisation via backward bisimulation is computationally as efficient as the algorithm of [4] (see Theorem 15 and [4, Sect. 3]), it supersedes the minimisation algorithm of [4].

4 Forward Bisimulation

Foundation. In this section we consider a computationally simpler notion of bisimulation. Minimisation via forward bisimulation coincides with classical minimisation on deterministic nta. In addition, the two minimisation procedures greatly increase their potential when they are used together in an alternating fashion (see Sect. 5).

Definition 18. *Let $M = (Q, \Sigma, \delta, F)$ be a nta, and let $\mathcal{R}$ be an equivalence relation on Q. We say that $\mathcal{R}$ is a* forward bisimulation *on M if for every (p,q) in $\mathcal{R}$ we have (i) $p \in F$ if and only if $q \in F$; and (ii) for every symbol f in $\Sigma_{(k)}$, index $i \in [1,k]$, sequence of states $q_1, \dots, q_k$ in Q, and block D in $(Q/\mathcal{R})$*

$$\bigvee_{r\in D} f(q_1,\dots,q_{i-1},p,q_{i+1},\dots,q_k) \xrightarrow{\delta} r \Longleftrightarrow \bigvee_{r\in D} f(q_1,\dots,q_{i-1},q,q_{i+1},\dots,q_k) \xrightarrow{\delta} r\ .$$

□

Note that Condition (ii) in Definition 18 is automatically fulfilled for all nullary symbols. Let us continue Example 4 (the aggregated nta is defined in Definition 3).

Example 19. Recall the aggregated nta from Example 4. An isomorphic nta 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 10 that N admits only the trivial backward bisimulation. Let us consider $\mathcal{P} = \{1\}^2 \cup \{2\}^2 \cup \{3,4\}^2$. We claim that $\mathcal{P}$ is a forward bisimulation on N. Condition (i) of Definition 18 is met, and since $(1,2) \notin \mathcal{P}$ and the states 3 and 4 only appear on the right hand side of $\xrightarrow{\delta}$, also Condition (ii) holds. Thus $\mathcal{P}$ is a forward bisimulation.

The aggregated nta $(N/\mathcal{P})$ is $(Q', \Sigma, \delta', F')$ with $Q' = \{[1],[2],[3]\}$ and $F' = \{[3]\}$ 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, we let $M = (Q, \Sigma, \delta, F)$ be an arbitrary but fixed nta and $\mathcal{R}$ 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 recognise the same tree language. However, $(M/\mathcal{R})$ and M do recognise the same tree language.

Lemma 20 (cf. [9, Theorem 3.1]). *$\mathcal{L}((M/\mathcal{R}))_{[q]} = \bigcup_{p \in [q]} \mathcal{L}(M)_p$ for every $q \in Q$.* $\square$

Theorem 21 (cf. [9, Corollary 3.4]). *$\mathcal{L}((M/\mathcal{R})) = \mathcal{L}(M)$.* $\square$

The coarsest of all forward bisimulations on M yields the smallest aggregated nta. This nta cannot be reduced further by collapsing it with respect to some forward bisimulation.

Theorem 22. *There exists a coarsest forward bisimulation $\mathcal{P}$ on M, and the identity is the only forward bisimulation on $(M/\mathcal{P})$.* $\square$

Minimisation algorithm. We now modify the algorithm of Sect. 3 so as to minimise with respect to forward bisimulation. As in Sect. 3 this requires us to extend our notation. We denote by C_Q^k the of set of *contexts* over Q: the set of k-tuples over $Q \cup \{\Box\}$ that contain the special symbol $\Box$ exactly once. We denote by $c[\![q]\!]$, where $c \in C_Q^k$ and $q \in Q$, the tuple that is obtained by replacing the unique occurrence of $\Box$ in c by q.

Definition 23. *For each state q in Q and $k \in \mathbb{N}$, the map $\mathrm{obsf}_q^k \colon \Sigma_{(k)} \times C_Q^k \times \mathfrak{P}(Q) \to \mathbb{N}$ is defined by $\mathrm{obsf}_q^k(f,c,D) = |\{q' \in D \mid f(c[\![q]\!]) \xrightarrow{\delta} q'\}|$ for every symbol $f \in \Sigma_{(k)}$, context $c \in C_Q^k$, and set $D \subseteq Q$ of states.* $\square$

Like obs_q^k, obsf_q^k is a local observation of the properties of q. The difference here, is that $\mathrm{obsf}_q^k(f, c, D)$ is the number of f-transitions that match the sequence $c[\![q]\!]$ and lead to a state of D. In contrast, obs_q^k looked from the other side of the rule.

Definition 24. *Let D and D' be subsets of Q.*

- *We write* $\mathrm{splitf}(D)$ *for the set of all pairs (q, q') in $Q \times Q$, for which there exist $f \in \Sigma_{(k)}$ and $c \in C_Q^k$ such that exactly one of $\mathrm{obsf}_q^k(f, c, D)$ and $\mathrm{obsf}_{q'}^k(f, c, D)$ is non-zero.*
- *Similarly, we write* $\mathrm{splitfn}(D, D')$ *for the set of all pairs (q, q') in $Q \times Q$, for which there exist $f \in \Sigma_{(k)}$ and $c \in C_Q^k$ such that $\mathrm{obsf}_p^k(f, c, D) = \mathrm{obsf}_p^k(f, c, D')$ holds for either $p = q$ or $p = q'$ but not both.* □

We can now construct a minimisation algorithm based on forward bisimulation by replacing the initialisation of $\mathcal{R}_0$ in Alg. 1 with $\mathcal{R}_0 = ((Q \setminus F)^2 \cup F^2) \setminus \mathrm{splitf}(Q)$ and the computation of $\mathcal{R}_{i+1}$ with $\mathcal{R}_{i+1} = \big(\mathcal{R}_i \setminus \mathrm{splitf}(B_i)\big) \setminus \mathrm{splitfn}(S_i, B_i)$.

Example 25. We show the execution of the minimisation algorithm on the nta N from Example 19. In the initialisation of $\mathcal{R}_0$, states 3 and 4 are separated because they are accepting. State 1 is distinguished as only obsf_1^2 is non-zero on the symbol f, context $(\square, 2) \in C_{[1,4]}^2$, and block Q in $\mathcal{P}_0$. We thus have the relations $\mathcal{P}_0 = Q \times Q$ and $\mathcal{R}_0 = \{1\}^2 \cup \{2\}^2 \cup \{3, 4\}^2$. As neither 3 nor 4 appear on a left-hand side of any transition, they will not be separated, so the algorithm terminates with $(M/\mathcal{R}_0)$ in the second iteration, when $\mathcal{P}_0$ has been refined to $\mathcal{R}_0$. □

Note that also the modified algorithm is correct and terminates in less than n iterations where n is the cardinality of Q.

Theorem 26. *$\mathcal{R}_t$ is the coarsest forward bisimulation on M.* □

The time complexity of the backward 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 $f(q_1 \cdots q_k) \to q$, where $q_j \in B_i$ for some $j \in [1, k]$, the forward algorithm considers only those transitions that are of the form $f(q_1 \cdots q_k) \to q$, 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 27. *The forward minimisation algorithm is in $O(rm \log n)$.* □

Next, we show that forward bisimulation minimisation coincides with classical minimisation and yields the minimal deterministic nta.

Definition 28. *We say that M is* deterministic *(respectively,* complete*), if for every symbol f in $\Sigma_{(k)}$, and sequence $(q_1, \ldots, q_k) \in Q^k$ of states there exists at most (respectively, at least) one state q in Q such that $f(q_1, \ldots, q_k) \to q$ is in δ.* □

Clearly, the automaton $(M/\mathcal{R})$ is deterministic and complete whenever M is so. Moreover, there exists a unique minimal complete and deterministic nta N that recognises the language $\mathcal{L}(M)$. The next theorem shows that N is isomorphic to $(M/\mathcal{R})$ if $\mathcal{R}$ is the coarsest forward bisimulation on M.

Theorem 29. *Let M be a deterministic and complete nta without useless states. Then $(M/\mathcal{R}_t)$ is a minimal deterministic and complete nta recognising $\mathcal{L}(M)$.*

5 Implementation

In this section, we present some experimental results that we obtained by applying a prototype implementation of Alg. 1 to the problem of *language modelling* in the natural language processing domain [11]. 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 [12,13]. Thus, the problem is elevated to determining whether a given syntactic tree is in a language. Language models are naturally representable as finite-state acceptors. 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 modelling. We instead model *n-subtrees*, fixed-size pieces of a syntactic tree.

Table 1. Reduction of states and rules using backward, forward, and AKH bisimulation

TREES	ORIGINAL		BACKWARD		FORWARD		AKH	
	states	rules	states	rules	states	rules	states	rules
58	353	353	252	252	286	341	353	353
161	953	953	576	576	749	905	953	953
231	1373	1373	781	781	1075	1299	1373	1373
287	1726	1726	947	947	1358	1637	1726	1726

Table 2. Reduction of states and rules when combining backward and forward bisimulation

TREES	ORIGINAL		BW AFTER FW		FW AFTER BW	
	states	rules	states	rules	states	rules
58	353	353	185	240	180	235
161	953	953	378	534	356	512
231	1373	1373	494	718	468	691
287	1726	1726	595	874	563	842

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 [14]. An initial nta was constructed by representing each 3-subtree in a single path. We then wrote an implementation of the forward and backward variants of Alg. 1 in Perl and applied them to data sets of various sizes of 3-subtrees. To illustrate that the two algorithms perform different minimisations, we then ran the forward algorithm on the result from the backward algorithm, and vice-versa. As Table 2 shows, 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.

Table 1 includes the state and rule count of the same automata after minimisation with respect to AKH-bisimulation. 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 16 is obtained from Definition 1 if the two-way implication in Definition 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.

6 Conclusion

We have introduced a general algorithm for bisimulation minimisation 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 non-deterministic tree automata. We plan to include a refined implementation of this algorithm in a future version of the tree automata toolkit described in [15].

Acknowledgements. 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 [4]. We would also like to thank the referees for extensive and useful comments. This work was partially supported by NSF grant IIS-0428020.

References

1. Hopcroft, J.E.: An $n \log n$ algorithm for minimizing states in a finite automation. In: Kohavi, Z. (ed.) Theory of Machines and Computations, Academic Press, London (1971)
2. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: Proc. 13th Annual Symp. Foundations of Computer Science, pp. 125–129. IEEE Computer Society Press, Los Alamitos (1972)
3. Gramlich, G., Schnitger, G.: Minimizing NFAs and regular expressions. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 399–411. Springer, Heidelberg (2005)

4. Abdulla, P.A., Högberg, J., Kaati, L.: Bisimulation minimization of tree automata. In: IJFCS (2007)
5. Abdulla, P.A., Jonsson, B., Mahata, P., d'Orso, J.: Regular tree model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 555–568. Springer, Heidelberg (2002)
6. Knight, K., Graehl, J.: An overview of probabilistic tree transducers for natural language processing. In: Gelbukh, A. (ed.) CICLing 2005. LNCS, vol. 3406, pp. 1–24. Springer, Heidelberg (2005)
7. Paige, R., Tarjan, R.: Three partition refinement algorithms. SIAM Journal on Computing 16, 973–989 (1987)
8. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata: Techniques and applications (1997), Available on `http://www.grappa.univ-lille3.fr/tata`
9. Buchholz, P.: Bisimulation relations for weighted automata (unpublished, 2007)
10. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, Reading (1994)
11. Jelinek, F.: Continuous speech recognition by statistical methods. Proc. IEEE 64, 532–557 (1976)
12. Galley, M., Hopkins, M., Knight, K., Marcu, D.: What's in a translation rule? In: Proc. 2004 Human Language Technology Conf. of the North American Chapter of the Association for Computational Linguistics, pp. 273–280 (2004)
13. Yamada, K., Knight, K.: A syntax-based statistical translation model. In: Proc. 39th Meeting of the Association for Computational Linguistics, pp. 523–530. Morgan Kaufmann, San Francisco (2001)
14. Marcus, M.P., Marcinkiewicz, M.A., Santorini, B.: Building a large annotated corpus of english: The Penn treebank. Computational Linguistics 19, 313–330 (1993)
15. May, J., Knight, K.: Tiburon: A weighted tree automata toolkit. In: Ibarra, O.H., Yen, H.-C. (eds.) CIAA 2006. LNCS, vol. 4094, pp. 102–113. Springer, Heidelberg (2006)