Simulation Algorithms for Symbolic Automata

Lukáš Holík[1], Ondřej Lengál[1(✉)], Juraj Síč[1,2], Margus Veanes[3], and Tomáš Vojnar[1]

[1] FIT, Brno University of Technology, IT4Innovations Centre of Excellence, Brno, Czech Republic
lengal@fit.vutbr.cz
[2] Faculty of Informatics, Masaryk University, Brno, Czech Republic
[3] Microsoft Research, Redmond, USA

Abstract. We investigate means of efficient computation of the simulation relation over symbolic finite automata (SFAs), i.e., finite automata with transitions labeled by predicates over alphabet symbols. In one approach, we build on the algorithm by Ilie, Navaro, and Yu proposed originally for classical finite automata, modifying it using the so-called mintermisation of the transition predicates. This solution, however, generates all Boolean combinations of the predicates, which easily causes an exponential blowup in the number of transitions. Therefore, we propose two more advanced solutions. The first one still applies mintermisation but in a local way, mitigating the size of the exponential blowup. The other one focuses on a novel symbolic way of dealing with transitions, for which we need to sacrifice the counting technique of the original algorithm (counting is used to decrease the dependency of the running time on the number of transitions from quadratic to linear). We perform a thorough experimental evaluation of all the algorithms, together with several further alternatives, showing that all of them have their merits in practice, but with the clear indication that in most of the cases, efficient treatment of symbolic transitions is more beneficial than counting.

1 Introduction

We investigate algorithms for computing simulation relations on states of symbolic finite automata. *Symbolic finite automata* (SFAs) [1,2] extend the classical (nondeterministic) finite automata (NFAs) by allowing one to annotate a transition with a predicate over a possibly infinite alphabet. Such *symbolic* transitions then represent a set of all (possibly infinitely many) concrete transitions over all the individual symbols that satisfy the predicate. SFAs offer a practical solution for automata-based techniques whenever the alphabet is prohibitively large to be processed with a standard NFA, for instance, when processing Unicode-encoded text (e.g., within various security-related analyses) or in automata-based decision procedures for logics such as MSO or WS1S [3,4]. Applications of SFAs

S. K. Lahiri and C. Wang (Eds.): ATVA 2018, LNCS 11138, pp. 109–125, 2018.
https://doi.org/10.1007/978-3-030-01090-4_7

over arithmetic alphabets and formulas arise also when dealing with symbolic transducers in the context of various sanitizer and encoder analyses [4].

A *simulation relation* on an automaton underapproximates the inclusion of languages of individual states [20]. This makes it useful for reducing nondeterministic automata and in testing inclusion and equivalence of their languages [5,6,20]. Using simulation for these purposes is often the best compromise between two other alternatives: (i) the cheap but strict bisimulation and (ii) the liberal but expensive language inclusion.

The obvious solution to the problem of computing simulation over an SFA is to use the technique of *mintermisation*: the input SFA is transformed into a form in which predicates on transitions partition the alphabet. Predicates on transitions can then be treated as ordinary alphabet symbols and most of the existing algorithms for NFAs can be used out of the box, including a number of algorithms for computing simulations. We, in particular, consider mintermisation mainly together with the algorithm by Ilie, Navaro, and Yu from [7] (called INY in the following), and, in the experiments, also with the algorithm by Ranzato and Tapparo (called also RT) [19]. A fundamental problem is that mintermisation can increase the number of transitions exponentially due to generating all Boolean combinations of the original transition predicates. Moreover, this problem is not only theoretical, but causes a significant blowup in practice too, as witnessed in the experiments presented in this paper.

We therefore design algorithms that do not need mintermisation. We take as our starting point the algorithm INY, which has the best available time complexity $\mathcal{O}(nm)$ in terms of the number of states n and transitions m of the input NFA. We propose two generalisations of this algorithm. The first one (called LocalMin) reflects closely the ideas that INY uses to achieve the low complexity. Instead of applying INY on a globally mintermised SFA, it, however, requires only a *locally mintermised form*: for every state, the predicates on its outgoing transitions partition the alphabet. Local mintermisation is thus exponential only to the maximal out-degree of a state.

Our second algorithm (called NoCount) is fundamentally different from LocalMin because it trades off the upfront mintermisation cost against working with predicates in the algorithm, and therefore has a different worst case computational complexity wrt the number of transitions. We show experimentally that this trade-off pays off. To facilitate this trade-off, we had to drop a counting technique that INY uses to improve its time complexity from $\mathcal{O}(n^2m)$ to $\mathcal{O}(nm)$ and that replaces repeated tests for existence of transitions with certain properties by maintaining their number in a dedicated counter and testing it for zero. Dropping the counter-based approach (which depends on at least local mintermisation) in turn allowed an additional optimisation based on aggregating a batch of certain expensive operations (satisfiability checking) on symbolic transitions into one. Overall, this improves the efficiency and ultimately reduces a worst-case 2^m cost, which is typically *independent* of the Boolean algebra, to the cost of inlining the Boolean algebra operations, which may be polynomial or even (sub)linear in m.

In our experiments, although each of the considered algorithms wins in some cases, our new algorithms performed overall significantly better than INY with global mintermisation. NoCount performed the best overall, which suggests that avoiding mintermisation and aggregating satisfiability tests over transition labels is practically more advantageous than using the counting technique of INY. We have also compared our algorithms with a variant [8] of the RT algorithm, one of the fastest algorithms for computing simulation, run on the globally mintermised automata (we denote the combination as GlobRT). The main improvement of RT over INY is its use of partition-relation pairs, which allows one to aggregate operations with blocks of the so-far simulation indistinguishable states. Despite this powerful optimisation and the fine-tuned implementation of RT in the Vata library [9], NoCount has a better performance than GlobRT on automata with high diversity of transition predicates (where mintermisation significantly increases the number of transitions).

Related work. Simulation algorithms for NFAs might be divided between *simple* and *partition-based.* Among the simple algorithms, the algorithm by Henzinger, Henzinger, and Köpke [10] (called HHK) is the first algorithm that achieved the time complexity $\mathcal{O}(nm)$ on Kripke structures. The later algorithm INY [7] is a small modification of HHK and works on finite automata in a time at worst $\mathcal{O}(nm)$. The automata are supposed to be complete (every state has an outgoing transition for every alphabet symbol). INY can be adapted for non-complete automata by adding an initialisation step which costs $\mathcal{O}(\ell n^2)$ time where ℓ is the size of the alphabet, resulting in $\mathcal{O}(nm + \ell n^2)$ overall complexity (cf. Sect. 3.1).

The first partition-based algorithm was RT, proposed in [19]. The main innovation of RT is that the overapproximation of the simulation relation is represented by a so-called *partition-relation pair*. In a partition-relation pair, each class of the partition of the set of states represents states that are simulation-equivalent in the current approximation of the simulation, and the relation on the partition denotes the simulation-bigger/smaller classes. Working with states grouped into blocks is faster than working with individual states, and in the case of the most recent partition-based algorithms for Kripke structures [11], it allows to derive the time complexity $\mathcal{O}(n'm)$ where n' is the number of classes of the simulation equivalence (the partition-based algorithms are also significantly faster in practice, although their complexity in terms of m and n is still $\mathcal{O}(nm)$). See e.g. [11] for a more complete overview of algorithms for computing simulation over NFAs and Kripke structures.

Our choice of INY over HHK among the simple algorithms is justified by a smaller dependence of the data structures of INY on the alphabet size. The main reason for basing our algorithms on one of the simple algorithms is their relative simplicity. Partition-based algorithms are intricate as well as the proofs of their small asymptotic complexity. Moreover, they compute predecessors of dynamically refined blocks of states via individual alphabet symbols, which seems to be a problematic step to efficiently generalise for symbolic SFA transitions. Having said that, it remains true that the technique of representing preorders through partition-relation pairs is from the high-level perspective

orthogonal to the techniques we have developed to generalise INY. Combining both types of optimisations would be a logical continuation of this work. It is, however, questionable if generalising already very complex partition-based algorithms, such as [11,19], is the best way to approach computing simulations over SFAs. Most of the intricacy of the partition-based algorithms aims at combining the counting technique with the partition-relation pairs. Our experimental results suggest, however, that rather than using the counting technique, it is more important to optimise the treatment of symbolic transitions and to avoid mintermisation.

Our work complements other works on generalising classical automata algorithms to SFAs, mainly the deterministic minimisation [12] and computing of bisimulation [13].

2 Preliminaries

Throughout the paper, we use the following notation: If $R \subseteq A_1 \times \cdots \times A_n$ is an n-ary relation for $n \geq 2$, then $R(x_1, \ldots, x_{n-1}) \stackrel{\text{def}}{=} \{y \in A_n \mid R(x_1, \ldots, x_{n-1}, y)\}$ for any $x_1 \in A_1, \ldots, x_{n-1} \in A_{n-1}$. Let $R^{\complement} \stackrel{\text{def}}{=} (A_1 \times \ldots \times A_n) \setminus R$.

Effective Boolean algebra. An *effective Boolean algebra* is defined as a tuple $\mathcal{A} = (\mathfrak{D}, \mathbb{P}, [\![\cdot]\!], \vee, \wedge, \neg)$ where $\mathbb{P}$ is a set of *predicates* closed under predicate transformers $\vee, \wedge : \mathbb{P} \times \mathbb{P} \to \mathbb{P}$ and $\neg : \mathbb{P} \to \mathbb{P}$. A first order interpretation (denotation) $[\![\cdot]\!] : \mathbb{P} \to 2^{\mathfrak{D}}$ assigns to every predicate of $\mathbb{P}$ a subset of the *domain* $\mathfrak{D}$ such that, for all $\varphi, \psi \in \mathbb{P}$, it holds that $[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$, $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$, and $[\![\neg\varphi]\!] = \mathfrak{D} \setminus [\![\varphi]\!]$. For $\varphi \in \mathbb{P}$, we write $\mathit{IsSat}(\varphi)$ when $[\![\varphi]\!] \neq \emptyset$ and say that φ is *satisfiable*. The predicate *IsSat* and the predicate transformers $\wedge$, $\vee$, and $\neg$ must be effective (computable). We assume that $\mathbb{P}$ contains predicates $\top$ and $\bot$ with $[\![\top]\!] = \mathfrak{D}$ and $[\![\bot]\!] = \emptyset$. Let Φ be a subset of $\mathbb{P}$. If the denotations of any two distinct predicates in Φ are disjoint, then Φ is called a *partition* (of the set $\bigcup_{\varphi \in \Phi} [\![\varphi]\!]$). The set $\mathit{Minterms}(\Phi)$ of *minterms* of a finite set Φ of predicates is defined as the set of all satisfiable predicates of $\{\bigwedge_{\varphi \in \Phi'} \varphi \wedge \bigwedge_{\varphi \in \Phi \setminus \Phi'} \neg\varphi \mid \Phi' \subseteq \Phi\}$. Notice that every predicate of Φ is equivalent to a disjunction of minterms in $\mathit{Minterms}(\Phi)$.

Below, we assume that it is possible to measure the size of the predicates of the effective Boolean algebra $\mathcal{A}$ that we work with. We denote by $\mathcal{C}_{sat}(x, y)$ the worst-case complexity of constructing a predicate obtained by applying x operations of $\mathcal{A}$ on predicates of the size at most y and checking its satisfiability.

Symbolic finite automata. We define a *symbolic finite automaton* (SFA) as a tuple $M = (Q, \mathcal{A}, \Delta, I, F)$ where Q is a finite set of *states*, $\mathcal{A} = (\mathfrak{D}, \mathbb{P}, [\![\cdot]\!], \vee, \wedge, \neg)$ is an effective Boolean algebra, $\Delta \subseteq Q \times \mathbb{P} \times Q$ is a finite *transition relation*, $I \subseteq Q$ is a set of *initial states*, and $F \subseteq Q$ is a set of *final states*. An element (q, ψ, p) of Δ is called a *(symbolic) transition* and denoted by $q \text{-}\{\psi\}\!\!\rightarrow p$. We write $[\![q \text{-}\{\psi\}\!\!\rightarrow p]\!]$ to denote the set $\{q \text{-}\{a\}\!\!\rightarrow p \mid a \in [\![\psi]\!]\}$ of *concrete transitions* represented by $q \text{-}\{\psi\}\!\!\rightarrow p$, and we let $[\![\Delta]\!] \stackrel{\text{def}}{=} \bigcup_{q \text{-}\{\psi\}\!\!\rightarrow p \in \Delta} [\![q \text{-}\{\psi\}\!\!\rightarrow p]\!]$. For $q \in Q$ and $a \in \mathfrak{D}$ let $\Delta(q, a) \stackrel{\text{def}}{=} \{p \in Q \mid q \text{-}\{a\}\!\!\rightarrow p\}$.

In the following, it is assumed that predicates of all transitions of an SFA are satisfiable unless stated otherwise. A sequence $\rho = q_0 a_1 q_1 a_2 \cdots a_n q_n$ with $q_{i-1} \text{-}\{a_i\}\text{-} q_i \in [\![\Delta]\!]$ for every $1 \leq i \leq n$ is a *run* of M over the word $a_1 \cdots a_n$. The run ρ is *accepting* if $q_0 \in I$ and $q_n \in F$, and a word is *accepted* by M if it has an accepting run. The *language* $\mathcal{L}(M)$ of M is the set of all words accepted by M.

An SFA M is *complete* iff, for all $q \in Q$ and $a \in \mathfrak{D}$, there is $p \in Q$ with $p \text{-}\{a\}\text{-} q \in [\![\Delta]\!]$. An SFA can be completed in a straightforward way: from every state q, we add a transition from q labelled with $\neg \bigvee \{\varphi \mid \exists p \in Q : q \text{-}\{\varphi\}\text{-} p \in \Delta\}$ to a new non-accepting sink state, if the disjunction is satisfiable.

An SFA M is *globally mintermised* if the set $\mathbb{P}_\Delta \stackrel{\text{def}}{=} \{\varphi \in \mathbb{P} \mid \exists p, q : p \text{-}\{\varphi\}\text{-} q \in \Delta\}$ of the predicates appearing on its transitions is a partition. Every SFA can be made globally mintermised by replacing each $p \text{-}\{\varphi\}\text{-} q \in \Delta$ by the set of transitions $\{p \text{-}\{\omega\}\text{-} q \mid \omega \in \mathit{Minterms}(\mathbb{P}_\Delta) \wedge \mathit{IsSat}(\omega \wedge \varphi)\}$ (see e.g. [12] for an efficient algorithm), where $\mathit{IsSat}(\omega \wedge \varphi)$ is an implementation of the test $[\![\omega]\!] \subseteq [\![\varphi]\!]$, because if ω is a minterm of $\mathbb{P}_\Delta$ and $\varphi \in \mathbb{P}_\Delta$ then $[\![\omega]\!] \cap [\![\varphi]\!] \neq \emptyset$ implies that $[\![\omega]\!] \subseteq [\![\varphi]\!]$. Since for a set of predicates Φ, the size of $\mathit{Minterms}(\Phi)$ is at worst $2^{|\Phi|}$, global mintermisation is exponential in the number of transitions.

A classical *(nondeterministic) finite automaton* (NFA) $N = (Q, \Sigma, \Delta, I, F)$ over a finite alphabet Σ can be seen as a special case of an SFA where Δ contains solely transitions of the form $q \text{-}\{a\}\text{-} r$ s.t. $a \in \Sigma$ and $[\![a]\!] = \{a\}$ for all $a \in \Sigma$. Below, we will sometimes interpret an SFA $M = (Q, \mathcal{A}, \Delta, I, F)$ as its *syntactic NFA* $N = (Q, \mathbb{P}_\Delta, \Delta, I, F)$ in which the predicates are treated as syntactic objects.

Simulation. Let $M = (Q, \mathcal{A}, \Delta, I, F)$ be an SFA. A relation S on Q is a *simulation* on M if whenever $(p, r) \in S$, then the following two conditions hold: (C1) if $p \in F$, then $r \in F$, and (C2) for all $a \in \mathfrak{D}$ and $p' \in Q$ such that $p \text{-}\{a\}\text{-} p' \in [\![\Delta]\!]$, there is $r' \in Q$ such that $r \text{-}\{a\}\text{-} r' \in [\![\Delta]\!]$ and $(p', r') \in S$. There exists a unique maximal simulation on M, which is reflexive and transitive. We call it the *simulation (preorder)* on M and denote it by $\preceq_M$ (or $\preceq$ when M is clear from the context). Computing $\preceq$ on a given SFA is the subject of this paper. A simulation that is symmetric is called a *bisimulation*, and *the bisimulation equivalence* is the (unique) largest bisimulation, which is always an equivalence relation.

3 Computing Simulation over SFAs

In this section, we present our new algorithms for computing the simulation preorder over SFAs. We start by recalling an algorithm for computing the simulation preorder on an NFA of Ilie, Navarro, and Yu from [7] (called INY), which serves as the basis for our work. Then, we introduce three modifications of INY for SFAs: (i) GlobINY, (ii) LocalMin, and (iii) NoCount. GlobINY is merely an application of the mintermisation technique: first globally mintermise the SFA and then use INY to compute the NFA simulation preorder over the result. The main contribution of our paper lies in the other two algorithms, which are sub-

tler modifications of INY that avoid global mintermisation by reasoning about the semantics of transition predicates of SFAs.

Before turning to the different algorithms, we start by explaining how $\preceq_M$ can be computed by an abstract fixpoint procedure and provide the intuition behind how such a procedure can be lifted to the symbolic setting.

Abstract procedure for computing $\preceq_M$. We start by presenting an *abstract fixpoint procedure* for computing the simulation $\preceq_M$ on an SFA $M = (Q, \mathcal{A}, \Delta, I, F)$. We formulate it using the notion of *minimal nonsimulation* $\not\preceq_M$ (which is a dual concept to the maximal simulation $\preceq_M$ introduced before), defined as the least subset $\not\preceq \subseteq Q \times Q$ s.t. for all $s, t \in Q$, it holds that

$$\begin{aligned} s \not\preceq t \Leftrightarrow (s \in F \wedge t \notin F) \vee \\ \exists i \in Q. \underbrace{\exists a \in \mathfrak{D}. (s \dashv\{a\}\mapsto i \wedge \forall j \in Q. (t \dashv\{a\}\mapsto j \Rightarrow i \not\preceq j))}_{(1^*)}. \end{aligned} \tag{1}$$

Informally, s cannot be simulated by t iff (line 1) s is accepting and t is not, or (line 2) s can continue over some symbol a into i, while t cannot simulate this move by any of its successors j. It is easy to see that $\preceq_M = \not\preceq_M^{\complement}$. The algorithms for computing simulation over NFAs are efficient implementations of such a fixpoint procedure using *counter-based* implementations for evaluating (1*). Namely, for every symbol a and a pair of states t and i, it keeps count of those states j that could possibly contradict the universally quantified property. The count dropping to zero means that the property holds universally.

Symbolic abstract procedure for computing $\preceq_M$. When the domain $\mathfrak{D}$ is very large or infinite, then evaluating (1*) directly is infeasible. If $\mathit{Minterms}(\mathbb{P}_\Delta)$ is exponentially larger than the set $\mathbb{P}_\Delta$, then evaluating (1*) with a ranging over $\mathit{Minterms}(\mathbb{P}_\Delta)$ may also be infeasible. Instead, we want to utilize the operations of the algebra $\mathcal{A}$ without explicit reference to elements in $\mathfrak{D}$ and without constructing $\mathit{Minterms}(\mathbb{P}_\Delta)$. The key insight is that condition (1*) is equivalent to

$$\mathit{IsSat}(\varphi_{si} \wedge \neg \Gamma(t, \not\preceq^{\complement}(i))) \tag{2}$$

where, for $t, s, i \in Q$ and $J \subseteq Q$, we define $\varphi_{si} \stackrel{\text{def}}{=} \bigvee_{(s,\psi,i)\in\Delta} \psi$ and $\Gamma(t, J) \stackrel{\text{def}}{=} \bigvee_{j\in J} \varphi_{tj}$, i.e., $\Gamma(t, \not\preceq^{\complement}(i))$ is a disjunction of predicates on all transitions leaving t and entering a state that simulates i. Using (2) to compute (1*) in the abstract procedure thus eliminates the explicit quantification over $\mathfrak{D}$ and avoids computation of $\mathit{Minterms}(\mathbb{P}_\Delta)$. The equivalence between (1*) and (2) holds because, for all $a \in \mathfrak{D}$ and $R \subseteq Q \times Q$, we have

$$a \in [\![\neg\Gamma(t, R^{\complement}(i))]\!] \;\Leftrightarrow\; \neg\exists j(t \dashv\{a\}\mapsto j \wedge (i, j) \in R^{\complement}) \;\Leftrightarrow\; \forall j(t \dashv\{a\}\mapsto j \Rightarrow (i, j) \in R).$$

The fixpoint computation based on (2) is used in our algorithm NoCount, which does not require mintermisation. Its disadvantage is that it is not compatible with the counting technique. Our algorithm LocalMin is then a compromise between mintermisation and NoCount that retains the counting technique for the price of using a cheaper, local variant of mintermisation.

Algorithm 1: INY

Input: An NFA $N = (Q, \Sigma, \Delta, I, F)$
Output: The simulation preorder $\preceq_N$

```
1  for p, q ∈ Q, a ∈ Σ do  N_a(q, p) := |Δ(q, a)|
2  Sim := Q × Q;
3  NotSim := F × (Q \ F) ∪ {(q, r) | ∃a ∈ Σ : Δ(q, a) ≠ ∅ ∧ Δ(r, a) = ∅};
4  while NotSim ≠ ∅ do
5  |   remove some (i, j) from NotSim and Sim;
6  |   for t -{a}-> j ∈ Δ do
7  |   |   N_a(t, i) := N_a(t, i) − 1;
8  |   |   if N_a(t, i) = 0 then                                  // t^a↑i = ∅
9  |   |   |   for s -{a}-> i ∈ Δ s.t. (s, t) ∈ Sim do
10 |   |   |   |   NotSim := NotSim ∪ {(s, t)};
11 return Sim;
```

3.1 Computing Simulation over NFAs (INY)

In Algorithm 1, we give a slightly modified version of the algorithm INY from [7] for computing the simulation preorder over an NFA $N = (Q, \Sigma, \Delta, I, F)$.

The algorithm refines an overapproximation *Sim* of the simulation preorder until it satisfies the definition of a simulation. The set *NotSim* is used to store pairs of states (i, j) that were found to contradict the definition of the simulation preorder. *NotSim* is initialised to contain (a) pairs that contradict condition C1 and (b) pairs that cannot satisfy condition C2 regardless of the rest of the relation, as they relate states with incompatible outgoing symbols. All pairs (i, j) in *NotSim* are subsequently processed by removing (i, j) from *Sim* and propagating the change of *Sim* according to condition C2: for all transitions $t \text{-}\{a\}\!\!\rightarrow j \in \Delta$, it is checked whether j was the last a-successor of t that could be simulation-greater than i (hence there are no more such transitions after removing (i, j) from *Sim*). If this is the case, then t cannot simulate any a-predecessor s of i, and so all such pairs $(s, t) \in Sim$ are added to *NotSim*. In order to have the previous test efficient (a crucial step for the time complexity of the algorithm), the algorithm uses a three-dimensional array of counters $N_a(t, i)$, whose invariant at line 5 is $N_a(t, i) = |t^a\!\uparrow i|$ where $t^a\!\uparrow i$ is the set $\Delta(t, a) \cap Sim(i)$ of successors of t over a that simulate i in the current simulation approximation *Sim*. In order to test $t^a\!\uparrow i = \emptyset$—i.e. the second conjunct of (1*)—, it is enough to test if $N_a(t, i) = 0$.

The lemma below shows the time complexity of INY in terms of $n = |Q|$, $m = |\Delta|$, and $\ell = |\Sigma|$. The original paper [7] proves the complexity $O(nm)$ for complete automata, in which case $m \geq \ell n$, so the factor ℓn^2 is subsumed by nm. Since completion of NFAs can be expensive, the initialization step on line 3 of our algorithm is modified (similarly as in [14]) to start with considering states with different sets of symbols appearing on their outgoing transitions as simulation-different; the cost of this step is subsumed by the factor ℓn^2 (see [21] for the proof of our formulation of the algorithm).

Lemma 1. INY *computes* $\preceq_N$ *in time* $\mathcal{O}\big(nm + \ell n^2\big)$.

Algorithm 2: GlobINY

Input: An SFA $M = (Q, \mathcal{A}, \Delta, I, F)$
Output: The simulation preorder $\preceq_M$
1 $\Delta_G :=$ globally mintermised Δ;
2 **return** INY$((Q, \mathbb{P}_{\Delta_G}, \Delta_G, I, F))$;

3.2 Global Mintermisation-based Algorithm for SFAs (GlobINY)

The algorithm GlobINY (Algorithm 2) is the initial solution for the problem of computing the simulation preorder over SFAs. It first globally mintermises the input automaton $M = (Q, \mathcal{A}, \Delta, I, F)$, then interprets the result as an NFA over the alphabet of the minterms, and runs INY on the NFA. The following lemma (together with Lemma 1) implies the correctness of this approach.

Lemma 2. *Let $N = (Q, \mathbb{P}_\Delta, \Delta, I, F)$ be the syntactic NFA of a globally mintermised SFA $M = (Q, \mathcal{A}, \Delta, I, F)$. Then $\preceq_M = \preceq_N$.*

The lemma below shows the time complexity of GlobINY in terms of $n = |Q|$, $m = |\Delta|$, and the size k of the largest predicate used in Δ.

Lemma 3. GlobINY *computes $\preceq_M$ in time $\mathcal{O}\big(nm2^m + \mathcal{C}_{sat}(m,k)2^m\big)$.*

Intuitively, the complexity follows from the fact each transition of Δ can be replaced by at most 2^m transitions in Δ_G since there can be at most 2^m minterms in $\mathit{Minterms}(\mathbb{P}_\Delta)$. Nevertheless, 2^m minterms will always be generated (some of them unsatisfiable, though), each of them generated from m predicates of size at most k. More details are available in [21].

3.3 Local Mintermisation-based Algorithm for SFAs (LocalMin)

Our next algorithm, called LocalMin (Algorithm 3), represents an attempt of running INY on the original SFA without the global mintermisation used above. The main challenge in LocalMin is how to symbolically represent the counters $N_a(q,r)$—representing them explicitly would contradict the idea of symbolic automata and would be impossible if the domain $\mathfrak{D}$ were infinite. We will therefore use counters $N_\psi(q,r)$ indexed with labels ψ of outgoing transitions of q to represent all counters $N_a(q,r)$, with $a \in [\![\psi]\!]$. A difficulty here is that if the automaton is not globally mintermised, then for some $q \text{-}\{\varphi\}\!\mapsto p$ and $a, b \in [\![\varphi]\!]$, the sizes of $q^{a}\!\!\nearrow r$ and $q^{b}\!\!\nearrow r$ may differ and hence cannot be represented by a single counter. [1] For example, if the only outgoing transition of q other than $q \text{-}\{\varphi\}\!\mapsto p$ is $q \text{-}\{\psi\}\!\mapsto r$ with $(p,r) \in \mathit{Sim}$, $[\![\varphi]\!] = \{a, b\}$, and $[\![\psi]\!] = \{b\}$, then $|q^{a}\!\!\nearrow r| = 1$ while $|q^{b}\!\!\nearrow r| = 2$. To avoid this problem, we introduce the so-called local mintermised form, in which only labels on outgoing transitions of every state must form a partition.

[1] When describing an algorithm that works over an SFA, we use the notation $q^{a}\!\!\nearrow r$ to represent the set $[\![\Delta]\!](q,a) \cap \mathit{Sim}(r)$, i.e., it refers to the *concrete* transitions of $[\![\Delta]\!]$.

Algorithm 3: LocalMin

Input: A complete SFA $M = (Q, \mathcal{A}, \Delta, I, F)$
Output: The simulation preorder $\preceq_M$

```
1  Δ_L := locally mintermised form of Δ;
2  for p, q ∈ Q, q -{ψ}-> t ∈ Δ_L do
3  |   N_ψ(q, p) := |Δ_L(q, ψ)| ;
4  Sim := Q × Q; NotSim := F × (Q \ F)
5  while NotSim ≠ ∅ do
6  |   remove some (i, j) from NotSim and Sim;
7  |   for t -{ψ_tj}-> j ∈ Δ_L do
8  |   |   N_ψtj(t, i) := N_ψtj(t, i) − 1;
9  |   |   if N_ψtj(t, i) = 0 then                          // t^ψtj↛ i = ∅
10 |   |   |   for s -{φ_si}-> i ∈ Δ s.t. (s, t) ∈ Sim do
11 |   |   |   |   if IsSat(ψ_tj ∧ φ_si) then
12 |   |   |   |   |   NotSim := NotSim ∪ {(s, t)};
13 return Sim;
```

Formally, we say that an SFA $M = (Q, \mathcal{A}, \Delta, I, F)$ is *locally mintermised* if for every state $p \in Q$, the set $\mathbb{P}_{\Delta,p} \stackrel{\text{def}}{=} \{\varphi \in \mathbb{P} \mid \exists q : p \text{-}\{\varphi\}\!\!\rightarrow q \in \Delta\}$ of the predicates used on the transitions starting from p is a partition. A locally mintermised form is obtained by replacing every transition $p \text{-}\{\varphi\}\!\!\rightarrow q$ by the set of transitions $\{p \text{-}\{\omega\}\!\!\rightarrow q \mid \omega \in \mathit{Minterms}(\mathbb{P}_{\Delta,p}) \wedge \mathit{IsSat}(\omega \wedge \varphi)\}$. Local mintermisation can hence be considerably cheaper than global mintermisation as it is only exponential to the maximum out-degree of a state (instead of the number of transitions of the whole SFA). The key property of a locally mintermised SFA M_L is the following: for any transition $q \text{-}\{\varphi\}\!\!\rightarrow p$ of M_L and a state $r \in Q$, and for any value of Sim, it holds that $|q \not\rightarrow^{a} r|$ is the same for all $a \in [\![\varphi]\!]$. This means that the set of counters $\{N_a(q, r) \mid a \in [\![\varphi]\!]\}$ for all symbols in the semantics of φ can be represented by a single counter $N_\varphi(q, r)$.

The use of only locally mintermised transitions also necessitates a modification of the **for** loop on line 6 of INY. In particular, the test on line 9 of INY, which determines the states s that cannot simulate t over the symbol a, only checks syntactic equivalence of the symbols. This could lead to incorrect results because (syntactically) different local minterms of different source states t and s can still have overlapping semantics. It can, in particular, happen that if a counter $N_{\psi_{tj}}(t, i)$, for some predicate ψ_{tj}, reaches zero on line 9 of LocalMin, there is a transition from state s to i over a predicate φ_{si} different from ψ_{tj} but with some symbol $a \in [\![\varphi_{si}]\!] \cap [\![\psi_{tj}]\!]$. Because of a, the state t cannot simulate s, but this would not happen if the two predicates were only compared syntactically. LocalMin solves this issue on lines 10 and 11, where it iterates over all transitions entering i and leaving a state s simulated by t (wrt Sim), and tests whether the predicate φ_{si} on the transition semantically intersects with ψ_{tj}.

LocalMin is correct only if the input SFA is complete. As mentioned in Sect. 2, this is, however, not an issue, since completion of an SFA is, unlike for

NFAs, straightforward, and its cost is negligible compared with the complexity of LocalMin presented below.

The lemma below shows the time complexity of LocalMin in terms of $n = |Q|$, $m = |\Delta|$, the size k of the largest predicate used in Δ, the *out-degree* m_q for each $q \in Q$ (i.e. the number of transitions leaving q), and the overall *maximum out-degree* $W = \max\{m_q \mid q \in Q\}$.

Lemma 4. LocalMin *derives* $\preceq_M$ *in time*

$$\mathcal{O}\big(n \sum_{q \in Q} m_q 2^{m_q} + m\mathcal{C}_{sat}(W,k) \sum_{q \in Q} 2^{m_q}\big).$$

As shown in more detail in [21], the result can be proved in a similar way as in the case of INY and GlobINY, taking into account that each transition is, again, replaced by its mintermised versions. This time, however, the mintermised versions are computed independently and locally for each state (and the complexities are summed). Consequently, the factor 2^m gets replaced by 2^{m_q} for the different states $q \in Q$ (together with the replacement of $\mathcal{C}_{sat}(m,k)$ by $\mathcal{C}_{sat}(W,k)$), which can significantly decrease the complexity. On the other hand, as mintermisation is done separately for each state (which can sometimes lead to re-doing some work done only once in GlobINY) and as one needs the satisfiability test on line 11 of LocalMin instead of the purely syntactic test on line 9 of INY, on which GlobINY is based, GlobINY can sometimes win in practice. This fact shows up even in our experiments presented in Sect. 4.

3.4 Counter-Free Algorithm for SFAs (NoCount)

Before we state our last algorithm, named NoCount (Algorithm 4), let us recall that given an SFA $M = (Q, \mathcal{A}, \Delta, I, F)$, a set $S \subseteq Q$, and a state $q \in Q$, we use $\Gamma(q,S)$ to denote the disjunction of all predicates that reach S from q. We will also write $q \to S$ to denote that there is a transition from q to some state in S.

In NoCount, we sacrifice the counting technique in order to avoid the local mintermisation (which is still a relatively expensive operation). The obvious price for dropping the counters and local mintermisation is that the emptiness of $t\overset{a}{\nearrow}i$ for symbols $a \in \psi_{ti}$ can no more be tested in a constant time by asking whether $N_{\psi_{ti}}(t,i) = 0$ as on line 9 of LocalMin. It does not even hold any more that $t\overset{a}{\nearrow}i$ is uniformly empty or non-empty for all $a \in \psi_{ti}$. To resolve the issue, we replace the test from line 9 of LocalMin by computing the formula $\psi = \Gamma(t, Sim(i))$ on line 7 of NoCount, which is then used in the test on line 9. Intuitively, ψ represents all b's such that $t\overset{b}{\nearrow}i$ is *not* empty. By taking the negation of ψ, the test on line 9 of NoCount then explicitly asks whether there is some $a \in [\![\varphi_{si}]\!]$ for which s can go to i and t cannot simulate this move.

Further, notice that NoCount uses the set Rm for the following optimisation. Namely, if the use of Rm were replaced by an analogy of line 6 from LocalMin, it could choose a sequence of several $j \in Q$ such that $(i,j) \in NotSim$, and then the same ψ would be constructed for each j and tested against

Algorithm 4: NoCount

```
   Input: A complete SFA M = (Q, A, Δ, I, F)
   Output: The simulation preorder ⪯_M
1  Sim := Q × Q; NotSim := F × (Q \ F);
2  while ∃i ∈ Q : NotSim(i) ≠ ∅ do
3  |  Rm := {t | t → NotSim(i)};
4  |  Sim(i) := Sim(i) \ NotSim(i);
5  |  NotSim(i) := ∅;
6  |  for t ∈ Rm do
7  |  |  ψ := Γ(t, Sim(i));
8  |  |  for s -{φ_si}-> i ∈ Δ s.t. (s, t) ∈ Sim do
9  |  |  |  if IsSat(¬ψ ∧ φ_si) then
10 |  |  |  |  NotSim := NotSim ∪ {(s, t)};
11 return Sim;
```

the same φ_{si}. In contrast, due to its use of Rm, NoCount will process all $j \in NotSim(i)$ in a single iteration of the main **while** loop, in which ψ is computed and tested against φ_{si} only once.

Lemma 5 shows the complexity of NoCount in the terms used in Lemma 4.

Lemma 5. NoCount *computes* $\preceq_M$ *in time* $\mathcal{O}\big(n \sum_{q \in Q} m_q^2 + m^2 \mathcal{C}_{sat}(W, k)\big)$.

Observe that $\sum_{q \in Q} m_q = m$ and $W \leq n$, so the above complexity is bounded by $\mathcal{O}\big(m^2 \mathcal{C}_{sat}(n, k)\big)$. Out-degrees are, however, typically small constants.

The proof of the lemma can be found in [21]. Compared with the time complexity of LocalMin, we can see that, by sacrificing the use of the counters, the complexity becomes quadratic in the number of transitions (since the decrement of the counter on line 8 followed by the test of the counter being zero on line 9 in LocalMin is replaced by the computation of Γ on line 7 combined with the test on line 9 in NoCount). On the other hand, since we completely avoid mintermisation, the 2^{m_q} factors are lowered to at most m (m_q in the left-hand side term).

The overall worst-case complexity of NoCount is thus clearly better than those of GlobINY and LocalMin. Moreover, as shown in Sect. 4, NoCount is also winning in most of our experiments. Another advantage of avoiding mintermisation is that it often requires a lot of memory. Consequently, GlobINY and LocalMin can run out of memory before even finishing the mintermisation, which is also witnessed in our experiments. If m_q is small for all $q \in Q$ and the predicates do not intersect much, the number of generated minterms can, however, be rather small compared with the number of transitions, and LocalMin can in some cases win, as witnessed in our experiments too.

4 Experimental Evaluation

We now present an experimental evaluation of the algorithms from Sect. 3 implemented in the Symbolic Automata Toolkit [2]. All experiments were run on

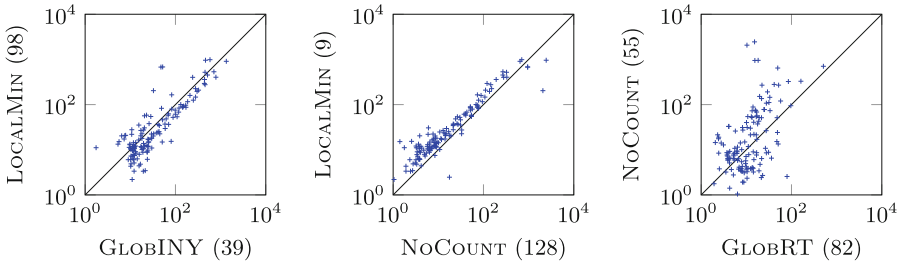


Fig. 1. Comparison of runtimes of algorithms on SFAs from RegEx. Times are in milliseconds (logarithmic scale).

an Intel Core i5-3230M CPU@2.6 GHz with 8 GiB of RAM. We used the following two benchmarks:

RegEx. We evaluated the algorithms on SFAs created from 1,921 regular expressions over the UTF-16 alphabet using the $\mathbf{BDD}_{16}$ algebra, which is the algebra of *binary decision diagrams* over 16 Boolean variables representing particular bits of the UTF-16 encoding. These regular expressions were taken from the website [15], which contains a library of regular expressions created for different purposes, such as matching email addresses, URIs, dates, times, street addresses, phone numbers, etc. The SFAs created from these regular expressions were used before when evaluating algorithms minimising (deterministic) SFAs [12] and when evaluating bisimulation algorithms for SFAs [13]. The largest automaton has 3,190 states and 10,702 transitions; the average transition density of the SFAs is 2.5 transitions per state. Since the UTF-16 alphabet is quite large, a symbolic representation is needed for efficient manipulation of these automata.
WS1S. For this benchmark, we used 131 SFAs generated when deciding formulae of the *weak-monadic second order logic of one successor* (WS1S) [16]. We used two batches of SFAs: 93 deterministic ones from the tool Mona [17] and 38 nondeterministic from dWiNA [18]. These automata have at most 2,508 states and 34,374 transitions with the average transition density of 6 transitions per state. These SFAs use the algebra $\mathbf{BDD}_k$ where k is the number of variables in the corresponding formula.

4.1 Comparison of Various Algorithms for Computing Simulation

We first evaluate the effect of our modifications of INY presented in Sect. 3. The results presented below clearly show the superiority of our new algorithms over GlobINY, with NoCount being the overall winner. In addition, we also compare the performance of our new algorithms to a version of the RT algorithm from [19], which is one of the best simulation algorithms. In particular, we use its adaptation for NFAs, which we run after global mintermisation (similarly as INY in GlobINY). We denote the whole combination GlobRT. RT is much faster than INY due to its use of the so-called partition-relation pairs to represent the intermediate preorder. Its *C++* implementation in the Vata library [9] is also much more optimised than the *C#* implementation of our algorithms.

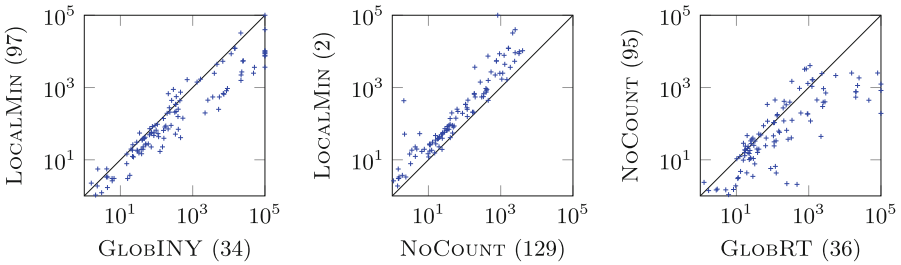


Fig. 2. Comparison of runtimes of algorithms on SFAs from WS1S. Times are in miliseconds (logarithmic scale).

Despite that, the comparison on automata with many global minterms is clearly favourable to our new algorithms.

To proceed to concrete data, Figs. 1 and 2 show scatter plots with the most interesting comparisons of the runtimes of the considered algorithms on our benchmarks (we give in parentheses the number of times the corresponding algorithm *won* over the other one). The timeout was set to 100 s. Fig. 1 shows the comparison of the algorithms on SFAs from the RegEx benchmark. In this experiment, we removed the SFAs where all algorithms finished within 10 ms (to mitigate the effect of imprecise measurement and noise caused by the *C#* runtime), which gave us 138 SFAs. Moreover, we also removed one extremely challenging SFA, which dominated the whole benchmark (we report on that SFA, denoted as M_c, later), which left us with the final number of 137 SFAs. On the other hand, Fig. 2 shows the comparison for WS1S. We observe the following phenomena: (i) LocalMin is in the majority of cases faster than GlobINY, (ii) NoCount clearly dominates LocalMin, and (iii) the comparison of NoCount and GlobRT has no clear winner: on the RegEx benchmark, GlobRT is more often faster, but on the WS1S benchmark, NoCount wins (in many cases, quite significantly).

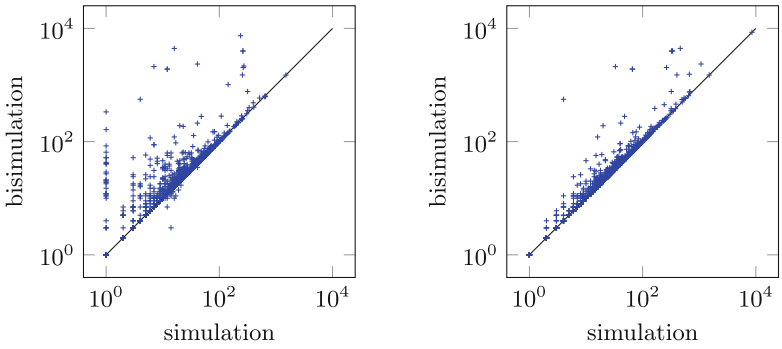
Further, we also give aggregated results of the experiment in Table 1. In the table, we accumulated the runtimes of the algorithms over the whole benchmark (column "time") and the number of times each algorithm was the best among all algorithms (column "wins"). The column "fails" shows how many times the respective algorithm failed (by being out of time or memory). In the parentheses, we give the number of times the failure occurred already in the mintermisation. When a benchmark fails, we assign it the time 100 s (the timeout) for the computation of "time". The times of the challenging SFA M_c from RegEx were: 21 s for GlobINY, 16 s for LocalMin, 25 s for NoCount, and 148 s for GlobRT. Obviously, including those times would bias the whole evaluation.

Observe that in this comparison, the performance of the algorithms on the two benchmarks differs—although GlobRT wins on the RegEx benchmark and the other three algorithms have a comparable overall time (but NoCount still wins in the majority of SFAs among the three), on the more complex benchmark (WS1S), NoCount is the clear winner. The distinct results on the two

Table 1. Aggregated results of the performance experiment.

Algorithm	RegEx		WS1S		
	Time	Wins	Time	Wins	Fails
GlobINY	12.3 s	2	1,258 s	1	9 (2)
LocalMin	11.9 s	0	316 s	0	1 (1)
NoCount	12.4 s	54	44 s	94	0 (0)
GlobRT	2.8 s	81	594 s	36	3 (2)

benchmark sets can be explained by a different diversity of predicates used on the transitions of SFA. In the RegEx benchmark, the globally mintermised automaton has on average 4.5 times more transitions (with the ratio ranging from 1 to 13), while in the WS1S benchmark, the mintermised automaton has on average 23.5 times more transitions (with the ratio ranging from 1 to 716). This clearly shows that our algorithms are effective in avoiding the potential blow-up of mintermisation. As expected, they are slower than RT on examples where the mintermisation is cheap since they do not use the partition-relation data structure.



(a) Iterative reduction. (b) One iteration only.

Fig. 3. Simulation vs. bisimulation-based reduction: the number of transitions of the reduced automaton.

4.2 Comparison of Simulation and Bisimulation

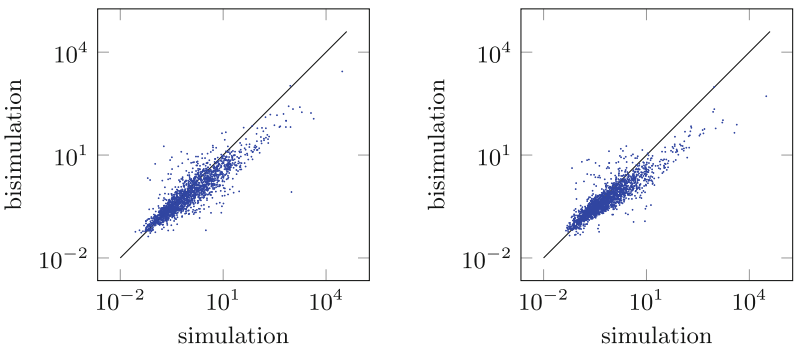
In the second experiment, we evaluate the benefit of computing simulation over computing bisimulation (we use the implementation of bisimulation computation from [13]). In particular, we focus on an application of (bi-)simulation for (language-preserving) reduction of SFAs from the whole RegEx benchmark.

For every SFA M from the benchmark, we compute its simulation preorder $\preceq_M$, take its biggest symmetric fragment (which constitutes an equivalence), and for each of its classes, merge all states of the class into a single state.

We also eliminate simulation subsumed transitions (the so-called *little brothers*) using the technique introduced in [20]. In particular, for a state q s.t. there exist transitions $q \dashv\{a\}\mapsto p$ and $q \dashv\{a\}\mapsto p'$ with $p \preceq p'$, we remove the transition $q \dashv\{a\}\mapsto p$ (and also the states that have become unreachable). After that, we reverse the automaton and repeat the whole procedure. These steps continue until the number of states no longer decreases. Similar steps apply to bisimulation (with the exception of taking the symmetric fragment and removing transitions as a bisimulation is already an equivalence).

The results comparing the number of transitions of the output SFAs are given in Fig. 3a, showing that the simulation-based reduction is usually much more significant.[2] Figure 3b shows the reduction after the first iteration (it corresponds to the "ordinary" simulation and bisimulation-based reduction).

The comparison of the numbers of states gives a very similar picture as the comparison of the numbers of transitions (cf. [21]) but simulation wins by a slightly larger margin when comparing the numbers of transitions. This is probably due to the use of the removal of simulation-subsumed transitions, which does not have a meaningful counterpart when working with bisimulations.



(a) Iterative reduction. (b) One iteration only.

Fig. 4. Simulation vs. bisimulation-based reduction: runtime in miliseconds.

As for the runtimes, they differ significantly on the different case studies with some of the cases won by the simulation-based reduction process, some by the bisimulation-based reduction, as can be seen in Fig. 4. Figure 4a shows comparison of runtimes for the whole iterative process, Fig. 4b shows the comparison for the first iteration only—essentially the time taken by computing the simulation preorder or the bisimulation equivalence. One may see that bisimulation is notably cheaper, especially when the automata are growing larger and

[2] There are still some cases when bisimulation achieved a larger reduction than simulation, which may seem unintuitive since the largest bisimulation is always contained in the simulation preorder. This may happen, e.g., when a simulation-based reduction disables an (even greater) reduction on the subsequent reversed SFA.

both algorithms are taking more time (note the logarithmic scale). Computing simulation was, however, faster in surprisingly many cases.

5 Conclusion and Future Work

We have introduced two new algorithms for computing simulation over symbolic automata that do not depend on global mintermisation: one that needs a local and cheaper variant of mintermisation, and one that does not need mintermisation at all. They perform well especially on automata where mintermisation significantly increases the number of transitions. In the future, we would like to come up with a partition-based algorithm that could run on an SFA without the need of mintermisation. Such algorithm might, but does not necessarily need to, be based on an NFA partition-based algorithm such as RT. Further, we wish to explore the idea of encoding NFAs over finite alphabets compactly as SFAs over a fast Boolean algebra (such as bit-vector encoding of sets) and compare the performance of our algorithms with known NFA simulation algorithms.

Acknowledgements. This paper was supported by the Czech Science Foundation projects 16-17538S and 16-24707Y, the IT4IXS: IT4Innovations Excellence in Science project (LQ1602), and the FIT BUT internal project FIT-S-17-4014.

References

1. Watson, B.W.: Implementing and Using Finite Automata Toolkits. Cambridge University Press, New York (1999)
2. Veanes, M., Bjørner, N.: Symbolic automata: the toolkit. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 472–477. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_33
3. Veanes, M.: Applications of symbolic finite automata. In: Konstantinidis, S. (ed.) CIAA 2013. LNCS, vol. 7982, pp. 16–23. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39274-0_3
4. D'Antoni, L., Veanes, M.: The power of symbolic automata and transducers. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 47–67. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_3
5. Abdulla, P.A., Chen, Y.-F., Holík, L., Mayr, R., Vojnar, T.: When simulation meets antichains. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 158–174. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12002-2_14
6. Bonchi, F., Pous, D.: Checking NFA equivalence with bisimulations up to congruence. In: Proceeding of POPL'13, ACM (2013)
7. Ilie, L., Navarro, G., Yu, S.: On NFA reductions. In: Karhumäki, J., Maurer, H., Păun, G., Rozenberg, G. (eds.) Theory Is Forever. LNCS, vol. 3113, pp. 112–124. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27812-2_11
8. Holík, L., Šimáček, J.: Optimizing an LTS-simulation algorithm. In: Masaryk, U. (ed.) Proceedings of MEMICS'09 (2009)
9. Lengál, O., Šimáček, J., Vojnar, T.: VATA: a library for efficient manipulation of non-deterministic tree automata. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 79–94. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_7

10. Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing simulations on finite and infinite graphs. In: Proceedings of FOCS'95, IEEE (1995)
11. Cécé, G.: Foundation for a series of efficient simulation algorithms. In: Proceedings of LICS'17, IEEE (2017)
12. D'Antoni, L., Veanes, M.: Minimization of symbolic automata. In: Proceedings of POPL'14, ACM (2014)
13. D'Antoni, L., Veanes, M.: Forward bisimulations for nondeterministic symbolic finite automata. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10205, pp. 518–534. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_30
14. Eberl, M.: Efficient and verified computation of simulation relations on NFAs. Bachelor's thesis, TU Munich, 2012
15. Regular expression library. http://regexlib.com/
16. Comon, H., et al.: Tree automata techniques and applications (2007)
17. Elgaard, J., Klarlund, N., Møller, A.: MONA 1.x: new techniques for WS1S and WS2S. In: Hu, A.J., Vardi, M.Y. (eds.) CAV 1998. LNCS, vol. 1427, pp. 516–520. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0028773
18. Fiedor, T., Holík, L., Lengál, O., Vojnar, T.: Nested antichains for WS1S. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 658–674. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_59
19. Ranzato, F.: A more efficient simulation algorithm on Kripke structures. In: Chatterjee, K., Sgall, J. (eds.) MFCS 2013. LNCS, vol. 8087, pp. 753–764. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40313-2_66
20. Bustan, D., Grumberg, O.: Simulation-based minimization. ACM Trans. Comput. Log. **4**(2), 181–206 (2003)
21. Holík, L., Lengál, O., Síč, J., Veanes, M., Vojnar, T.: Simulation algorithms for symbolic automata (Technical Report). CoRR, arXiv:abs/1807.08487 (2018)