Forward Bisimulations for Nondeterministic Symbolic Finite Automata

Loris D'Antoni[1] and Margus Veanes[2]

[1] University of Wisconsin, Madison
loris@cs.wisc.edu

[2] Microsoft Research
margus@microsoft.com

Abstract. Symbolic automata allow transitions to carry predicates over rich alphabet theories, such as linear arithmetic, and therefore extend classic automata to operate over infinite alphabets, such as the set of rational numbers. Existing automata algorithms rely on the alphabet being finite, and generalizing them to the symbolic setting is not a trivial task. In our earlier work, we proposed new techniques for minimizing deterministic symbolic automata and, in this paper, we generalize these techniques and study the foundational problem of computing forward bisimulations of nondeterministic symbolic finite automata. We propose three algorithms. Our first algorithm generalizes Moore's algorithm for minimizing deterministic automata. Our second algorithm generalizes Hopcroft's algorithm for minimizing deterministic automata. Since the first two algorithms have quadratic complexity in the number of states and transitions in the automaton, we propose a third algorithm that only requires a number of iterations that is linearithmic in the number of states and transitions at the cost of an exponential worst-case complexity in the number of distinct predicates appearing in the automaton. We implement our algorithms and evaluate them on 3,625 nondeterministic symbolic automata from real-world applications.

1 Introduction

Finite automata are used in many applications in software engineering, including software verification [8] and text processing [3]. Despite their many applications, finite automata suffer from a major drawback: in the most common forms they can only handle finite and small alphabets. Symbolic automata allow transitions to carry predicates over a specified alphabet theory, such as linear arithmetic, and therefore extend finite automata to operate over infinite alphabets, such as the set of rational numbers [13]. Symbolic automata are therefore more general and succinct than their finite-alphabet counterparts. Traditional algorithms for finite automata do not always generalize to the symbolic setting, making the design of algorithms for symbolic automata challenging. A notable example appears in [11]: while allowing finite state automata transitions to read multiple adjacent inputs does not add expressiveness, in the symbolic case this extension makes problems such as checking equivalence undecidable.

Symbolic finite automata (s-FA) are closed under Boolean operations and enjoy decidable equivalence if the alphabet theory forms a decidable Boolean algebra [13]. s-FAs have been used in combination with symbolic transducers to analyze complex string and list-manipulating programs [12, 16]. In these applications it is crucial to keep the automata "small" and, in our previous work, we proposed algorithms for minimizing deterministic s-FAs [13]. However, no algorithms have been proposed to reduce the state space of nondeterministic s-FAs (s-NFAs). While computing minimal nondeterministic automata is a hard problem [18], several techniques have been proposed to produce "small enough" automata. These algorithms compute bisimulations over the state space and use them to collapse bisimilar states [26, 2]. In this paper, we study the problem of computing forward bisimulations for s-NFAs.

While the problem of computing forward bisimulations has been studied for classic NFAs, it is not easy to adapt these algorithms to s-NFAs. Most efficient automata algorithms view the size of the alphabet as a constant and use data structures that are optimized for this view [2]. We propose three new algorithms for computing forward bisimulation of s-NFAs. First, we extend the classic Moore's algorithm for minimizing deterministic finite automata [25] and define an algorithm that operates in quadratic time. We then adapt our previous algorithm for minimizing deterministic s-FAs [13] to the problem of computing forward bisimulations and show that a natural implementation leads to a quadratic running time algorithm. Finally, we adapt a technique proposed by Abdulla et al. [2] to our setting, and propose a new symbolic data-structure that allows us to perform only a number of iterations that is linearithmic in the number of states and transitions. However, this improved state complexity comes at the cost of an exponential complexity in the number of distinct predicates appearing in the automaton. We compare the performance of the three algorithms on 3,625 s-FAs obtained from regular expressions and NFAs appearing in verification applications and show that, unlike for the case of deterministic s-FAs, no algorithm strictly outperforms the other ones.

Contributions. In summary, our contributions are:

- a formal study of the notion of forward bisimulations for s-FAs and their relation to state reduction for nondeterministic s-FAs (§ 3);
- three algorithms for computing forward bisimulations (§ 4, 5 and 6);
- an implementation and a comprehensive evaluation of the algorithms on 3,625 s-FAs obtained from real-world applications (§ 7).

2 Effective Boolean algebras and s-NFAs

We define the notion of effective Boolean algebra and symbolic finite automata. An *effective Boolean algebra* $\mathcal{A}$ has components $(U, \Psi, [\![\_]\!], \bot, \top, \vee, \wedge, \neg)$. U is a set called the *universe*. Ψ is a set of *predicates* closed under the Boolean connectives and $\bot, \top \in \Psi$. The *denotation function* $[\![\_]\!] : \Psi \to 2^U$ is such that, $[\![\bot]\!] = \emptyset$,

$[\![\top]\!] = U$, for all $\varphi, \psi \in \Psi$, $[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$, $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$, and $[\![\neg\varphi]\!] = U \setminus [\![\varphi]\!]$. For $\varphi \in \Psi$, we write $\mathbf{SAT}(\varphi)$ when $[\![\varphi]\!] \neq \emptyset$ and say that φ is *satisfiable*. $\mathcal{A}$ is *decidable* if $\mathbf{SAT}$ is decidable.

Intuitively, such an algebra is represented programmatically as an API with corresponding methods implementing the Boolean operations and the denotation function. We are primarily going to use the following two effective Boolean algebras in the examples, but the techniques in the paper are fully generic.

$\mathbf{2^{bv\mathit{k}}}$ is the powerset algebra whose domain is the finite set $\textsc{bv}k$, for some $k > 0$, consisting of all non-negative integers smaller than 2^k—i.e., all k-bit bit-vectors. A predicate is represented by a Binary Decision Diagram (BDD) of depth k.[3] Boolean operations correspond directly to BDD operations and $\perp$ is the BDD representing the empty set. The denotation $[\![\beta]\!]$ of a BDD β is the set of all integers n such that a binary representation of n corresponds to a solution of β.

int[k] is an algebra for small finite alphabets of the form $\Sigma = \{0, \ldots, 32k - 1\}$. A predicate φ is an array of k unsigned 32-bit integers, $\varphi = [a_1, \ldots, a_k]$, and for all $i \in \Sigma$: $i \in [\![\varphi]\!]$ iff in the integer $a_{i/32+1}$ the bit in position i mod 32 is 1. Boolean operations can be performed efficiently using bit-vector operations. For example, the conjunction $[a_1, \ldots, a_k] \wedge [b_1, \ldots, b_k]$ corresponds to $[a_1 \& b_1, \ldots, a_k \& b_k]$, where & is the bit-wise and of two integers.

We can now define symbolic finite automata. Intuitively, a symbolic finite automaton is a finite automaton over a symbolic alphabet, where edge labels are replaced by predicates. In order to preserve the classical Boolean closure operations (intersection, complement, and union) over languages, the predicates must also form a Boolean algebra. Since the core topic of the paper is about *nondeterministic* automata we adopt the convention often used in studies of NFAs [22, 10, 28] that an automaton has a *set* of initial states rather than a single initial state as used in other literature on automata theory [21].

Definition 1. A *symbolic nondeterministic finite automaton* (*s-NFA*) M is a tuple $(\mathcal{A}, Q, I, F, \Delta)$ where $\mathcal{A}$ is an effective Boolean algebra, called the *alphabet*, Q is a finite set of *states*, $I \subseteq Q$ is the set of *initial states*, $F \subseteq Q$ is the set of *final states*, and $\Delta \subseteq Q \times \Psi_{\mathcal{A}} \times Q$ is a finite set of *moves* or *transitions*.

Elements of $U_{\mathcal{A}}$ are called *characters* and finite sequences of characters, elements of $U_{\mathcal{A}}^*$, are called *words*; ϵ denotes the empty word. A move $\rho = (p, \varphi, q) \in \Delta$ is also denoted by $p \xrightarrow{\varphi}_M q$ (or $p \xrightarrow{\varphi} q$ when M is clear from the context), where p is the *source* state, q is the *target* state, and φ is the *guard* or *predicate* of the move. Given a character $a \in U_{\mathcal{A}}$, an *a-move* of M is a tuple (p, a, q) such that $p \xrightarrow{\varphi}_M q$ and $a \in [\![\varphi]\!]$, also denoted $p \xrightarrow{a}_M q$ (or $p \xrightarrow{a} q$ when M is clear). In the following let $M = (\mathcal{A}, Q, I, F, \Delta)$ be an s-NFA.

[3] Let the variable order of the BDD be the reverse bit order of the binary representation of a number, i.e., the most significant bit has the lowest ordinal, etc.

Definition 2. Given a state $p \in Q$, the *(right) language of* p *in* M, denoted $\mathscr{L}(p, M)$, is the set of all $w = [a_i]_{i=1}^k \in U_{\mathcal{A}}^*$ such that, either $w = \epsilon$ and $p \in F$, or $w \neq \epsilon$ and there exist $p_{i-1} \xrightarrow{a_i}_M p_i$ for $1 \leq i \leq k$, such that $p_0 = p$, and $p_k \in F$. The *language of* M is $\mathscr{L}(M) \overset{\text{def}}{=} \bigcup_{q \in I} \mathscr{L}(q, M)$. Two states p and q of M are *indistinguishable* if $\mathscr{L}(p, M) = \mathscr{L}(q, M)$. Two s-NFAs M and N are *equivalent* if $\mathscr{L}(M) = \mathscr{L}(N)$.

The following terminology is used to characterize various key properties of M. A state $p \in Q$ is called *complete* if for all $a \in U_{\mathcal{A}}$ there exists an a-move from p, p is *partial* otherwise. A move is *feasible* if its guard is satisfiable.

- M is *deterministic*: $|I| = 1$ and whenever $p \xrightarrow{a} q$ and $p \xrightarrow{a} q'$ then $q = q'$.
- M is *complete*: all states of M are complete; M is *partial*, otherwise.
- M is *clean*: all moves of M are feasible.
- M is *normalized*: for all $(p, \varphi, q), (p, \psi, q) \in \Delta$: $\varphi = \psi$.
- M is *minimal*: there exists no equivalent s-NFA with fewer states.

In the following, we always assume that M is clean. If E is an equivalence relation over Q, then, for $q \in Q$, $q_{/E}$ denotes the E-equivalence class containing q, for $X \subseteq Q$, $X_{/E}$ denotes $\{q_{/E} \mid q \in X\}$. The *E-quotient* of M is the s-NFA

$$M_{/E} \overset{\text{def}}{=} (\mathcal{A}, Q_{/E}, I_{/E}, F_{/E}, \{(p_{/E}, \varphi, q_{/E}) \mid (p, \varphi, q) \in \Delta\})$$

3 Forward bisimulations

Here we adapt the notion of forward bisimulation to s-NFAs. Below, consider a fixed s-NFA $M = (\mathcal{A}, Q, I, F, \Delta)$.

Definition 3. Let $E \subseteq Q \times Q$ be an equivalence relation. E is a *forward bisimulation on* M when, for all $(p, q) \in E$, if $p \in F$ then $q \in F$, and, for all $a \in U_{\mathcal{A}}$ and $p' \in Q$, if $p \xrightarrow{a} p'$ then there exists $q' \in p'_{/E}$ such that $q \xrightarrow{a} q'$.

If E is a forward bisimulation on M then the quotient $M_{/E}$ preserves the language of all states in M, as stated formally by Theorem 1, as a generalization of the same property known in the classical case when the alphabet is finite.

Theorem 1. *Let E be a forward bisimulation on M. Then, for all states q of M, $\mathscr{L}(q, M) = \mathscr{L}(q_{/E}, M_{/E})$.*

Proof. We prove the statement $\phi(w)$ by induction over $|w|$ for $w \in U_{\mathcal{A}}^*$:

$$\phi(w) : \forall p \in Q_M (w \in \mathscr{L}(p, M) \Leftrightarrow w \in \mathscr{L}(p_{/E}, M_{/E}))$$

The base case $|w| = 0$ follows from the property of the forward bisimulation E on M that if $p \in F$ then $p_{/E} \subseteq F$ and by definition of E-quotient of M that its set of final states is $F_{/E}$.

For the induction case assume that $\phi(w)$ holds as the IH. Let $a \in U_{\mathcal{A}}$. We prove $\phi(a \cdot w)$. Fix $p \in Q_M$.

$$\begin{aligned}
a \cdot w \in \mathscr{L}(p, M) \;&\Leftrightarrow\; \exists q \in Q \text{ such that } (p \xrightarrow{a}_M q,\ w \in \mathscr{L}(q, M)) \\
&\overset{\text{by IH}}{\Leftrightarrow}\; \exists q \in Q \text{ such that } (p \xrightarrow{a}_M q,\ w \in \mathscr{L}(q_{/E}, M_{/E})) \\
&\overset{(*)}{\Leftrightarrow}\; \exists q \in Q \text{ such that } (p_{/E} \xrightarrow{a}_{M_{/E}} q_{/E},\ w \in \mathscr{L}(q_{/E}, M_{/E})) \\
&\Leftrightarrow\; a \cdot w \in \mathscr{L}(p_{/E}, M_{/E})
\end{aligned}$$

Proof of $(*)$:
($\Rightarrow$): If $p \xrightarrow{a}_M q$ then there is $(p, \varphi, q) \in \Delta_M$ such that $a \in \llbracket \varphi \rrbracket$. By definition of $M_{/E}$, there is $(p_{/E}, \varphi, q_{/E}) \in \Delta_{M_{/E}}$, hence $p_{/E} \xrightarrow{a}_{M_{/E}} q_{/E}$.
($\Leftarrow$): Fix a q such that $p_{/E} \xrightarrow{a}_{M_{/E}} q_{/E}$ and $w \in \mathscr{L}(q_{/E}, M_{/E})$. By definition of $\Delta_{M_{/E}}$ there exists a transition (p_1, α, q_1) in Δ_M where $a \in \llbracket \alpha \rrbracket$ and $p_{1/E} = p_{/E}$ and $q_{1/E} = q_{/E}$, so $p_1 \xrightarrow{a}_M q_1$. By the assumption that E is a bisimulation on M it follows that there exists $q' \in q_{1/E}$ such that $p \xrightarrow{a}_M q'$. But $q_{1/E} = q_{/E}$, so $q'_{/E} = q_{/E}$ and therefore $\exists q' \in Q$ such that $(p \xrightarrow{a}_M q',\ w \in \mathscr{L}(q'_{/E}, M_{/E}))$. $\boxtimes$

Corollary 1. *Let E be a forward bisimulation on M. Then $\mathscr{L}(M) = \mathscr{L}(M_{/E})$.*

For a deterministic s-NFA M one can efficiently compute the *coarsest* forward bisimulation relation $\equiv_M$ over Q_M defined by indistinguishability of states, in order to construct $M_{/\equiv_M}$ as the minimal canonical (up to equivalence of predicates) deterministic s-NFA that is equivalent to M [13, Theorem 2]. The nondeterministic case is much more difficult because there exists, in general, no canonical minimal NFA [22] for a given regular language.

Our aim in this paper is to study algorithms for computing forward bisimulations for s-NFAs. Once a forward bisimulation E has been computed for an s-NFA M, it can be applied, according to Corollary 1, to build the equivalent E-quotient $M_{/E}$ with reduced number of states, $M_{/E}$ need not be minimal though.

4 Symbolic partition refinement

We start by presenting the high-level idea of symbolic partition refinement for forward bisimulations as an abstract algorithm. Let the given s-NFA be $M = (\mathcal{A}, Q, I, F, \Delta)$. It is convenient to view Δ, without loss of generality, as a function from $Q \times Q$ to $\Psi_{\mathcal{A}}$, and we also lift the definition over its second argument to subsets $S \subseteq Q$ of states,

$$\Delta(p, q) \overset{\text{def}}{=} \bigvee_{(p, \varphi, q) \in \Delta} \varphi, \quad \Delta(p, S) \overset{\text{def}}{=} \bigvee_{q \in S} \Delta(p, q),$$

where the predicates are effectively constructed using $\vee_{\mathcal{A}}$. Essentially, this view of Δ corresponds to M being normalized, where all pairs (p, q) such that there is no transition from p to q have $\Delta(p, q) = \bigvee \emptyset \overset{\text{def}}{=} \bot$, else the guard of the transition from p to q is $\Delta(p,q)$. The predicate $\Delta(p,S)$ denotes the set of all those characters that transition from p to some state in S.

M is assumed to be nontrivial, so that both F and $Q\backslash F$ are nonempty. We construct partitions $\mathcal{P}_i$ of Q such that $\mathcal{P}_i$ is a *refinement* of $\mathcal{P}_{i-1}$ for $i \geq 1$, i.e., each block in $\mathcal{P}_i$ is a subset of some block in $\mathcal{P}_{i-1}$. Initially let

$$\mathcal{P}_0 = \{Q\},\ \mathcal{P}_1 = \{F, Q\backslash F\}.$$

For a partition $\mathcal{P}$ of Q define $\sim_{\mathcal{P}}$ as the following equivalence relation over Q:

$$p \sim_{\mathcal{P}} q \overset{\text{def}}{=} \exists B \in \mathcal{P} \text{ such that } (p, q \in B).$$

Let $\sim_i \overset{\text{def}}{=} \sim_{\mathcal{P}_i}$. The partition $\mathcal{P}_i$ is refined until $\mathcal{P}_{n+1} = \mathcal{P}_n$ for some $n \geq 1$. Each such refinement step maintains the invariant (1) for $i \geq 1$ and $p, q \in Q$:[4]

$$p \sim_{i+1} q \Longleftrightarrow p \sim_i q \text{ and for all } B \in \mathcal{P}_i\text{: } [\![\Delta(p,B)]\!] = [\![\Delta(q,B)]\!] \qquad (1)$$

Under the assumption that $\mathcal{A}$ is decidable, $[\![\Delta(p,B)]\!] = [\![\Delta(q,B)]\!]$ can be decided by checking that $\Delta(p,B) \not\Leftrightarrow \Delta(q,B)$ is *unsatisfiable*.[5] So $\mathcal{P}_{i+1}$ can be computed effectively from $\mathcal{P}_i$ and iterating this step provides an abstract algorithm for computing the fixpoint $\sim_M \overset{\text{def}}{=} \sim_{\mathcal{P}_n}$ such that $\mathcal{P}_{n+1} = \mathcal{P}_n$.

Theorem 2. *$\sim_M$ is the coarsest forward bisimulation on M.*

Proof. Let $\sim\ = \ \sim_M$. We show first that $\sim$ is a forward bisimulation on M by way of contradiction. Suppose that $\sim$ is not a forward bisimulation on M. Since $p \sim_1 q$ iff $p, q \in F$ or $p, q \notin F$, and $\sim$ refines $\sim_1$, the condition that for $p \sim q$ if $p \in F$ then $q \in F$ holds. Therefore, there must exists $p \sim q$ such that for some $a \in \mathcal{U}_{\mathcal{A}}$ and $p' \in Q$ we have $p \xrightarrow{a} p'$, while for all q' such that $q \xrightarrow{a} q'$ we have $q' \nsim p'$. Hence there is $B \in \mathcal{P}_i$ for some $i \geq 1$, namely $B = p'_{/\sim}$, such that $a \in [\![\Delta(p,B)]\!]$ but $a \notin [\![\Delta(q,B)]\!]$, so $[\![\Delta(p,B)]\!] \neq [\![\Delta(q,B)]\!]$. But then $p \nsim_{i+1} q$, contradicting that $p \sim q$. So $\sim$ is a forward bisimulation on M.

Next, consider any bisimulation $\simeq$ on M. We show that $\simeq\ \subseteq\ \sim_i$ for all $i \geq 1$.

Base case. Suppose $p \simeq q$. If $p \in F$ then $q \in F$, by Definition 3, and, since $\simeq$ is an equivalence relation, symmetrically, if $p \notin F$ then $q \notin F$. So $p \sim_1 q$.

Induction case. Assume as the IH that $\simeq\ \subseteq\ \sim_i$. We prove that $\simeq\ \subseteq\ \sim_{i+1}$. Suppose $p \simeq q$. We show that $p \sim_{i+1} q$. By using the IH, we have that $p \sim_i q$. By using Equation (1), we need to show that for all $B \in \mathcal{P}_i$, $[\![\Delta(p,B)]\!] = [\![\Delta(q,B)]\!]$. By way of contradiction, suppose there exists $B \in \mathcal{P}_i$ such that $[\![\Delta(p,B)]\!] \neq [\![\Delta(q,B)]\!]$. Then, w.l.o.g., there exists $a \in \mathcal{U}_{\mathcal{A}}$ and $p' \in B$ such that $p \xrightarrow{a} p'$, and for all $q' \in Q$ if $q \xrightarrow{a} q'$ then $q' \notin B$, i.e., $q' \nsim_i p'$, and by using the contrapositive of the IH ($\nsim_i\ \subseteq\ \not\simeq$) we have $q' \not\simeq p'$. But then $p \xrightarrow{a} p'$ while there is no $q' \in p'_{/\simeq}$ such that $q \xrightarrow{a} q'$, contradicting, by Definition 3, that $p \simeq q$. Thus, for all $B \in \mathcal{P}_i$, $[\![\Delta(p,B)]\!] = [\![\Delta(q,B)]\!]$. So $p \sim_{i+1} q$.

It follows that $\simeq\ \subseteq\ \sim$ which proves that $\sim$ is coarsest. ☒

```
$SimpleBisimSFA(M = (\mathcal{A}, Q, I, F, \varDelta)) \stackrel{\text{def}}{=}$
  $\mathcal{P} := \{F,\ Q\backslash F\}$   //initial partition
  $W := \{F,\ Q\backslash F\}$   //workset
  while $(W \neq \emptyset)$
    pull $R$ from $W$   //choose a splitter candidate
    while (exists $B$ in $\mathcal{P}$ and $q, r$ in $B$ such that $\mathbf{SAT}(\varDelta(q, R) \wedge \neg\varDelta(r, R))$)
      let $D = \{p \in B \mid \mathbf{SAT}(\varDelta(p, R) \wedge \varDelta(q, R) \wedge \neg\varDelta(r, R))\}$
      $\mathcal{P} := (\mathcal{P} \setminus \{B\}) \cup \{D, B\backslash D\}$   //refine the partition
      $W := (W\backslash\{B\}) \cup \{D, B\backslash D\}$   //update the workset
  return $\sim_{\mathcal{P}}$
```

```
$GreedyBisimSFA(M = (\mathcal{A}, Q, I, F, \varDelta)) \stackrel{\text{def}}{=}$
  $\mathcal{P} := \{F, Q\backslash F\}$   //initial partition
  $W := \{$if $(|F| \leq |Q\backslash F|)$ then $F$ else $Q\backslash F\}$   //workset
  $\textsc{super}(F) := Q$; $\textsc{super}(Q\backslash F) := Q$   //$\textsc{super}(B)$ is the superblock of $B$
  while $(W \neq \emptyset)$
    pull $R$ from $W$   //choose a splitter candidate
    let $R' = \textsc{super}(R)\backslash R$
    while (exists $B$ in $\mathcal{P}$ and $q, r$ in $B$ such that
      $\mathbf{SAT}(\varDelta(q, R) \wedge \neg\varDelta(r, R))$ or $\mathbf{SAT}(\varDelta(q, R') \wedge \neg\varDelta(r, R'))$)
      let $D =$ if $\mathbf{SAT}(\varDelta(q, R) \wedge \neg\varDelta(r, R))$
              then $\{p \in B \mid \mathbf{SAT}(\varDelta(p, R) \wedge \varDelta(q, R) \wedge \neg\varDelta(r, R))\}$
              else $\{p \in B \mid \mathbf{SAT}(\varDelta(p, R') \wedge \varDelta(q, R') \wedge \neg\varDelta(r, R'))\}$
      $\mathcal{P} := (\mathcal{P}\backslash\{B\}) \cup \{D, B\backslash D\}$   //refine $\mathcal{P}$
      if $(B \in W)$ then   //add both parts into the workset
        $W := (W\backslash\{B\}) \cup \{D, B\backslash D\}$
        $\textsc{super}(D) := \textsc{super}(B)$;   //$\textsc{super}(B)$ remains the superblock of $B$ parts
        $\textsc{super}(B\backslash D) := \textsc{super}(B)$
      else   //add only the smaller of the two parts into the workset
        $W := W \cup \{$if $(|D| \leq |B\backslash D|)$ then $D$ else $B\backslash D\}$
        $\textsc{super}(D) := B$;   //$B$ becomes the superblock of both parts
        $\textsc{super}(B\backslash D) := B$
  return $\sim_{\mathcal{P}}$
```

Fig. 1. Simple and greedy algorithms for computing $\sim_M$.

A simple algorithm for computing $\sim_M$ is shown in Figure 1. It differs from the abstract algorithm in that the partition is refined in smaller increments, rather than in large parallel refinement steps corresponding to Equation (1). The order of such steps does not matter as long as progress is made at each step.

Theorem 3. *SimpleBisimSFA*(M) *computes* $\sim_M$.

[4] One can view one iteration of refinement from $\mathcal{P}_i$ to $\mathcal{P}_{i+1}$ as computing $\nsim_{i+1}$ from $\nsim_i$, which is often how Moore's algorithm is presented for DFAs.

[5] $\varphi \Leftrightarrow \psi$ is defined as $((\varphi \vee \neg\psi) \wedge (\neg\varphi \vee \psi))$ and $\varphi \not\Leftrightarrow \psi$ stands for $\neg(\varphi \Leftrightarrow \psi)$.

Proof (outline). The key observation is the following: if $[\![\Delta(q, B)]\!] \neq [\![\Delta(r, B)]\!]$ holds for some $q \sim_{\mathcal{P}} r$ and $B \in \mathcal{P}$ and B has been split into $\{B_i\}_{i=1}^{n}$ before it has been chosen from the workset then $[\![\Delta(q, B_i)]\!] \neq [\![\Delta(r, B_i)]\!]$ for some i, or else $[\![\Delta(q, B)]\!] = \bigcup_i [\![\Delta(q, B_i)]\!] = \bigcup_i [\![\Delta(r, B_i)]\!] = [\![\Delta(r, B)]\!]$. In other words, even if B has not yet been used as a splitter, the fact that $q \nsim_M r$ holds will be detected at some later point using one of the blocks B_i because all subblocks are added to the workset W.

The splitting of B into D and $B \backslash D$ requires some explanation. First note that $q \in D$ and $r \in B \backslash D$, so both new blocks are nonempty. Second, pick any $p \in D$ and any $s \in B \backslash D$. We need to show that $[\![\Delta(p, R)]\!] \neq [\![\Delta(s, R)]\!]$ to justify the split. We know that $\mathbf{SAT}(\Delta(p, R) \wedge \Delta(q, R) \wedge \neg\Delta(r, R))$ holds. Thus, if $\Delta(p, R)$ were equivalent to $\Delta(s, R)$ then $\mathbf{SAT}(\Delta(s, R) \wedge \Delta(q, R) \wedge \neg\Delta(r, R))$ would also hold, contradicting that $s \notin D$.

It follows that upon termination, when $W = \emptyset$, $\mathcal{P}$ cannot be refined further and thus $\sim_{\mathcal{P}} = \sim_M$. ⊠

Complexity. If the complexity of checking satisfiability of predicates of size ℓ is $f(\ell)$, then *SimpleBisimSFA*(M) has complexity $\mathcal{O}(mnf(n\ell))$, where m is the number of transitions in the input s-FA, n is the number of states, and ℓ is the size of the largest predicate in the input s-FA.[6] Since we check satisfiability by taking the union of all predicates in multiple transition (e.g., $\Delta(q, R)$), satisfiability checks are performed on predicates of size $\mathcal{O}(n\ell)$.

5 Greedy symbolic partition refinement

We can improve the simple algorithm by incorporating Hoprcoft's "keep the smaller half" partition refinement strategy [19]. This strategy is also reused in Paige-Tarjan's relational coarsest partition algorithm [26]. Hopcroft's strategy is generalized to symbolic alphabets in [13] by incorporating the idea of using symmetric differences of character predicates during partition refinement, instead of single characters, as illustrated also in the simple algorithm. Here we further generalize the algorithm from [13] to s-NFAs. The algorithm can also be seen as a generalization of Paige-Tarjan's relational coarsest partition algorithm from computing the coarsest forward bisimulation of an NFA to that of an s-NFA.

The greedy algorithm is shown in Figure 1. The computation of partition $\mathcal{P}$ is altered in such a way that whenever a block B (that is no longer, or never was, in the workset W) is split into D and $B \backslash D$, only the smaller of the two halves is added to the workset. In order to preserve correctness, the original **SAT** condition involving R must be augmented with a corresponding condition involving $R' = \textsc{super}(R) \backslash R$, where $\textsc{super}(R)$ is the block that contained R before splitting. This means that the other half will also participate in the splitting process. The gain is how efficiently the information computed for a block is reused in the computation. The core difference to the deterministic case [13] is that if M is

[6] This bound is obtained using the same amortized complexity argument used for Moore's minimization algorithm [25].

deterministic then the use of R' is redundant, i.e., the **SAT** check holds for R iff it holds for $\textsc{super}(R)\backslash R$, so the superblock mapping is not needed.

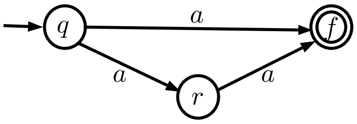


Fig. 2. Sample NFA.

Example 1. This example illustrates why the additional **SAT**-checks on $\textsc{super}(R)\backslash R$ are needed in the greedy algorithm, when M is nondeterministic. Let M be the NFA in Figure 2, where $\mathcal{U}_\mathcal{A} = \{a\}$. Then initially $W = \{\{f\}\}$ and $\mathcal{P} = \{\{q, r\}, \{f\}\}$. So, in the first iteration $R = \{f\}$. Let $R' = \textsc{super}(R)\backslash R = \{q, r\}$. The only candidate block for B is $\{q, r\}$. $\mathbf{SAT}(\Delta(q, R) \wedge \neg\Delta(r, R))$ fails because $[\![\Delta(q, R)]\!] = [\![\Delta(r, R)]\!] = \{a\}$, while $[\![\Delta(q, R')]\!] = \{a\}$ and $[\![\Delta(r, R')]\!] = \emptyset$. Thus, if $\mathbf{SAT}(\Delta(q, R') \wedge \neg\Delta(r, R'))$ was omitted then the algorithm would return $\sim_{\{\{q,r\},\{f\}\}}$ but $q \nsim_M r$. ⊠

Theorem 4. *GreedyBisimSFA*(M) *computes* $\sim_M$.

Proof (outline). The justification behind splitting of B into D and $B\backslash D$ based on R or $\textsc{super}(R)\backslash R$ is analogous to the argument provided in the proof of Theorem 3. We show that no splits are missed due to the additional optimization.

In the case a block B in W has not yet been used as a splitter, its original superblock $B^s = \textsc{super}(B)$ must be kept as the superblock of the new subblocks D and $B\backslash D$. This implies that blocks $B^s\backslash D$ and $B^s\backslash(B\backslash D)$ serve as the replacement candidate splitters for the block $B^s\backslash B$. In the case a block B is not in W, its use as a splitter is already covered, and it serves as the superblock for its subblocks D and $B\backslash D$, i.e., $\textsc{super}(D) = B$ and $\textsc{super}(B\backslash D) = B$, which implies that $\textsc{super}(D)\backslash D = B\backslash D$ and $\textsc{super}(B\backslash D)\backslash(B\backslash D) = D$. ⊠

Complexity. If the complexity of checking satisfiability of predicates of size ℓ is $f(\ell)$, the naive implementation of *GreedyBisimSFA*(M) presented in Fig. 1, which explicitly computes $\Delta(r, \textsc{super}(R)\backslash R)$, has complexity $\mathcal{O}(mnf(n\ell))$, with m as the number of transitions in the input s-FA and n as the number of states. Even though only the small block is added to added to W after a split, both blocks are eventually visited. Therefore, we still have a quadratic complexity as n and m are multiplied. In the next section, we discuss a different data structure that yields a different complexity for the greedy algorithm in Figure 1.

6 Counting symbolic partition refinement

We want to avoid explicit computation of $\Delta(p, \textsc{super}(R)\backslash R)$ in the greedy algorithm. We investigate a method that can reuse the computation performed for $\textsc{super}(R)$ and R in order to calculate $\Delta(p, \textsc{super}(R)\backslash R)$. We consider a *symbolic bag* datastructure that, by using predicates in $\Psi_\mathcal{A}$, provides a finite partition for $\mathcal{U}_\mathcal{A}$ and maps each part in the partition into a natural number. A (symbolic) bag σ denotes a function $[\![\sigma]\!]$ from $\mathcal{U}_\mathcal{A}$ to $\mathbb{N}$ that has a *finite* range. All elements that map to the same number effectively define a part or block of the partition. For $p \in Q$ and $S \subseteq Q$ let $\mathit{Bag}(p, S)$ be a bag such that, for all $a \in U_{\mathcal{A}}$,

$$[\![\mathit{Bag}(p, S)]\!](a) = |\{q \in S \mid p \xrightarrow{a} q\}|.$$

In other words, in addition to encoding if a character a can reach S from p, the bag also encodes, to *how many different target states*. Let *Set* be a function that transforms bags σ to predicates in $\Psi_{\mathcal{A}}$ such that

$$[\![\mathit{Set}(\sigma)]\!] = \{a \in U_{\mathcal{A}} \mid [\![\sigma]\!](a) > 0\}$$

In particular $[\![\mathit{Set}(\mathit{Bag}(p, S))]\!] = [\![\Delta(p, S)]\!]$. A bag can be implemented effectively in several ways and we defer the discussion of such choices to below. We assume that there is an effective difference operation $\sigma \dot{-} \tau$ over bags such that, for all $a \in U_{\mathcal{A}}$, given $m \dot{-} n \stackrel{\text{def}}{=} \max(0, m - n)$, $[\![\sigma \dot{-} \tau]\!](a) = [\![\sigma]\!](a) \dot{-} [\![\tau]\!](a)$. So

$$[\![\Delta(p, \textsc{super}(R) \backslash R)]\!] = [\![\mathit{Set}(\mathit{Bag}(p, \textsc{super}(R)) \dot{-} \mathit{Bag}(p, R))]\!].$$

This shows that each $\Delta(p, X)$ in the greedy algorithm can be represented using a symbolic bag. The potential advantage is, provided that we can efficiently implement the difference and the *Set* operations, that in the computation of $\mathit{Bag}(p, \textsc{super}(R)) \dot{-} \mathit{Bag}(p, R)$ we can reuse the prior computations of $\mathit{Bag}(p, \textsc{super}(R))$ and $\mathit{Bag}(p, R)$, and therefore do not need $\textsc{super}(R) \backslash R$.

We call the instance of the greedy algorithm that uses symbolic bags, the *counting* algorithm or *CountingBisimSFA*. The counting algorithm is a generalization of the bisimulation based minimization algorithm of NFAs [2] from using algebraic decision diagrams (ADDs) [4] and binary decision diagrams (BDDs) [9] for representing multisets ands sets of characters, to symbolic bags and predicates. If the size of the alphabet is $k = 2^p$ then p is the depth or the number of bits required in the ADDs. An open problem for symbolic bags is to maintain an equally efficient data structure. Although theoretically p is bounded by the number of predicates in the s-NFA, the actual computation of those bits and their relationship to the predicates of the s-NFA requires that the s-NFA is first transformed into an NFA. However, the NFA transformation has complexity $O(2^p)$. This factor is also reflected in the complexity of the algorithm in [2] that is $O(km \log n)$ with k, m and n as above.

Implementation. We define symbolic bags over $\mathcal{A}$, denoted $\mathit{Bag}_{\mathcal{A}}$, as the least set of expressions that satisfies the following conditions.

- If $n \in \mathbb{N}$ then $\boldsymbol{nat}(n) \in \mathit{Bag}_{\mathcal{A}}$.
- If $\varphi \in \Psi_{\mathcal{A}}$ and $\sigma, \tau \in \mathit{Bag}_{\mathcal{A}}$ then $\boldsymbol{ite}(\varphi, \sigma, \tau) \in \mathit{Bag}_{\mathcal{A}}$.

The denotation of a bag σ is a function $[\![\sigma]\!] : U_{\mathcal{A}} \to \mathbb{N}$ such that, for all $a \in U_{\mathcal{A}}$,

$$[\![\boldsymbol{nat}(n)]\!](a) \stackrel{\text{def}}{=} n, \quad [\![\boldsymbol{ite}(\varphi, \sigma, \tau)]\!](a) \stackrel{\text{def}}{=} \begin{cases} [\![\sigma]\!](a), \text{ if } a \in [\![\varphi]\!]; \\ [\![\tau]\!](a), \text{ otherwise.} \end{cases}$$

We say that a symbolic bag is *clean* if all paths from the root to any of its leaves is satisfiable. In our operations over bags we maintain cleanness. An operator $\diamond$, such as $+$ or $\dot{-}$, over $\mathbb{N}$ is lifted to bags as follows.

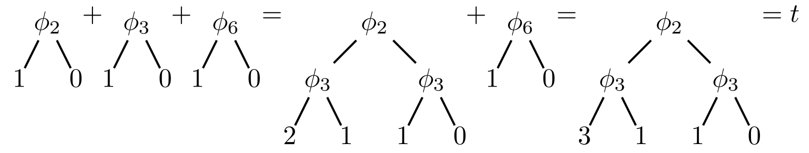
$$\begin{aligned}
\sigma \diamond \tau &\overset{\text{def}}{=} \sigma \diamond_{\top} \tau \\
\boldsymbol{nat}(m) \diamond_{\gamma} \boldsymbol{nat}(n) &\overset{\text{def}}{=} \boldsymbol{nat}(m \diamond n) \\
\boldsymbol{ite}(\varphi, \sigma, \tau) \diamond_{\gamma} \rho &\overset{\text{def}}{=} \boldsymbol{ite}(\varphi, \sigma \diamond_{\gamma\wedge\varphi} \rho, \tau \diamond_{\gamma\wedge\neg\varphi} \rho) \\
\boldsymbol{nat}(n) \diamond_{\gamma} \boldsymbol{ite}(\varphi, \sigma, \tau) &\overset{\text{def}}{=} \begin{cases} \boldsymbol{nat}(n) \diamond_{\gamma} \tau, & \text{if not } \mathbf{SAT}(\gamma \wedge \varphi); \\ \boldsymbol{nat}(n) \diamond_{\gamma} \sigma, & \text{else if not } \mathbf{SAT}(\gamma \wedge \neg\varphi); \\ \boldsymbol{ite}(\varphi, \boldsymbol{nat}(n) \diamond_{\gamma\wedge\varphi} \sigma, \boldsymbol{nat}(n) \diamond_{\gamma\wedge\neg\varphi} \tau), & \text{otherwise.} \end{cases}
\end{aligned}$$

Cleaning of the result is done incrementally during construction by passing the context condition γ with the operator $\diamond_{\gamma}$. Observe that if $\alpha \wedge \beta$ is unsatisfiable (i.e., $[\![\alpha]\!] \cap [\![\beta]\!] = \emptyset$) then α implies $\neg\beta$ (i.e., $[\![\alpha]\!] \subseteq [\![\neg\beta]\!]$). For all $p, q \in Q$ let

$$Bag(p, q) \overset{\text{def}}{=} \begin{cases} \boldsymbol{ite}(\Delta(p, q), \boldsymbol{nat}(1), \boldsymbol{nat}(0)), & \text{if } \Delta(p, q) \neq \bot; \\ \boldsymbol{nat}(0), & \text{otherwise.} \end{cases}$$

Let $Bag(p, R) \overset{\text{def}}{=} \sum_{q \in R} Bag(p, q)$. One additional simplification that is performed is that if $[\![\sigma]\!] = [\![\tau]\!]$ then the expression $\boldsymbol{ite}(\varphi, \sigma, \tau)$ is simplified to σ. The $Set(\sigma)$ operation replaces each non-zero leaf in σ with $\top$ and each zero leaf in σ with $\bot$, assuming, w.l.o.g., that $\mathcal{A}$ has the corresponding operator $\boldsymbol{ite}(\varphi, \psi, \gamma)$ with the expected semantics that $[\![\boldsymbol{ite}(\varphi, \psi, \gamma)]\!] = [\![(\varphi \wedge \psi) \vee (\neg\varphi \wedge \gamma)]\!]$.

Example 2. Consider an s-NFA M with alphabet $\mathcal{A}$ such that $U_{\mathcal{A}} = \mathbb{N}$ that has the following transitions from a given state p: $\{p \xrightarrow{\phi_2} q_2, p \xrightarrow{\phi_3} q_3, p \xrightarrow{\phi_6} q_6\}$ where ϕ_k for $k \geq 1$ is a predicate such that $n \in [\![\phi_k]\!]$ iff n is divisible by k. In the following $\boldsymbol{ite}(\varphi, l, r)$ is depicted with φ as the node, l as the left subtree, and r as the right subtree. Let $R = \{q_2, q_3, q_6\}$. Then $Bag(p, R) = Bag(p, q_2) + Bag(p, q_3) + Bag(p, q_6)$ is computed as follows:



In the second addition, all the branch conditions of the leaves of the first tree, other than the first branch, become unsatisfiable with the condition ϕ_6. Only the very first branch condition $\phi_2 \wedge \phi_3$ is consistent (in this case equivalent) with ϕ_6 while $\boldsymbol{nat}(0)$ is the identity. Hence $\boldsymbol{nat}(3) = \boldsymbol{nat}(2) + \boldsymbol{nat}(1)$ in t. $\boxtimes$

Complexity. In this implementation, $\Delta(r, B)$ is represented by $Set(Bag(r, B))$, and $\Delta(r, \textsc{super}(R) \backslash R)$ can be computed from $Bag(r, \textsc{super}(R))$ and $Bag(r, R)$ without having to iterate over the automaton transitions. However, in the worst case, at each step in the algorithm, the Bag data structure can have exponential

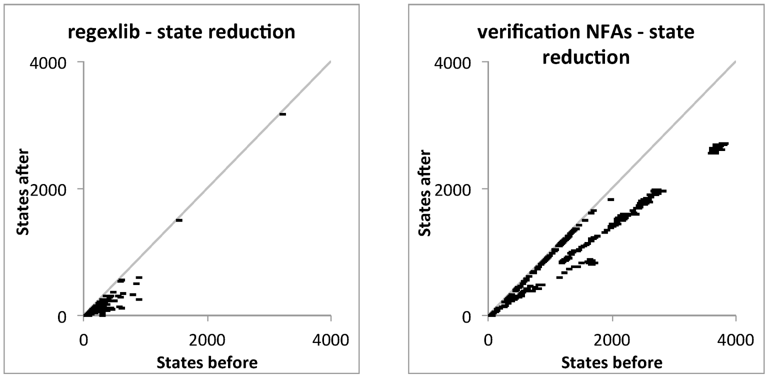


Fig. 3. State reduction for the two benchmark sets.

size in p, the number of distinct predicates in the s-FA. Using a similar amortized complexity argument to that used by Hopcroft's algorithm for minimizing DFAs [20], we have that, if we ignore the cost of computing the *bag* data structure, the algorithm has complexity $\mathcal{O}(m \log n)$. In summary, if the complexity of checking satisfiability of predicates of size ℓ is $f(\ell)$, the counting implementation of *GreedyBisimSFA*(M) presented in Fig. 1 has complexity $\mathcal{O}(2^p m \log n f(n\ell))$, where m is the number of transitions in the input s-FA and n is the number of states, and p is the number of distinct predicates in the automaton. Concretely, while this implementation helps reducing the number of iterations over the automaton transitions, it suffers from an extra cost that is a function of the alphabet complexity and of the predicates appearing in the automaton. Notice, that in the case of finite alphabets 2^p is exactly the size of the alphabet and this problem does not exist [2]. This is another remarkable case of how adapting classic algorithms to the symbolic setting is not always possible.

7 Evaluation

We evaluate our algorithms on two sets of benchmarks. We report the state reduction obtained using forward bisimulations and, for each algorithm, we compare the running times and the number of explored blocks. We use Simple to denote the algorithm presented at the top of Fig. 1, Greedy to denote the algorithm presented in Sec. 5, and Count to denote the counting based algorithm described in Sec. 6. As a sanity check, we assured that all the algorithms computed the same results. All the experiments were run on a 4-core Intel i7-2600 CPU 3.40GHz, with 8GB of RAM.

Regexlib. We collected the s-NFAs over the alphabet $\mathbf{2}^{\mathrm{BV16}}$ resulting from converting 2,625 regular expressions appearing in http://regexlib.com/. This website contains a library of crowd-sourced regular expressions for tasks such as detecting URLs, emails, and phone numbers. These s-NFAs have 1 to 3,174 states, 1 to 10,670 transitions, and have an average of 2 transitions per state.

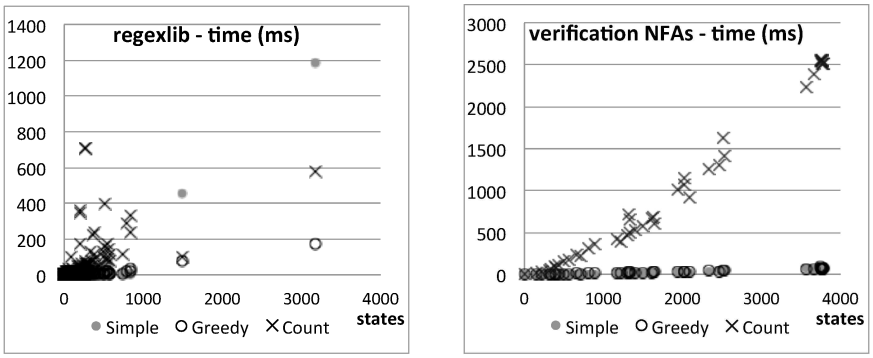


Fig. 4. Running times of three algorithms on regular expression from regexlib.com and on NFAs from verification applications. In the second plot, we do not show data points that are very close to each other to make the figure readable.

These benchmarks operate over very large alphabets and can only be handled symbolically. We use the algebra $\mathbf{2}^{\mathrm{BV}k}$.

Verification s-NFAs. We collected 1,000 s-NFAs over small alphabets (2-40 symbols) appearing in verification applications from [8]. These s-NFAs are generated from the steps of abstract regular model checking while verifying the bakery algorithm, a producer-consumer system, bubble sort, an algorithm that reverses a circular list, and a Petri net model of the readers/writers protocol. These s-FAs have 4 to 3,782 states, 7 to 18,670 transitions, and have an average of 4.1 transitions per state. Given the small size of the alphabets, these automata are quite dense. We represent the alphabet using the algebra **int**[k].

State reduction. Figure 3 shows the state reduction obtained by our algorithm. Each point (x, y) in the figure shows that an automaton with x states was reduced to an equivalent automaton with y states. On average, the number of states reduces by 14% and 19% for the regexlib benchmarks and the verification NFAs respectively.

Runtime. Figure 4 shows the running times of the algorithms on each benchmark s-FA. For the regexlib s-FAs, most automata take less than 1ms to complete causing the same running time for the three algorithms on 2528 benchmarks. In general, the Greedy algorithm is slightly faster than the other two algorithms and the Count algorithm is at times slower than both the other two algorithms (93 cases total), on relatively small cases. On two large instances (1,502 and 3,174 states, 1,502 and 10,670 transitions) the Greedy and Count algorithms clearly outperform the Simple algorithm.

For s-FAs from [8], the algorithms Simple and Greedy, have very comparable performances (Greedy is, on average, 6 ms slower than Simple). The Count algorithm is slower than both these algorithms in 90% of the cases and has the same performance in the remaining 10% of the cases.

In both experiments, almost all the computation time of the Count algorithm is spent manipulating the counting data structure presented in Sec. 6. In sum-

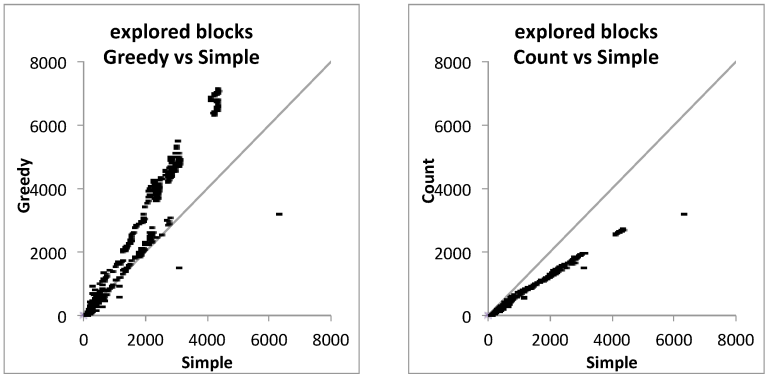


Fig. 5. Ratio of number of explored blocks between the Simple algorithm and the other algorithms.

mary, the Count algorithm, despite having $m \log n$ complexity, is consistently slower than the other two algorithms and the slowdown is due to the complexity of manipulating the counting data structure.

Explored blocks. We measure the number of blocks pushed into the worklist W for the different algorithms. Figure 5 shows the ratio between the explored blocks of the Simple algorithm and the other two algorithms. As expected from the theoretical complexities, the Count algorithm consistently explores fewer blocks than the Simple algorithm. As we observed in Figure 4, this is not enough to achieve better speedups. The Greedy algorithm often explores more blocks than the other two algorithms. This is because $R' = \textsc{super}(R) \backslash R$ of a set R is explored even in the cases where R' has already been split into subsets. In this case, the simple algorithm will only explore the splits and not the original set, while the Greedy algorithm will explore both R' as well as its splits.

8 Related work

Minimization of deterministic automata. Automata minimization algorithms have been studied and analyzed extensively in several different aspects. Moore's and Hopcroft's algorithms [25, 20] are the two most common algorithms for minimizing DFAs. Both of these algorithms compute forward bisimulations over DFAs and can be implemented with complexity $O(kn \log n)$ (where k is the size of the alphabet). This bound is tight [7, 6, 5]. The two algorithms, although in different ways, iteratively refine a partition of the set of states until the forward bisimulation is computed. In the case of DFAs, the equivalence relation induced by the bisimulation relation produces a minimal and canonical DFA. In our earlier work, we extended Hopcroft's algorithm to work with symbolic alphabets [13] and showed how, for deterministic s-FAs, the algorithm can be implemented in $\mathcal{O}(m \log n f(nl))$ for automata with m transitions, n states, and predicates of size l. Here $f(x)$ is the cost of checking satisfiability of predicates of size x. The algorithm proposed in [13] is similar to the greedy algorithm in Figure 1. The main difference is in the necessity to use $\textsc{super}(R)\setminus R$ in the **SAT** checks and that this seemingly small change has drastic complexity implications.

Minimization and state reduction in nondeterministic automata. In the case of NFAs, there exists no canonical minimal automaton and the problem of finding a minimal NFA is known to be PSPACE complete [24]. It is shown in [18] that it is not even possible to efficiently approximate NFA minimization. The original search based algorithm for minimizing NFAs is known as the Kameda-Weiner method [22]. A generalization of the Kameda-Weiner method based on atoms of regular languages [10] was recently introduced in [28]. Most practical approaches for computing small nondeterministic automata use notions of state reductions that do not always produce a minimal NFAs [2]. These techniques are based on computing various kinds of simulation and bisimulation relations. The set of most common such relations has been described in detail and extended to Büchi automata in [23]. In this paper, we are only concerned with performing state reduction by computing forward bisimulations.

Abdulla et al. were the first to observe that forward bisimulation for NFAs could be computed with complexity $\mathcal{O}(km \log n)$ by keeping track of the number of states each symbol can reach from a certain part of a partition [2]. In their paper, they also proposed an efficient implementation based on BDDs and algebraic decision diagrams for the special case in which the alphabet is a set of bit-vectors. The techniques proposed in [2] are tailored for finite alphabets and the goal of our paper is extending them to arbitrary alphabets that form a decidable Boolean algebra. In this paper, we propose an extension based on our symbolic bag data structure and experimentally show that, unlike for the case of finite alphabets, the counting algorithm is not practical.

Recently, Geldenhuys et al. have proposed a technique for reducing the size of certain classes of NFAs using SAT solvers [17]. In this technique, a SAT formula is used to describe the existence of an NFA that is equivalent to the original one, but has at most k states. Applying these techniques to symbolic automata is an interesting research direction.

Automata with predicates. The concept of automata with predicates instead of concrete symbols was first mentioned in [31] and was first discussed in [29] in the context of natural language processing. Since then s-FAs have been studied extensively and we have seen algorithms for minimizing deterministic s-FAs [13] and deterministic s-FAs over trees [14], and extensions of classic logic results to s-FAs [15]. To the best of our knowledge, the problem of reducing the states and efficiently computing forward bisimulations for nondeterministic s-FAs has not been studied before. The term symbolic automata is sometimes used to refer to automata over finite alphabets where the state space is represented using BDDs [27]. This meaning is different from the one described in this paper.

AutomataDotNet. This is an open source Microsoft Automata project [1] that is an extension of the automata toolkit originally introduced in [30]. The source code (written in C#) of all the algorithms discussed in this paper as well as the source code of the experiments discussed in Section 7 are available in [1].

Acknowledgements. Loris D'Antoni performed part of this work while visiting Microsoft Research, Redmond. We thank Zachary Kincaid for his feedback.

References

1. *AutomataDotNet*. `https://github.com/AutomataDotNet/`.
2. P. Abdulla, J. Deneux, L. Kaati, and M. Nilsson. Minimization of non-deterministic automata with large alphabets. In *CIAA 2005*, volume 3845 of *LNCS*, pages 31–42. Springer, 2006.
3. R. Alur, L. D'Antoni, and M. Raghothaman. Drex: A declarative language for efficiently evaluating regular string transformations. *SIGPLAN Not.*, 50(1):125–137, Jan. 2015.
4. R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in Systems Design*, 10(2/3):171–206, 1997.
5. J. Berstel, L. Boasson, and O. Carton. Hopcroft's automaton minimization algorithm and Sturmian words. In *DMTCS'2008*, pages 355–366, 2008.
6. J. Berstel and O. Carton. On the complexity of Hopcroft's state minimization algorithm. In *CIAA'2004*, volume 3317, pages 35–44, 2004.
7. N. Blum. An $0(n \log n)$ implementation of the standard method for minimizing n-state finite automata. *Information Processing Letters*, 57:65–69, 1996.
8. A. Bouajjani, P. Habermehl, and T. Vojnar. *Abstract Regular Model Checking*, pages 372–386. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
9. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.
10. J. Brzozowski and H. Tamm. Theory of átomata. *Theoretical Computer Science*, 539:13–27, 2014.
11. L. D'Antoni and M. Veanes. Equivalence of extended symbolic finite transducers. In N. Sharygina and H. Veith, editors, *CAV 2013*, volume 8044 of *LNCS*, pages 624–639. Springer, 2013.
12. L. D'Antoni and M. Veanes. Static analysis of string encoders and decoders. In R. Giacobazzi, J. Berdine, and I. Mastroeni, editors, *VMCAI 2013*, volume 7737 of *LNCS*, pages 209–228. Springer, 2013.
13. L. D'Antoni and M. Veanes. Minimization of symbolic automata. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, (POPL '14)*, pages 541–553. ACM, 2014.
14. L. D'Antoni and M. Veanes. Minimization of symbolic tree automata. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM, 2016.
15. L. D'Antoni and M. Veanes. Monadic second-order logic on finite sequences. In *Proceedings of the 44th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, (POPL '17)*. ACM, 2017.
16. L. D'Antoni, M. Veanes, B. Livshits, and D. Molnar. Fast: A transducer-based language for tree manipulation. *ACM Trans. Program. Lang. Syst.*, 38(1):1–32, 2015.
17. J. Geldenhuys, B. van der Merwe, and L. van Zijl. *Reducing Nondeterministic Finite Automata with SAT Solvers*, pages 81–92. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

18. G. Gramlich and G. Schnitger. *Minimizing NFA's and Regular Expressions*, pages 399–411. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
19. J. Hopcroft. An $n\log n$ algorithm for minimizing states in a finite automaton. In Z. Kohavi, editor, *Theory of machines and computations, Proc. Internat. Sympos., Technion, Haifa, 1971*, pages 189–196, New York, 1971. Academic Press.
20. J. E. Hopcroft and J. D. Ullman. *Formal languages and their relation to automata*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1969.
21. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.
22. T. Kameda and P. Weiner. On the state minimization of nondeterministic finite automata. *IEEE Transactions on Computers*, C-19(7):617–627, 1970.
23. R. Mayr and L. Clemente. Advanced automata minimization. In *POPL'13*, pages 63–74, 2013.
24. A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th Annual Symposium on Switching and Automata Theory (SWAT'72)*, pages 125–129. IEEE, 1972.
25. E. F. Moore. Gedanken-experiments on sequential machines. *Automata studies, Annals of mathematics studies*, (34):129–153, 1956.
26. R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.
27. K. Y. Rozier and M. Y. Vardi. A multi-encoding approach for LTL symbolic satisfiability checking. In *FM 2011: Formal Methods - 17th International Symposium on Formal Methods, Limerick, Ireland, June 20-24, 2011. Proceedings*, pages 417–431, 2011.
28. H. Tamm. New interpretation and generalization of the Kameda-Weiner method. In I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 116:1–116:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.
29. G. van Noord and D. Gerdemann. Finite state transducers with predicates and identities. *Grammars*, 4(3):263–286, 2001.
30. M. Veanes and N. Bjørner. Symbolic automata: The toolkit. In *TACAS*, volume 7214 of *LNCS*, pages 472–477. Springer, 2012.
31. B. W. Watson. Implementing and using finite automata toolkits. In *Extended finite state models of language*, pages 19–36, New York, NY, USA, 1999. Cambridge University Press.