Fundamenta Informaticae XXI (2001) 1001–1020 1001
IOS Press

Saving Space in a Time Efficient Simulation Algorithm

Silvia Crafa

Francesco Ranzato

Francesco Tapparo
Dipartimento di Matematica Pura ed Applicata
University of Padova, Italy

Abstract. A number of algorithms for computing the simulation preorder on Kripke structures and on labelled transition systems are available. Among them, the algorithm by Ranzato and Tapparo [2007] has the best time complexity, while the algorithm by Gentilini et al. [2003] – successively corrected by van Glabbeek and Ploeger [2008] – has the best space complexity. Both space and time complexities are critical issues in a simulation algorithm, in particular memory requirements are crucial in the context of model checking when dealing with large state spaces. Here, we propose a new simulation algorithm that is obtained as a space saving modification of the time efficient algorithm by Ranzato and Tapparo: a symbolic representation of sets is embedded in this algorithm so that any set of states manipulated by the algorithm can be efficiently stored as a set of blocks of a suitable state partition. It turns out that this novel simulation algorithm has a space complexity comparable with Gentilini et al.'s algorithm while improving on Gentilini et al.'s time bound.

Keywords: Simulation preorder, simulation algorithm, coarsest partition problem.

1. Introduction

The simulation preorder is widely used both as a behavioural relation between concurrent systems [7, 14, 17] and in model checking as an appropriate abstraction to reduce state spaces [2, 5]. In particular, in model checking one is often interested in quotienting the concrete state space w.r.t. simulation equivalence. The simulation problem consists in computing the greatest simulation preorder R_{sim} among the states of a given Kripke structure or of a given labelled transition system (LTS). Simulation equivalence is then the equivalence relation obtained as symmetric reduction of the preorder R_{sim}, namely $R_{\mathrm{sim}} \cap R_{\mathrm{sim}}^{-1}$. It is not hard to reduce the simulation problem for LTSs to that for Kripke structures, and conversely, algorithms on Kripke structures can be easily adapted to LTSs.

A simulation algorithm should address both time and space efficiency issues, since memory requirement is clearly a critical problem in the context of model checking. Among the algorithms for computing the simulation preorder, the most well known are by Henzinger, Henzinger and Kopke [13], Bloom and Paige [3], Bustan and Grumberg [4], Tan and Cleaveland [18], Gentilini, Piazza and Policriti [10] and Ranzato and Tapparo [15]. Let Σ denote the state space, $\rightarrow$ the transition relation and P_{sim} the partition of Σ induced by simulation equivalence. As far as time complexity is concerned, the most efficient algorithm is that by Ranzato and Tapparo [15], here denoted by RT, which runs in $O(|P_{\mathrm{sim}}||{\rightarrow}|)$-time and $O(|P_{\mathrm{sim}}||\Sigma|\log|\Sigma|)$-space, where a bit space complexity measure is considered. On the other hand, the algorithm by Gentilini, Piazza and Policriti [10], that was originally flawed and has been successively corrected by van Glabbeek and Ploeger [11], has the best space complexity in $O(|P_{\mathrm{sim}}|^2+|\Sigma|\log|P_{\mathrm{sim}}|)$. However, Gentilini et al.-van Glabbeek and Ploeger's algorithm, here denoted by GPP-GP, runs in $O(|P_{\mathrm{sim}}|^2|{\rightarrow}|)$-time and in this respect is therefore significantly less efficient than RT.

The efficiency of the best available simulation algorithms RT and GPP-GP actually depends on the fact that they solve the simulation problem in terms of a so-called coarsest partition problem. In this equivalent formulation, the simulation relation is efficiently represented in a symbolic way through a partition P of the state space Σ and a relation $R \subseteq P \times P$ among the blocks of P. In these algorithms, two distinct states s and t that will be eventually determined to be simulation equivalent are symbolically represented as belonging to a same block of the partition P instead of being separately represented in an explicit way. On the other hand, the relation R between blocks of P allows us to represent the simulation preorder as a relation between blocks of simulation equivalent states. The space complexity crucially depends on this kind of representation: while an explicit representation of a relation on the state space Σ takes $O(|\Sigma|^2)$-space, a partition-relation pair $\langle P, R\rangle$ can be represented in $O(|\Sigma|\log|P_{\mathrm{sim}}| + |P_{\mathrm{sim}}|^2)$-space, where $|\Sigma|\log|P_{\mathrm{sim}}|$ accounts for the relation that maps each state in Σ to the block of P containing it and $|P_{\mathrm{sim}}|^2$ is the space needed for storing the relation R.

Compared to GPP-GP, the algorithm RT relies on a faster approach to solve a coarsest partition problem. Very roughly, both algorithms iteratively refine a partition-relation pair $\langle P, R\rangle$ that is maintained as an over-approximation of $\langle P_{\mathrm{sim}}, R_{\mathrm{sim}}\rangle$. The main difference is that RT also stores and maintains some additional information that is needed for refining $\langle P, R\rangle$ and this allows us to save the time to recompute at any iteration such information from scratch. On the other hand, this time benefit gives rise to the space loss of RT w.r.t. GPP-GP, which is therefore due to the need of RT of storing this additional information.

Here, we propose a simulation algorithm, denoted by SA, that keeps as much as possible the fast approach of RT andimproves the space complexity of RT. The main feature of SA is the use of an additional state partition Q to symbolically represent all the concrete information maintained between different iterations. Interestingly, it turns out that it is sufficient to define Q as the coarsest partition that refines P and strongly progresses (in the sense of Sangiorgi and Walker [16]) to P, meaning that Q is stable w.r.t. the predecessor sets of the blocks of P. We show that in SA such a compact representation of the information needed for computing the simulation preorder leads to an effective space saving w.r.t. RT, but needs additional time to update it. More precisely, let P_{sp} be the coarsest partition refinement of P_{sim} that strongly progresses to P_{sim} and let $\rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}$ denote the existential abstract transition relation between P_{sp} and P_{sim}, i.e., for any $B \in P_{\mathrm{sp}}$ and $C \in P_{\mathrm{sim}}$, $(B, C) \in \rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}$ iff there exist $s \in B$ and $t \in C$ such that $s \rightarrow t$. Then, the space complexity of SA is $O((|P_{\mathrm{sim}}||P_{\mathrm{sp}}| + |\Sigma|)\log|P_{\mathrm{sp}}|)$ while its time complexity is $O(|P_{\mathrm{sim}}||{\rightarrow}| + |P_{\mathrm{sim}}|^2|{\rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}}|)$. The cubic factor $|P_{\mathrm{sim}}|^2|{\rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}}|$ represents the price to pay for saving space in RT. While in general $|P_{\mathrm{sim}}| \leq |P_{\mathrm{sp}}|$, we show experimentally on a

significant set of benchmarks that for relatively large state spaces $|P_{\mathrm{sim}}| \approx |P_{\mathrm{sp}}|$, since their sizes differ on average by less than 1%. Hence, on the one hand SA retains a space complexity comparable with that of GPP-GP while improving the time complexity of GPP-GP, and on the other hand SA improves the space complexity of RT while significantly worsening the time complexity of RT. We think that this algorithm SA therefore sheds new light on the trade-off between time and space efficiency in the simulation problem.

The rest of the paper is organized as follows. Section 2 introduces the notation and the algorithms SA is based on. Section 3 illustrates the new algorithm SA together with a simple example. Section 4 proves the correctness and complexity bounds of SA and describes how to adapt it to LTSs. Finally, Section 5 reports on the experimental evaluation of SA on a number of benchmarks and discusses the trade-off between time and space efficiency between the algorithms SA, RT and GPP-GP. A preliminary version of this paper appeared in [8].

2. Background

2.1. Notation

Let Σ be a finite state space. Given a relation $R \subseteq \Sigma \times \Sigma$, the set $R(s) \stackrel{\mathrm{def}}{=} \{t \in \Sigma \mid (s,t) \in R\}$ is the R-image of $s \in \Sigma$. A partition P of Σ is a set of nonempty subsets of Σ, called blocks, that are pairwise disjoint and whose union gives Σ. $\mathrm{Part}(\Sigma)$ denotes the set of partitions of Σ. Hence, given $P \in \mathrm{Part}(\Sigma)$, $P(s)$ denotes the block of P that contains s. If $P, Q \in \mathrm{Part}(\Sigma)$ then $Q \preceq P$, i.e. P is coarser than Q (or Q refines P), if for any $s \in \Sigma$, $Q(s) \subseteq P(s)$. If $Q \preceq P$ and $B \in Q$ then $\mathrm{parent}_P(B)$ denotes the unique block in P that contains B. Given $P \in \mathrm{Part}(\Sigma)$ and a subset $S \subseteq \Sigma$, called splitter, $\mathit{Split}(P, S)$ denotes the partition obtained from P by replacing each block $B \in P$ with its nonempty subsets $B \cap S$ and $B \smallsetminus S$, where we also allow no splitting, namely $\mathit{Split}(P, S) = P$ (this happens precisely when S is a union of some blocks of P).

We consider finite transition systems $(\Sigma, \rightarrow)$ consisting of a finite set of states Σ and a transition relation $\rightarrow \subseteq \Sigma \times \Sigma$. The predecessor/successor transformers $\mathrm{pre}, \mathrm{post} : \wp(\Sigma) \to \wp(\Sigma)$ (sometimes also denoted by $\mathrm{pre}_{\rightarrow}, \mathrm{post}_{\rightarrow}$) are defined on the powerset of Σ as usual, i.e., for $Y \subseteq \Sigma$, $\mathrm{pre}(Y) \stackrel{\mathrm{def}}{=} \{s \in \Sigma \mid \exists t \in Y.\ s \rightarrow t\}$ and $\mathrm{post}(Y) \stackrel{\mathrm{def}}{=} \{t \in \Sigma \mid \exists s \in Y.\ s \rightarrow t\}$. Notice that both pre and post are additive operators, i.e., they preserve unions of sets. By $s \rightarrow Y$ we mean that $s \in \mathrm{pre}(Y)$. We denote by $\rightarrow_\exists$ and $\rightarrow_\forall$, respectively, the abstract existential and universal transition relations on sets of states, i.e., if $S_1, S_2 \subseteq \Sigma$ then $S_1 \rightarrow_\exists S_2$ when $S_1 \cap \mathrm{pre}(S_2) \neq \varnothing$ and $S_1 \rightarrow_\forall S_2$ when $S_1 \subseteq \mathrm{pre}(S_2)$. If $P, Q \in \mathrm{Part}(\Sigma)$ then $\rightarrow^{P,Q} \subseteq P \times Q$ denotes the abstract existential transition relation between blocks in P and Q.

Given a set AP of atomic propositions (of some temporal language), a Kripke structure $(\Sigma, \rightarrow, \ell)$ over AP consists of a transition system $(\Sigma, \rightarrow)$ together with a state labeling function $\ell : \Sigma \to \wp(AP)$. We use the following notation: for any $s \in \Sigma$, $[s]_\ell \stackrel{\mathrm{def}}{=} \{s' \in \Sigma \mid \ell(s) = \ell(s')\}$ denotes the equivalence class of a state s w.r.t. the labeling ℓ, while $P_\ell \stackrel{\mathrm{def}}{=} \{[s]_\ell \mid s \in \Sigma\} \in \mathrm{Part}(\Sigma)$ is the state partition induced by ℓ.

2.2. Simulation

A relation $R \subseteq \Sigma \times \Sigma$ is a simulation on a Kripke structure $\mathcal{K} = (\Sigma, \rightarrow, \ell)$ if for any $s \in \Sigma$:

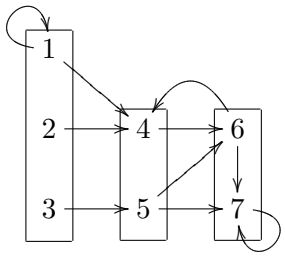

Figure 1. A Kripke structure.

(1) $R(s) \subseteq [s]_\ell$;

(2) for any $t \in \Sigma$, if $s \to t$ then $R(s) \to_\forall R(t)$.

A state $s' \in R(s)$ is called a simulator of s. The empty relation is a simulation and simulation relations are closed under union, so that the largest simulation relation exists. It turns out that the largest simulation is a preorder (i.e., a reflexive and transitive relation) called simulation preorder (on $\mathcal{K}$) and denoted by R_{sim}. Thus, for any $s \in \Sigma$, $R_{\mathrm{sim}}(s)$ is the set of all simulators of s. Simulation equivalence $\sim_{\mathrm{sim}} \subseteq \Sigma \times \Sigma$ is the symmetric reduction of R_{sim}, namely $\sim_{\mathrm{sim}} \stackrel{\mathrm{def}}{=} R_{\mathrm{sim}} \cap R_{\mathrm{sim}}^{-1}$ and $P_{\mathrm{sim}} \in \mathrm{Part}(\Sigma)$ denotes the state partition corresponding to the equivalence $\sim_{\mathrm{sim}}$.

Let us recall that a relation $R \subseteq \Sigma \times \Sigma$ is a bisimulation if both R and R^{-1} are simulation relations. The largest bisimulation relation exists and is an equivalence relation called bisimulation equivalence whose corresponding state partition is denoted by $P_{\mathrm{bis}} \in \mathrm{Part}(\Sigma)$. Let us also recall the notion of strong (bisimulation) progression [16]. Given two relations $R, S \subseteq \Sigma \times \Sigma$, R strongly progresses to S when $(s,t) \in R$ implies that:

(1) if $s \to v$ then there exists $w \in \Sigma$ such that $t \to w$ and $(v,w) \in S$;

(2) if $t \to w$ then there exists $v \in \Sigma$ such that $s \to v$ and $(v,w) \in S$.

When R and S specialize to the equivalence relations induced, respectively, by two partitions Q and R, we obtain the following notion of strongly progressing partitions: the partition Q strongly progresses to P when for all $B \in P$ and $E \in Q$, if $E \cap \mathrm{pre}(B) \neq \varnothing$ then $E \subseteq \mathrm{pre}(B)$. It is easy to see that the partition P_{bis} strongly progresses to both P_{sim} and P_{bis} itself.

As a running example, consider the Kripke structure in Figure 1, where $\Sigma = \{1,2,3,4,5,6,7\}$ and the labeling ℓ induces the partition $P_\ell = \{\{1,2,3\},\{4,5\},\{6,7\}\}$, whose blocks are depicted as rectangles. In this example, the simulation preorder R_{sim} is given by $R_{\mathrm{sim}}(1) = \{1\}$, $R_{\mathrm{sim}}(2) = R_{\mathrm{sim}}(3) = \{1,2,3\}$, $R_{\mathrm{sim}}(4) = R_{\mathrm{sim}}(5) = \{4,5\}$, $R_{\mathrm{sim}}(6) = \{6\}$ and $R_{\mathrm{sim}}(7) = \{6,7\}$. Hence, the simulation partition is $P_{\mathrm{sim}} = \{\{1\},\{2,3\},\{4,5\},\{6\},\{7\}\}$. We will describe in Section 3 how our algorithm SA computes the simulation preorder on this example.

2.3. Simulation Algorithms

Let us recall the two simulation algorithms SA is based on, namely Henzinger, Henzinger and Kopke's algorithm HHK [13], and Ranzato and Tapparo's algorithm RT [15].

2.3.1. Algorithm HHK

The algorithm HHK [13] works by iteratively refining a relation between states of the input Kripke structure until a fixpoint is reached, that is, the simulation preorder R_{sim}. We do not need to recall the full HHK algorithm here but merely its main data structures. The current relation between states is explicitly represented through a family of sets of states $\langle Sim(s)\rangle_{s\in\Sigma}$ indexed on the whole state space Σ: for any state $s \in \Sigma$, $Sim(s) \subseteq \Sigma$ represents the current set of states that are candidates to simulate s. Thus, the current family of sets $\langle Sim(s)\rangle_{s\in\Sigma}$ represents a relation that approximates from above the simulation preorder, i.e., for any state $s \in \Sigma$, $R_{\mathrm{sim}}(s) \subseteq Sim(s)$. At any iteration of HHK, some set $Sim(s)$ is selected and pruned so that the output family of sets provides exactly the simulation preorder R_{sim}. The $O(|\Sigma||{\to}|)$-time implementation of HHK relies on the maintenance of the following information, that is used for refining the current relation Sim:

(A) For any state $t \in \Sigma$, HHK maintains a set of states $Remove(t) \subseteq \Sigma$ such that if $s \to t$ then $Sim(s)$ is pruned to $Sim(s) \smallsetminus Remove(t)$.

(B) HHK maintains an integer matrix $Count(s,t)$, indexed on states $s, t \in \Sigma$, such that $Count(s,t) = |\operatorname{post}(s) \cap Sim(t)|$, i.e. $Count(s,t)$ stores the number of transitions from s to some state in $Sim(t)$.

These data structures are crucial for obtaining the $O(|\Sigma||{\to}|)$ time bound time. However, they have the serious drawback of a quadratic space complexity. In fact, HHK needs $\Omega(|\Sigma|^2 \log |\Sigma|)$-space because the integer matrix $Count$ is explicitly stored and takes exactly $|\Sigma|^2 \log |\Sigma|$ space, $\langle Sim(s)\rangle_{s\in\Sigma}$ needs $|\Sigma|^2$ space in the worst case and $\langle Remove(s)\rangle_{s\in\Sigma}$ takes $|\Sigma|^2 \log |\Sigma|$ space in the worst case.

2.3.2. Algorithm RT

The algorithm RT [15] is recalled in Figure 2. RT is obtained as a modification of HHK that is based on the idea of representing an over-approximation of the simulation preorder R_{sim} through a so-called partition-relation pair $\langle P, Rel\rangle$ (PR for short), where $P \in \mathrm{Part}(\Sigma)$ is a partition of Σ and $Rel \subseteq P \times P$ is a reflexive relation on P. A PR $\langle P, Rel\rangle$ induces the following relation $R_{\langle P,Rel\rangle} \subseteq \Sigma \times \Sigma$ between states:

$$R_{\langle P,Rel\rangle}(s) \stackrel{\mathrm{def}}{=} \cup \{B \in P \mid (P(s), B) \in Rel^*\}$$

where Rel^* is the transitive closure of Rel. It turns out that $R_{\langle P,Rel\rangle}$ is a preorder relation. Hence, while in HHK the family of sets $\langle Sim(s)\rangle_{s\in\Sigma}$ is an over-approximation of the simulation preorder R_{sim}, in RT we have that the current PR $\langle P, Rel\rangle$ approximates from above R_{sim}, namely, $R_{\mathrm{sim}} \subseteq R_{\langle P,Rel\rangle}$. More precisely, this means that the partition P is maintained as an over-approximation of the simulation partition P_{sim}, i.e. $P_{\mathrm{sim}} \preceq P$, so that if $P(s) \neq P(t)$ then s and t are not simulation equivalent. On the other hand, the reflexive relation Rel is maintained in such a way that Rel^* is an over-approximation of the simulation preorder, meaning that if a pair (B, C) of blocks of P does not belong to the relation Rel, then no state $s \in B$ is simulated by a state $t \in C$.

Partition-relations pairs are partially ordered as follows: $\langle P', Rel'\rangle$ is coarser than $\langle P, Rel\rangle$ when $P \prec P'$ or $P = P'$ implies $Rel \subseteq Rel'$. It turns out (see [15, Lemma 4.4]) that the simulation problem can be equivalently formulated as the problem of finding the coarsest partition-relation pair that represents the simulation preorder, namely, a pair $\langle P, Rel\rangle$ such that:

(1) $P \preceq P_\ell$;

```
1  RT(PartitionRelation ⟨P, Rel⟩) {
2    forall B ∈ P do Remove(B) := Σ ∖ pre(∪Rel(B));
3    initializeCount();
4    while ∃B ∈ P such that Remove(B) ≠ ∅ do
5      Remove := Remove(B);
6      Remove(B) := ∅;
7      B_prev := B;
8      P_prev := P;
9      P := Split(P, Remove);
10     newBlocks_P := P ∖ P_prev;
11     if newBlocks_P ≠ ∅ then
12       updateRel();
13       updateCount();
14       updateRemove();
15     RemoveList := {D ∈ P | D ⊆ Remove};
16     forall C ∈ P such that C ∩ pre(B_prev) ≠ ∅ do
17       forall D ∈ RemoveList such that Rel(C, D) do
18         Rel(C, D) := ff;
19         forall s ∈ pre(D) do
20           Count(s, C)--;
21           if Count(s, C) = 0 then Remove(C) := Remove(C) ∪ {s};
22 }
```

Figure 2. RT Simulation Algorithm.

(2) $\forall B, C \in P.\ B \rightarrow_{\exists} C \Rightarrow \cup Rel(B) \rightarrow_{\forall} \cup Rel(C)$.

The algorithm RT takes in input a PR $\langle P_\ell, Rel_\ell \rangle$ determined by the state labeling ℓ, where P_ℓ is the partition induced by ℓ and Rel_ℓ is the identity relation, i.e. $Rel_\ell = \{(B, B) \mid B \in P_\ell\}$. RT iteratively refines the current PR $\langle P, Rel \rangle$ until a fixpoint is reached, namely when $\langle P, Rel \rangle$ represents a simulation relation.

In RT, representing the relations as partition-relation pairs allows to save both space and time w.r.t. HHK. The intuition is that refining a partition and the corresponding relation between blocks can be more efficient than refining a relation between states. Consider the example in Figure 1: the states 4 and 5 have the same label as well as the same set of simulators, hence they may correctly (and efficiently) be represented as a single (abstract) state. Storing a PR $\langle P, Rel \rangle$ rather that an explicit relation $\langle Sim(s) \rangle_{s \in \Sigma}$ on the whole state space, requires $|P_{\mathrm{sim}}|^2$ space instead of $|\Sigma|^2$. Moreover, Rel is stored as a resizable boolean matrix so that insert operations take amortized constant time.

As the blocks of P play the role of (abstract) states, similarly to HHK, RT maintains for any block $B \in P$ a set of remove states $Remove(B) \subseteq \Sigma$ and an integer table $Count(s, B)$, indexed on states $s \in \Sigma$ and blocks $B \in P$, where $Count(s, B)$ stores the number of transitions from s to some state in $\cup Rel(B)$, i.e., $Count(s, B) \stackrel{\text{def}}{=} \sum_{C \in Rel(B)} |\operatorname{post}(s) \cap C|$. These two data structures need, respectively, $O(|P_{\mathrm{sim}}||\Sigma| \log |\Sigma|)$ and $O(|\Sigma||P_{\mathrm{sim}}| \log |\Sigma|)$ space. Finally, each state must have a pointer to the block of P containing it. Overall, it turns out that the space complexity of RT is therefore in

$O(|\Sigma||P_{\mathrm{sim}}|\log|\Sigma|)$. It is worth remarking that we are considering here a space bit-complexity measure where integers and pointers cannot be stored in constant space.

As far as time complexity is concerned, the $O(|P_{\mathrm{sim}}||{\rightarrow}|)$ time bound of RT crucially depends on the following properties.

- The $Count$ table allows to perform the test $Count(s, C) = 0$ at line 21 in constant time. This test logically corresponds to check whether $s \notin \mathrm{pre}(\cup Rel(C))$.
- The update functions at lines 12-14 for the data structures Rel, $Count$ and $Remove$ after a split operation work as follows: if a block B is split into two new blocks B_1 and B_2 then $Rel(B_i) = Rel(B)$, $Remove(B_i) = Remove(B)$ and $Count(s, B_i) = Count(s, B)$.
- If B_i and B_j are two blocks successively selected by the main while-loop at line 4 such that $B_i \subseteq B_j$ then we have that $Remove(B_i) \cap Remove(B_j) = \varnothing$. As a consequence of this property:
  - ◇ A split operation $Split(P, Remove)$ at line 9 can be done in $O(|\ Remove\ |)$-time. Thus, the overall cost of all the split operations is in $O(|P_{\mathrm{sim}}||\Sigma|)$-time.
  - ◇ If some block D is selected at line 17 as a subset of $Remove(B)$ then for any block $B' \subseteq B$ which is selected at some successive iteration of the main while-loop it turns out that any block E contained in D cannot be selected at line 17 as a subset of $Remove(B')$. This is the key property to prove that the overall time complexity of the for-loop at lines 16-21 is in $O(|P_{\mathrm{sim}}||{\rightarrow}|)$.

3. A New Simulation Algorithm

As discussed above, RT has been designed as a symbolic partition-based version of HHK, where two distinct states s and t that will be eventually determined to be simulation equivalent are symbolically represented as belonging to a same block of a state partition P. However, RT maintains two data structures that are still partially explicit: (1) in the family $\{Remove(B)\}_{B\in P}$ any $Remove(B) \subseteq \Sigma$ is represented as a set of explicit states; (2) the rows of the integer table $\{Count(s, B)\}_{s\in\Sigma, B\in P}$ are explicitly indexed on the whole state space Σ. We introduce here a new simulation algorithm that makes RT fully symbolic. The idea is to use an additional state partition Q in order to symbolically represent the data structure $Remove$ in terms of blocks of Q and to replace the semi-symbolic integer table $Count$ with a fully symbolic data structure that represents the abstract transition relation between blocks of Q and P. This allows us to save space in RT, although this space saving will give rise to a price to pay in time complexity for maintaining these fully symbolic data structures. This new simulation algorithm, called SA, is described in Figure 3.

3.1. Embedding an additional partition in RT

The basic idea of the algorithm SA is to maintain an additional state partition Q, finer than the partition P already maintained by the algorithm RT, that allows us to have the following invariant property: for any $B \in P$, any remove set $Remove(B)$ can be represented as (the union of) a set of blocks of Q. Of course, such a partition Q should be as coarse as possible in order to have this invariant property satisfied.

```
1  SA(PartitionRelation ⟨P, Rel⟩) {
2    Q := P;
3    forall C ∈ P do Q := Split(Q, pre(C));
4    computePreEE(P);
5    initializeRemove();
6    while ∃B ∈ P such that (Remove(B) ≠ ∅) do
7      Remove := Remove(B);
8      Remove(B) := ∅;
9      B_prev := B;
10     P_prev := P;
11     P := Split(P, ∪Remove);
12     newBlocks_P := P ∖ P_prev;
13     Q_prev := Q;
14     forall C ∈ newBlocks_P do
15       Remove(C) := Remove(parent_{P_prev}(C));
16       Q := Split(Q, pre(C));
17     newBlocks_Q := Q ∖ Q_prev;
18     if newBlocks_P ≠ ∅ then
19       updateRel();
20       if newBlocks_Q ≠ ∅ then
21         computePreEE(P);
22         updateRemove();
23       else
24         computePreEE(newBlocks_P);
25     RemoveList := {D ∈ P | D ⊆ ∪Remove};
26     forall C ∈ P such that C ∩ pre(B_prev) ≠ ∅ do
27       S := ∅;
28       forall D ∈ RemoveList such that Rel(C, D) do
29         Rel(C, D) := ff;
30         forall E ∈ preEE(D) do S := S ∪ {E};
31       if S ≠ ∅ then
32         forall D ∈ P such that Rel(C, D) do
33           forall E ∈ preEE(D) do S := S ∖ {E};
34         forall E ∈ S do Remove(C).append(E);
35 }
```

Figure 3. SA Simulation Algorithm.

The algorithm RT initializes and modifies a remove set at lines 2, 6 and 21, whereas the updateRemove() function at line 14 does not affect the remove sets. At line 2, $Remove(B)$ is initialized as $\Sigma \setminus \mathrm{pre}(\cup Rel(B))$. Then, after selecting a block $B \in P$, at line 6 RT make the corresponding $Remove(B)$ empty. Finally, a remove set $Remove(C)$ can be modified at line 21, where the states in the set $\mathrm{pre}(D) \setminus \mathrm{pre}(\cup Rel(C))$, for some $D \in P$, are added to $Remove(C)$. Recall that any set $\cup Rel(B) \subseteq \Sigma$ is a union of blocks of P and that the predecessor operator pre is additive. Thus, in order to represent the remove

sets through sets of blocks of Q, it is enough to have a partition Q such that any predecessor set $\mathrm{pre}(C)$, for some block $C \in P$, is a union of blocks of Q. This leads to define Q as the coarsest refinement of P that satisfies this condition. Let us observe that the stability of Q w.r.t. the predecessor sets of the blocks of P precisely corresponds to the notion of strong progression [16]. Hence, Q can be equivalently defined as the coarsest partition that refines P and strongly progresses to P. We denote this partition by $\mathrm{sp}(P)$ so that the invariant property that we have to guarantee becomes $Q = \mathrm{sp}(P)$. This is achieved by the algorithm SA as follows:

- The initialization of Q is performed at lines 2 and 3 by iteratively splitting P for all the predecessor sets $\mathrm{pre}(C)$, where $C \in P$. This guarantees exactly that $Q = \mathrm{sp}(P)$ initially holds.
- On the other hand, the partition P is refined by the split operation at line 11. Hence, in order to satisfy the invariant $Q = \mathrm{sp}(P)$ we need to split the current partition Q for all the predecessor sets $\mathrm{pre}(C)$ where C is a newly generated block of P at line 11. This is done by the for-loop at lines 14-16.

This allows us to represent any remove set $Remove(B)$ as a set of blocks of the partition Q. At the exit of SA, the partition Q will be $P_{\mathrm{sp}} \stackrel{\mathrm{def}}{=} \mathrm{sp}(P_{\mathrm{sim}})$, namely the coarsest partition refinement of P_{sim} that strongly progresses to P_{sim}. Such a representation of the remove sets therefore takes $O(|P_{\mathrm{sim}}||P_{\mathrm{sp}}| \log |P_{\mathrm{sp}}|)$ space. Since P_{bis} strongly progresses to P_{sim}, we have that $P_{\mathrm{bis}} \preceq P_{\mathrm{sp}}$ so that $P_{\mathrm{bis}} \preceq P_{\mathrm{sp}} \preceq P_{\mathrm{sim}}$ holds. Hence, $|P_{\mathrm{sim}}| \leq |P_{\mathrm{sp}}| \leq |P_{\mathrm{bis}}|$. To the best of our knowledge, there is no theoretical estimate of $|P_{\mathrm{sp}}|$ in terms of $|P_{\mathrm{sim}}|$. As expected, we observed experimentally (see Section 5) that $|P_{\mathrm{sp}}|$ tends to be much closer to $|P_{\mathrm{sim}}|$ rather than to $|P_{\mathrm{bis}}|$, so that $|P_{\mathrm{sim}}||P_{\mathrm{sp}}|$ is comparable to $|P_{\mathrm{sim}}|^2$.

3.2. Exploiting abstract transitions

The semi-symbolic integer table $\{Count(s, B)\}_{s\in\Sigma, B\in P}$ is used in RT at line 21 to test in constant time whether a state $s \in \mathrm{pre}(D)$ is such that $s \in \mathrm{pre}(\cup Rel(C))$ and this is done in order to add all the states in $\mathrm{pre}(D) \smallsetminus \mathrm{pre}(\cup Rel(C))$ to $Remove(C)$. In SA, the remove sets are represented in terms of blocks of Q, and since Q is stable w.r.t. blocks of P, we have that if some state $s \in \mathrm{pre}(D) \smallsetminus \mathrm{pre}(\cup Rel(C))$ must be added to $Remove(C)$, then the whole block of Q that contains s can be inserted. Instead of adding single states to $Remove(C)$ we then add entire blocks E of Q, and this must be done exactly when no state in E can reach a state in $\cup Rel(C)$, i.e., when $E \not\to_\exists \cup\, Rel(C)$. We therefore need to determine the blocks E of Q that belong to $\mathrm{pre}_{\to_\exists}(D) \smallsetminus \mathrm{pre}_{\to_\exists}(\cup\, Rel(C))$. Similarly to RT, in order to test whether $E \to_\exists \cup Rel(C)$ we could rely on a symbolic table $\{Count(E, B)\}_{E\in Q, B\in P}$ such that $Count(E, B)$ stores the number of abstract transitions $\to_\exists$ from the block $E \in Q$ to some block $B \in Rel(C)$. However, differently from the explicit $Count$ table in RT, this kind of abstract information cannot be easily updated through different iterations, hence in SA such an abstract $Count$ table should be recomputed each time from scratch. We then take a different approach: we store and maintain the abstract transition relation $\to_\exists \subseteq Q \times P$ from blocks of Q to blocks of P so that we can compute on-the-fly the set of blocks

$$\{E \in Q \mid E \to_\exists D,\ E \not\to_\exists \cup Rel(C)\}$$

to be added to $Remove(C)$. More precisely, the relation $\to_\exists$ is stored as the predecessor $preEE$: for any block $B \in P$, $preEE(B)$ is a list containing all the blocks E in Q such that $E \to_\exists B$. Now, the for-loop at lines 28-30, while refining the relation Rel, collects in the set S (this can be implemented

```
computePreEE(ListOfBlocks L) {
  forall B ∈ L do preEE(B) := ∅;
  forall B ∈ L do
    forall y ∈ B do
      forall x ∈ pre({y}) such that Q(x) unmarked do
        preEE(B).append(Q(x));
        mark(Q(x));
    forall E ∈ preEE(B) do unmark(E);
}

initializeRemove() {
  forall B ∈ P do
    forall C ∈ P such that Rel(B, C) do
      forall E ∈ preEE(C) do mark(E);
    forall E ∈ Q do
      if E marked then
        unmark(E);
      else
        Remove(B).append(E);
}

updateRel() {
  Resize the matrix Rel;
  forall B ∈ newBlocks_P do
    forall C ∈ P do
      Rel(B, C) := Rel(parent_{P_prev}(B), parent_{P_prev}(C));
  forall C ∈ newBlocks_P do
    forall B ∈ P ∖ newBlocks_P do
      Rel(B, C) := Rel(parent_{P_prev}(B), parent_{P_prev}(C));
}

updateRemove() {
  forall B ∈ P do
    forall E ∈ Remove(B) do
      replace E with {F ∈ newBlocks_Q | F ⊆ E};
}
```

Figure 4. Auxiliary functions.

through suitable markings of blocks) the blocks E that have an abstract transition to one of the blocks D that have been removed from $Rel(C)$. Then, lines 31-34 add to $Remove(C)$ those blocks in S that have no further abstract transitions to some block that still belongs to $Rel(C)$. We show in the next section that, even if a piece of information is computed on-the-fly, exploiting $preEE$ leads to an overall time complexity that still improves that of the GPP-GP algorithm. Moreover, storing the data structure

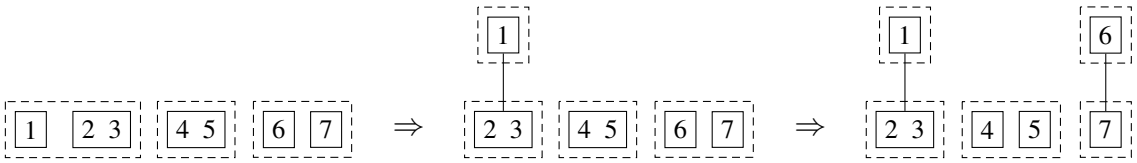

Figure 5. Refinements of the partition-relation.

$preEE$ takes exactly $|{\rightarrow}^{P_{\rm sp},P_{\rm sim}}| \log |P_{\rm sp}|$ space (notice that $|{\rightarrow}^{P_{\rm sp},P_{\rm sim}}| \leq |{\rightarrow}|$ always holds).

The function computePreEE() that computes the data structure $preEE$ is given in Figure 4. Notice that given a list L of blocks of P a call computePreEE(L) computes the predecessor list $preEE(B)$ for all the blocks $B \in L$. The function initializeRemove() in Figure 4 initializes $\{Remove(B)\}_{B\in P}$ taking advantage of the data structure $preEE$. The updateRel() function in Figure 4 updates the resizable boolean matrix Rel and works exactly as in RT. It is called at line 19 each time the partition P is refined. Finally, the updateRemove() function in Figure 4 is called at line 22 each time the partition Q is refined and simply replaces in all the remove lists the old blocks of Q with the corresponding new blocks of Q that are generated by a split operation. Notice at lines 14-16 that the partition Q can be refined only when P is refined.

3.3. An example

Let us illustrate an example showing how SA computes the simulation preorder for the Kripke structure in Figure 1. The diagrams in Figure 5 show how the computation of SA refines the current partitions P and Q and the relation Rel between blocks of P, where dashed, respectively solid, boxes represent blocks of the partition P, respectively Q, and if B and C are two blocks in P such that $C \in Rel(B)$ then C is drawed above B. Blocks are compactly denoted whitout curly brackets and commas.

The input PR is $\langle P_\ell, Rel_\ell \rangle$ where $P_\ell = \{B_1, B_2, B_3\}$ with $B_1 = 123$, $B_2 = 45$, $B_3 = 67$, and Rel_ℓ is the identity relation on the blocks of P_ℓ. The initialization phase computes the partition Q by splitting P_ℓ w.r.t. the predecessor sets of every block of P_ℓ. The resulting partition is $Q = \{E_0, E_1, E_2, E_3, E_4\}$ with $E_0 = 1$, $E_1 = 23$, $E_2 = 45$, $E_3 = 6$, $E_4 = 7$, which is strictly finer than P_ℓ. The situation is depicted by the diagram on the left of Figure 5. Next, for every block $B \in P_\ell$ the set $Remove(B)$ is built up as the set of blocks $E \in Q$ such that there is no existential transition from E to $Rel_\ell(B) = \{B\}$. This initializes the remove sets as follows:

$$Remove(B_1) = \{E_1, E_2, E_3, E_4\}, \quad Remove(B_2) = \{E_2, E_4\}, \quad Remove(B_3) = \{E_0, E_1\}.$$

Let the first iteration choose the block $B_1 \in P$. $Remove(B_1)$ is emptied and the initial partition P is split w.r.t. the set of states in $\cup_{i=1,..,4} E_i$. This yields the refined partition $P = \{B_1', B_1'', B_2, B_3\}$, where the block B_1 is split into $B_1' = 1$ and $B_1'' = 23$. Such a change for P does not affect the partition Q since $Q = \mathrm{sp}(P)$. On the other hand, Rel and $Remove$ are updated so that $Rel(B_1') = Rel(B_1'') = \{B_1', B_1''\}$ and $Remove(B_1') = Remove(B_1'') = \varnothing$. Next, the relation Rel is refined by removing the pairs of blocks $(C, D) \in P \times P$ such that $C \cap \mathrm{pre}(B_1) \neq \varnothing$ and $D \subseteq \cup_{i=1,4} E_i$: this removes the block B_1'' from $Rel(B_1')$. Moreover, since there is no $E \in Q$ such that $E \in preEE(B_1'')$ the remove relation is not

further modified. Summing up, at the end of the first iteration of the main while-loop, we have that:

$$\begin{array}{lcllcl} Rel(B_1') & = & \{B_1'\} & Remove(B_1') & = & \varnothing \\ Rel(B_1'') & = & \{B_1', B_1''\} & Remove(B_1'') & = & \varnothing \\ Rel(B_2) & = & \{B_2\} & Remove(B_2) & = & \{E_2, E_4\} \\ Rel(B_3) & = & \{B_3\} & Remove(B_3) & = & \{E_0, E_1\} \end{array}$$

as depicted by the diagram in the middle of Figure 5.

Let the second iteration choose the block $B_2 \in P$ so that $Remove(B_2)$ is set to empty and P is split w.r.t. $E_2 \cup E_4 = \{4, 5, 7\}$. The refined partition is $P = \{B_1', B_1'', B_2, B_3', B_3''\}$, where the block B_3 is split into $B_3' = 6$ and $B_3'' = 7$. Consequently, Q is split w.r.t. $\mathrm{pre}(B_3'') = \{5, 6, 7\}$ and is therefore refined to $Q = \{E_0, E_1, E_2', E_2'', E_3, E_4\}$ with $E_2' = 4$, $E_2'' = 5$. Rel and $Remove$ are updated so that $Rel(B_3') = Rel(B_3'') = \{B_3', B_3''\}$ and $Remove(B_3') = Remove(B_3'') = \{E_0, E_1\}$. The lines 26-34 of this second iteration remove B_3'' from $Rel(B_3')$ and adds the blocks E_3 and E_4 to $Remove(B_3')$. At the end of the second iteration, we have then:

$$\begin{array}{lcllcl} Rel(B_1') & = & \{B_1'\} & Remove(B_1') & = & \varnothing \\ Rel(B_1'') & = & \{B_1', B_1''\} & Remove(B_1'') & = & \varnothing \\ Rel(B_2) & = & \{B_2\} & Remove(B_2) & = & \varnothing \\ Rel(B_3') & = & \{B_3'\} & Remove(B_3') & = & \{E_0, E_1, E_3, E_4\} \\ Rel(B_3') & = & \{B_3', B_3''\} & Remove(B_3') & = & \{E_0, E_1\} \end{array}$$

as depicted by the diagram on the right of Figure 5. The third iteration chooses $B_3' \in P$ and leaves untouched both P and Q and the relation Rel. A final iteration chooses the unique block whose remove set is nonempty, that is B_3'', and does not modify P, Q and Rel. The output of SA is therefore as follows:

$$P_{\mathrm{sim}} = \{1, 23, 45, 6, 7\};$$
$$P_{\mathrm{sp}} = \{1, 23, 4, 5, 6, 7\};$$
$$Rel(1) = \{1\},\ Rel(23) = \{1, 23\},\ Rel(45) = \{45\},\ Rel(6) = \{6\},\ Rel(7) = \{6, 7\}.$$

Let us notice that $P_{\mathrm{sp}} \prec P_{\mathrm{sim}}$ and that $P_{\mathrm{bis}} \prec P_{\mathrm{sp}}$ because $P_{\mathrm{bis}} = \{1, 2, 3, 4, 5, 6, 7\}$. This example shows the space saving of SA compared to both RT and HHK. Compared to the explicit algorithm HHK, in SA (likewise RT) the sets of equivalent states $\{2, 3\}$ and $\{4, 5\}$ are represented as blocks. On the other hand, SA obtains a similar space saving over RT by representing the remove sets as sets of blocks of Q instead of as sets of explicit states, for instance, $Remove(B_1)$ is represented as $\{E_1, E_2, E_3, E_4\}$ instead of $\{2, 3, 4, 5, 6, 7\}$.

4. Correctness and Complexity

It turns out that the correctness of SA follows as a consequence of the correctness of RT and of the properties described above.

Theorem 4.1. (Correctness)
For any finite Kripke structure $\mathcal{K}$, the algorithm SA on input $\langle P_\ell, Rel_\ell \rangle$ always terminates and outputs a PR that induces the simulation preorder R_{sim} on $\mathcal{K}$.

Proof:
The proof relies on the correctness of the RT algorithm on input $\langle P_\ell, Rel_\ell \rangle$ and on a correspondence between the iterations of SA and RT. Let $\{\langle P^i_{\mathrm{SA}}, Rel^i_{\mathrm{SA}}\rangle\}_{i\in[1,n]}$ and $\{\langle Remove^i_{\mathrm{SA}}(B)\rangle_{B\in P^i_{\mathrm{SA}}}\}_{i\in[1,n]}$ be the sequence of PRs and remove sets computed in some sequence of iterations of SA, where $\langle P^i_{\mathrm{SA}}, Rel^i_{\mathrm{SA}}\rangle$ and $\langle Remove^i_{\mathrm{SA}}(B)\rangle_{B\in P^i_{\mathrm{SA}}}$ are the PR and the remove sets at the entry of the i-th iteration of the main while-loop of SA, so that $\langle P^1_{\mathrm{SA}}, Rel^1_{\mathrm{SA}}\rangle = \langle P_\ell, Rel_\ell\rangle$. Then, we show that RT can correspondingly compute a sequence of PRs $\{\langle P^i_{\mathrm{RT}}, Rel^i_{\mathrm{RT}}\rangle\}_{i\in[1,n]}$ and remove sets $\{\langle Remove^i_{\mathrm{RT}}(B)\rangle_{B\in P^i_{\mathrm{RT}}}\}_{i\in[1,n]}$ such that, for any $i \in [1, n]$:

(1) $P^i_{\mathrm{SA}} = P^i_{\mathrm{RT}}$;

(2) for any $B \in P^i_{\mathrm{SA}} = P^i_{\mathrm{RT}}$, $Rel^i_{\mathrm{SA}}(B) = Rel^i_{\mathrm{RT}}(B)$;

(3) for any $B \in P^i_{\mathrm{SA}} = P^i_{\mathrm{RT}}$, $\cup Remove^i_{\mathrm{SA}}(B) = Remove^i_{\mathrm{RT}}(B)$.

As a consequence of this, both termination and correctness of SA follow, respectively, from termination and correctness of RT.
First of all notice that from the invariant property $Q = \mathrm{sp}(P)$ of SA discussed above we also have the following invariant property of the main while-loop of SA:

$$\text{for any } B \in P,\ \mathrm{pre}(B) \text{ and } \mathrm{pre}(Rel_{\mathrm{SA}}(B)) \text{ are unions of some blocks of } Q \qquad (*)$$

At the beginning, i.e., for $i = 1$, it is clear that $P^1_{\mathrm{SA}} = P^1_{\mathrm{RT}} = P_\ell$ and $Rel^1_{\mathrm{SA}} = Rel^1_{\mathrm{RT}} = Rel_\ell$. As far as property (3) is concerned, note that $Remove^1_{\mathrm{RT}}(B) = \Sigma \smallsetminus \mathrm{pre}(\cup Rel^1_{\mathrm{RT}}(B))$ and $Remove^1_{\mathrm{SA}}(B) = \{E \in Q^1_{\mathrm{SA}} \mid E \not\doteq_\exists Rel^1_{\mathrm{SA}}(B)\}$. By property $(*)$ and since $Rel^1_{\mathrm{SA}} = Rel^1_{\mathrm{RT}}$ we have that $\Sigma \smallsetminus \mathrm{pre}(\cup Rel^1_{\mathrm{RT}}(B))$ is the union of the set of blocks E of Q such that $E \not\doteq_\exists Rel_{\mathrm{SA}}(B)$, that is, $\cup Remove^1_{\mathrm{SA}}(B) = Remove^1_{\mathrm{RT}}(B)$.
Assume now that the three properties hold at the beginning of the i-th iteration of SA and RT and let us prove that they still hold at the end of the i-th iteration.

(1) Let $B \in P^i_{\mathrm{SA}}$ be the block chosen by SA such that $Remove^i_{\mathrm{SA}}(B) \neq \varnothing$. By inductive hypothesis (1) and (3), $P^i_{\mathrm{SA}} = P^i_{\mathrm{RT}}$ so that $B \in P^i_{\mathrm{RT}}$ and $Remove^i_{\mathrm{RT}}(B) \neq \varnothing$. Then, SA and RT update their partition as follows: P^i_{SA} is refined to $Split(P^i_{\mathrm{SA}}, \cup\, Remove^i_{\mathrm{SA}}(B))$ and P^i_{RT} is refined to $Split(P^i_{\mathrm{RT}}, Remove^i_{\mathrm{RT}}(B))$. Hence, $P^{i+1}_{\mathrm{SA}} = P^{i+1}_{\mathrm{RT}}$.

(2) Both SA and RT update the relation Rel in two steps. First, for every block C that is split into two new blocks C_1 and C_2, both algorithms set $Rel(C_1) = Rel(C_2) = Rel(C)$. The second step may prune a set $Rel(C)$, for all $C \in P$ such that $C \cap \mathrm{pre}(B_{\mathrm{prev}}) \neq \varnothing$. In particular, SA and RT remove from $Rel(C)$, respectively, the sets $\{D \in P^{i+1}_{\mathrm{SA}} \mid D \subseteq \cup Remove^i_{\mathrm{SA}}(B_{\mathrm{prev}})\}$ and $\{D \in P^{i+1}_{\mathrm{RT}} \mid D \subseteq Remove^i_{\mathrm{RT}}(B_{\mathrm{prev}})\}$. By point (1) above and by inductive hypothesis on property (3), these two sets are identical, so that $Rel^{i+1}_{\mathrm{SA}}(C) = Rel^{i+1}_{\mathrm{RT}}(C)$.

(3) As far as the remove sets are concerned, the i-th iteration in SA updates the remove sets as follows: (i) $Remove^{i+1}_{\mathrm{SA}}(B) = \varnothing$; (ii) if a block $C \in P$ is split into two new blocks C_1 and C_2 then $Remove^{i+1}_{\mathrm{SA}}(C_1) = Remove^{i+1}_{\mathrm{SA}}(C_2) = Remove^i_{\mathrm{SA}}(C)$ (line 15) and these sets are possibly updated at line 22 so that they contain blocks of the possibly updated partition Q; and (iii) for all

$C \in P$ sucht that $C \cap \mathrm{pre}(B_{\mathrm{prev}}) \neq \varnothing$, the following set of blocks of Q is added to $Remove^{i}_{\mathrm{SA}}(C)$: $\{E \in Q \mid E \rightarrow_{\exists} Remove^{i}_{\mathrm{SA}}(B),\ E \not\rightarrow_{\exists} \cup Rel^{i+1}_{\mathrm{SA}}(C)\}$.

Similarly, the i-th iteration in RT makes the following updates: (i) $Remove^{i+1}_{\mathrm{RT}}(B) = \varnothing$; (ii) for every block C that is split into two new blocks C_1 and C_2, $Remove^{i+1}_{\mathrm{RT}}(C_1) = Remove^{i+1}_{\mathrm{RT}}(C_2) = Remove^{i}_{\mathrm{RT}}(C)$; and (iii) for all $C \in P$ such that $C \cap \mathrm{pre}(B_{\mathrm{prev}}) \neq \varnothing$, the following set is added to $Remove^{i}_{\mathrm{RT}}(C)$: $\{s \in \Sigma \mid s \rightarrow Remove^{i}_{\mathrm{RT}}(B),\ s \not\rightarrow \cup Rel^{i+1}_{\mathrm{RT}}(C)\} = \mathrm{pre}(Remove^{i}_{\mathrm{RT}}(B)) \smallsetminus \mathrm{pre}(\cup Rel^{i+1}_{\mathrm{RT}}(C))$. Clearly, after the first two steps (i) and (ii) the remove sets remain the same in both algorithms because by inductive hypothesis $\cup Remove^{i}_{\mathrm{SA}} = Remove^{i}_{\mathrm{RT}}$. Let $Rem = \cup Remove^{i}_{\mathrm{SA}} = Remove^{i}_{\mathrm{RT}}$. We have shown at point (2) that $Rel^{i+1}_{\mathrm{SA}}(C) = Rel^{i+1}_{\mathrm{RT}}(C)$, so we denote them by $Rel(C)$. It is therefore enough to check that:

$$\cup\{E \in Q \mid E \rightarrow_{\exists} Rem,\ E \not\rightarrow_{\exists} \cup Rel(C)\} = \mathrm{pre}(Rem) \smallsetminus \mathrm{pre}(\cup Rel(C)).$$

($\subseteq$) Since $E \not\rightarrow_{\exists} \cup Rel(C)$ we have that $E \cap \mathrm{pre}(\cup\, Rel(C)) = \varnothing$. Since $E \rightarrow_{\exists} Rem$, we have that $E \cap \mathrm{pre}(D) \neq \varnothing$ for some $D \in P$ such that $D \subseteq Rem$. Since $E \in Q$ and $Q = \mathrm{sp}(P)$, from $E \cap \mathrm{pre}(D) \neq \varnothing$ we have that $E \subseteq \mathrm{pre}(D) \subseteq \mathrm{pre}(Rem)$. Hence, $E \subseteq \mathrm{pre}(Rem) \smallsetminus \mathrm{pre}(\cup Rel(C))$.

($\supseteq$) Notice that Rem is a union of a set of blocks of P, hence we have that $\mathrm{pre}(Rem)$ is a union of a set of blocks of Q since $Q = \mathrm{sp}(P)$, . Moreover, by property $(*)$ above, we also have that $\mathrm{pre}(\cup Rel(C))$ is a union of set of blocks of Q. Hence, if $s \in \mathrm{pre}(Rem) \smallsetminus \mathrm{pre}(\cup Rel(C))$ then $Q(s) \rightarrow_{\exists} Rem$ and $Q(s) \not\rightarrow_{\exists} \cup Rel(C)$.

□

Space and time bounds for SA are as follows, where we assume, as usual in model checking, that the transition relation $\rightarrow$ is total (viz., $\forall s \in \Sigma. \exists t \in \Sigma.\ s \rightarrow t$) so that $|\Sigma| \leq |\rightarrow|$ and $|P_{\mathrm{sp}}| \leq |\rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}| \leq |P_{\mathrm{sp}}||P_{\mathrm{sim}}|$ and this allows us to simplify the expression of the time bound.

Theorem 4.2. (Complexity)
The algorithm SA runs in $O((|P_{\mathrm{sp}}||P_{\mathrm{sim}}| + |\Sigma|) \log |P_{\mathrm{sp}}|)$-space and $O(|P_{\mathrm{sim}}||\rightarrow| + |P_{\mathrm{sim}}|^2|\rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}|)$-time.

Proof:
Space complexity. The algorithm SA relies on the following data structures, where a space bit-complexity measure is considered:

- The state partitions Q and P that take, respectively, $O(|P_{\mathrm{sp}}| \log |\Sigma|)$ and $O(|P_{\mathrm{sim}}| \log |\Sigma|)$.
- Any state in Σ has a pointer to the block of Q containing it and any block of Q has a pointer to the block of P containing it. These take, respectively, $O(|\Sigma| \log |P_{\mathrm{sp}}|)$ and $O(|P_{\mathrm{sp}}| \log |P_{\mathrm{sim}}|)$.
- The resizable boolean matrix $\{Rel(B,C)\}_{B,C \in P}$ that takes $O(|P_{\mathrm{sim}}|^2)$.
- The remove lists $\{Remove(B)\}_{B \in P}$ that takes $O(|P_{\mathrm{sim}}||P_{\mathrm{sp}}| \log |P_{\mathrm{sp}}|)$.
- The lists of existential predecessors $\{preEE(B)\}_{B \in P}$ that takes $O(|\rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}| \log |P_{\mathrm{sp}}|)$.

Thus, the overall space complexity is in $O((|P_{\rm sp}||P_{\rm sim}| + |\Sigma|)\log|P_{\rm sp}|)$.

Time complexity. Let us analyze the time complexity of SA. Some points of the proof are similar to analogous points of the proof of complexity of RT in [15, Theorem 7.1]. For the sake of completeness, some of them are here recalled. Also, since the transition relation is supposed to be total, let us remark that $|Q| \leq |{\rightarrow^{Q,P}}|$ holds.
Let It denote the sequence of iterations of the main while-loop for some run of SA, where for any $i, j \in It$, $i < j$ means that j follows i. Moreover, for any $i \in It$, let B_i denote the block selected by the while-loop at line 6 and $\langle P^i, Rel^i\rangle$ denotes the PR at the entry point of the for-loop at line 26.
The key property of the remove sets is the following: For any $i, j \in It$,

$$B_i \subseteq B_j \ \ \& \ \ j < i \ \ \Rightarrow \ \ (\cup Remove(B_i)) \cap (\cup Remove(B_j)) = \varnothing \qquad (*)$$

Let us prove this property $(*)$. At iteration j, $Remove(B_j)$ is set to $\varnothing$ at line 8. If B_j generates, by the splitting operation at line 11, two new blocks $B_1, B_2 \subseteq B_j$ then their remove sets are set to $\varnothing$ at line 15. Successively, SA may add at line 34 of some iteration $k \geq j$ a block E to the remove set $Remove(C)$ of a block $C \subseteq B_j$ only if $E \in preEE(D)$ for some $D \in Rel^k(C)$. We also have that $\cup Rel^k(C) \subseteq \cup Rel^j(B_j)$ so that $\mathrm{pre}(\cup Rel^k(C)) \subseteq \mathrm{pre}(\cup Rel^j(B_j))$. Thus, if $B_i \subseteq B_j$ and $i > j$ then $\cup Remove(B_i) \subseteq \mathrm{pre}(\cup Rel^j(B_j))$. Therefore, $(\cup Remove(B_j)) \cap (\cup Remove(B_i)) \subseteq (\cup Remove(B_j)) \cap \mathrm{pre}(\cup Rel^j(B_j))$. Since $\cup Remove(B_j) \cap \mathrm{pre}(\cup Rel^j(B_j)) = \varnothing$ always holds at the beginning of the main while-loop, we have that $(\cup Remove(B_i)) \cap (\cup Remove(B_j)) = \varnothing$.
As a consequence of $(*)$, it turns out that $\sum_{i\in It} |\cup Remove(B_i)| \leq |P_{\rm sim}||\Sigma|$. Let us prove this inequality. For any block $E \in P_{\rm sim}$ of the final partition we define the following subset of iterations: $It_E \stackrel{\rm def}{=} \{i \in It \mid E \subseteq B_i\}$. Thus,

$$\begin{array}{rl} \sum_{i\in It} |\cup Remove(B_i)| \leq & \text{[by definition of } It_E\text{]} \\ \sum_{E\in P_{\rm sim}} \sum_{i\in It_E} |\cup Remove(B_i)| \leq & \text{[as the sets in } \{\cup Remove(B_i)\}_{i\in It_E} \text{ are pairwise disjoint]} \\ \sum_{E\in P_{\rm sim}} |\Sigma| = & \\ |P_{\rm sim}||\Sigma|. & \end{array}$$

Let us also show that the overall number of newly generated blocks by the splitting operation at line 11 is $2(|P_{\rm sim}| - |P_\ell|)$, namely it is in $O(|P_{\rm sim}|)$. In fact, let $\{P^i\}_{i\in[1,n]}$ be the sequence of partitions computed by SA where P^0 is the initial partition P_ℓ, P^n is the final partition $P_{\rm sim}$ and for all $i \in [0, n-1]$, $P^{i+1} \preceq P^i$. The number of newly generated blocks by one splitting operation that refines P^i to P^{i+1} is clearly given by $2(|P^{i+1}| - |P^i|)$, so that the overall number of newly generated blocks is $\sum_{i=0}^{n-1} 2(|P^{i+1}| - |P^i|) = 2(|P_{\rm sim}| - |P_\ell|)$.
The time complexity bound for SA is then shown by the following points.

- It is not hard to implement a standard splitting procedure $Split$ in such a way that any call $Split(P, S)$ scans all the states in S and takes $O(|S|)$ time (see [15, Section 7]).
- The initializations of Q (lines 2-3), $preEE$ (line 4), and $Remove$ (line 5) take time, respectively, $O(\sum_{B\in P} |\mathrm{pre}(B)|) = O(|{\rightarrow}|)$, $O(|{\rightarrow}|)$ and $O(|P||{\rightarrow^{Q,P}}|)$. Thus, the initialization phase takes $O(|{\rightarrow}| + |P||{\rightarrow^{Q,P}}|)$ time, i.e., $O(|{\rightarrow}| + |P_{\rm sim}||{\rightarrow^{P_{\rm sp},P_{\rm sim}}}|)$.

- Each call $Split(P, \cup Remove)$ at line 11 takes $O(|\cup Remove(B_i)|)$ time, so that the overall time complexity of line 14 is bounded by $\sum_{i \in It} |\cup Remove(B_i)| \leq |P_{\mathrm{sim}}||\Sigma|$.

- Each call to the function updateRel() at line 19 takes the time for resizing the boolean matrix Rel plus $|newBlocks_P||P_{\mathrm{sim}}|$. Since the boolean matrix that stores Rel is resizable, each operation at the first line of the function updateRel() that adds a new entry to this resizable matrix has an amortized cost which is in $O(|P_{\mathrm{sim}}|)$: in fact, the resizable matrix is just a resizable array A of resizable arrays so that when we add a new entry we need to add a new entry to A and then a new entry to each array in A. Thus, the overall time complexity of resizing the matrix Rel is in $O(|P_{\mathrm{sim}}|^2)$. Moreover, since $\sum_{i \in It} |newBlocks_{P_i}|$ is in $O|P_{\mathrm{sim}}|)$, we have that the overall time complexity of calling updateRel() at line 19 is in $O(|P_{\mathrm{sim}}|^2)$.

- Each split of the partition Q takes $O(|\mathrm{pre}(C)|)$ time and therefore $O(|{\to}|)$-time. This is done for all the new blocks of P so that the overall time complexity of splitting the partition Q at line 16 is in $O(|P_{\mathrm{sim}}||{\to}|)$-time.

- Each call to computePreEE() takes $O(|{\to}|)$ time. This function is called at lines 21 and 24 at most $O(|P_{\mathrm{sim}}|)$ time so that the overall time complexity of lines 21 and 24 is in $O(|P_{\mathrm{sim}}||{\to}|)$.

- Each call to updateRemove() takes $O(|P||Q|)$ time. This function is called at line 22 at most $O(|P_{\mathrm{sim}}|)$ time so that the overall time complexity of line 25 is in $O(|P_{\mathrm{sim}}|^2|P_{\mathrm{sp}}|)$.

- The difference between lines 26-30 in SA and lines 16-19 in RT, apart from the $O(1)$ operation $S := \varnothing$ at line 27, lies in the fact that at lines 16 and 19 in RT by means of $C \cap \mathrm{pre}(B_{\mathrm{prev}})$ and $s \in \mathrm{pre}(D)$ we traverse the concrete transition relation $\to$ while in SA at lines 26 and 30 by means of $C \cap \mathrm{pre}(B_{\mathrm{prev}})$ and $E \in preEE(D)$ we traverse instead the abstract existential transition relation $\to^{Q,P}$. As shown in [15, Theorem 7.1] — this proof is lengthy and not simple, we refer the interested reader to [15, Theorem 7.1] — the overall time complexity of lines 16-19 in RT is in $O(|P_{\mathrm{sim}}||{\to}|)$. As a consequence, the overall time complexity of lines 26-30 in SA is in $O(|P_{\mathrm{sim}}||{\to^{P_{\mathrm{sp}},P_{\mathrm{sim}}}}|)$.

- Lines 32-34 in SA are executed only when $S \neq \varnothing$. Notice that S can be nonempty only when at least a pair (C, D) has been removed from Rel at line 29. Thus, the overall number of times that lines 32-34 are executed is in $O(|P_{\mathrm{sim}}|^2)$. Moreover, lines 32-34 take $O(|{\to^{P,Q}}|)$ time. Therefore, the overall time complexity of lines 31-34 is in $O(|P_{\mathrm{sim}}|^2|{\to^{P_{\mathrm{sp}},P_{\mathrm{sim}}}}|)$.

Thus, summing up and recalling that, by totality of the transition relation, we have that $|P_{\mathrm{sim}}|^2|P_{\mathrm{sp}}| \leq |P_{\mathrm{sim}}|^2|{\to^{P_{\mathrm{sp}},P_{\mathrm{sim}}}}|$, it turns out that the time complexity of SA is in $O(|P_{\mathrm{sim}}||{\to}|+|P_{\mathrm{sim}}|^2|{\to^{P_{\mathrm{sp}},P_{\mathrm{sim}}}}|)$. □

Table 1 sums up the time and space complexities for the simulation algorithms RT, GPP-GP and SA.

4.1. Adapting SA to LTSs

The algorithms discussed so far compute the simulation preorder on Kripke structures, but they can be easily adapted to work over LTSs. The LTS version of the algorithm RT is given by Abdulla et al. [1]. For each block B, instead of a single set $Remove(B)$, this modified algorithm maintains a family of sets

Algorithm	Space complexity	Time complexity
RT	$O(\|P_{\rm sim}\|\|\Sigma\| \log \|\Sigma\|)$	$O(\|P_{\rm sim}\|\|\rightarrow\|)$
GPP-GP	$O(\|P_{\rm sim}\|^2 + \|\Sigma\| \log \|P_{\rm sim}\|)$	$O(\|P_{\rm sim}\|^2\|\rightarrow\|)$
SA	$O((\|P_{\rm sp}\|\|P_{\rm sim}\| + \|\Sigma\|) \log \|P_{\rm sp}\|)$	$O(\|P_{\rm sim}\|\|\rightarrow\| + \|P_{\rm sim}\|^2\|\rightarrow^{P_{\rm sp},P_{\rm sim}}\|)$

Table 1. Space and time complexities.

$Remove_a(B)$ indexed over the labels $a \in \mathfrak{L}$ of the LTS: a set $Remove_a(B)$ stores all the states that have no a-transition to a state that belong to some block C such that $(B, C) \in Rel$. Similarly, the $Count$ table is lifted to a table $\{Count_a(s, B)\}_{s\in\Sigma, B\in P, a\in\mathfrak{L}}$ whose additional third dimension takes into account the label of the transitions from s to some state in $\cup Rel(B)$. Both space and time complexities of this modified algorithm increase of a multiplicative factor $|\mathfrak{L}|$: the space complexity is $O(|\mathfrak{L}||P_{\rm sim}||\Sigma| \log |\Sigma|)$ while the time complexity is $O(|\mathfrak{L}||P_{\rm sim}||\Sigma| + |P_{\rm sim}||\rightarrow|)$.

Following the approach of [1], the algorithm SA can be adapted to LTSs as well. A first issue concerns the additional partition Q. More precisely, for any $B \in P$ and $a \in \mathfrak{L}$, any set $Remove_a(B)$ must still be represented as a set of blocks of the partition Q. In order to have this invariant satisfied, it is enough to consider a partition Q such that any a-predecessor set $\mathrm{pre}_a(C)$, for some label $a \in \mathfrak{L}$ and some block $C \in P$, is a union of blocks of Q. Hence, it is sufficient to iterate the split operations in lines 3 and 16 for any a-predecessor set $\mathrm{pre}_a(C)$, where a ranges over $\mathfrak{L}$. Moreover, the labeled abstract transition relation $\rightarrow^{Q,P} \subseteq Q \times P$ is represented by $\{preEE_a(B)\}_{B\in P, a\in\mathfrak{L}}$, where for any $E \in Q$, we have that $E \in preEE_a(B)$ iff there exists $s_1 \in E$ and $s_2 \in B$ such that $s_1 \xrightarrow{a} s_2$.

Analogously to RT, this change entails an increase for space and time complexities of a multiplicative factor $|\mathfrak{L}|$. More precisely, the space bound $O((|\mathfrak{L}||P_{\rm sp}||P_{\rm sim}| + |\Sigma|) \log |P_{\rm sp}|)$ takes into account the third dimension of the remove sets $\{Remove_a(B)\}_{B\in P, a\in\mathfrak{L}}$. On the other hand, the time complexity is $O(|P_{\rm sim}||\rightarrow| + |P_{\rm sim}|^2|\rightarrow^{P_{\rm sp},P_{\rm sim}}| + |\mathfrak{L}|(|P_{\rm sim}|^2|P_{\rm sp}| + |P_{\rm sim}||\Sigma|))$, and this takes into account the additional time spent to split the partition P, whose overall time complexity is $O(|\mathfrak{L}||P_{\rm sim}||\Sigma|)$, and to call the updateRemove() function, whose overall time complexity is $O(|\mathfrak{L}||P_{\rm sim}|^2|P_{\rm sp}|)$.

5. Experimental Evaluation and Discussion

A prototype of the SA algorithm has been developed in C++ and has been evaluated on a number of benchmarks. Our benchmarks include systems taken from publicly available examples in the VLTS (Very Large Transition Systems) benchmark suite [19], CWB-NC (The Concurrency WorkBench of New Century) [6] and mCRL2 (micro Common Representation Language 2) [12]. All these models are represented as labeled transition systems. Since we implemented the version of SA in Figure 3 that considers a Kripke structure as input, we exploit a procedure by Dovier et al. [9] that transforms a LTS M into a Kripke structure M' in such a way that bisimulation and simulation equivalences on M and M' coincide. This transformation acts as follows: any labeled transition $s_1 \xrightarrow{l} s_2$ is replaced by two unlabeled transitions $s_1 \rightarrow n$ and $n \rightarrow s_2$, where n is a new node that is labeled with l, while all the original states in M have the same label. This labeling provides an initial partition of the states of M' which is denoted by $P_{\rm init}$. Hence, this transformation grows the size of the system as follows: the number of transitions is doubled and the number of states of M' is the sum of the number of states and transitions

	Input			Output				Experim. Results	
Model	$\|\Sigma\|$	$\|\rightarrow\|$	$\|P_{\mathrm{init}}\|$	$\|P_{\mathrm{sim}}\|$	$\|P_{\mathrm{sp}}\|$	$\|P_{\mathrm{bis}}\|$	$\|\rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}\|$	Time(s)	Space(MB)
ABP-lossy [6]	187	260	4	32	32	32	48	0.20	0.02
ABP-lossy-2 [6]	1821	3280	12	1004	1004	1004	2456	0.24	0.65
ABP-safe [6]	123	148	4	40	40	40	53	0.10	0.02
two-link-netw [6]	8408	13638	4	336	375	448	883	0.32	0.76
one_pump [6]	15774	17926	22	3193	3193	3193	5345	35.74	5.75
cwi_1_2 [19]	4339	4774	27	2401	2401	2401	2701	1.45	1.66
cwi_3_14 [19]	18548	29104	3	123	123	123	122	0.91	1.68
vasy_0_1 [19]	1513	2448	3	21	21	21	32	0.10	0.13
vasy_1_4 [19]	5647	8928	7	87	87	87	118	0.51	0.44
vasy_8_24 [19]	33290	48822	12	1423	1423	1423	2200	5.61	3.42
vasy_8_38 [19]	47345	76848	82	963	963	963	1582	6.26	4.34
vasy_10_56 [19]	67005	112312	13	8048	8048	8048	17308	42.16	18.59
vasy_18_73 [19]	91789	146086	18	15618	15618	15618	27975	268.29	47.42
brp [12]	22716	24336	5	591	591	591	648	1.38	1.79
leader [12]	1520	2256	3	47	47	47	46	0.22	0.14
mpsu [12]	202	300	15	145	145	145	229	0.90	0.04
par [12]	209	236	6	58	58	58	67	0.50	0.03
parallel [12]	8000	14000	286	1540	1540	1540	2640	3.44	6.07
tree [12]	2049	2048	3	43	43	43	59	0.70	0.15
lift3-final [12]	14230	19836	17	1573	1573	1573	2388	2.27	1.57
cabp [12]	2096	3264	6	210	213	216	411	0.50	0.18
scheduler [12]	32	38	6	30	30	30	36	0.00	0.01
dining-phil-4 [12]	418	600	21	418	418	418	600	0.74	0.47
dining-phil-5 [12]	1642	2500	26	1642	1642	1642	2500	1.95	5.87

Table 2. Results of the experimental evaluation.

of M.

We ran our implementation of SA on M' on an Intel Core 2 Duo 1.86 GHz PC with 2GB RAM running Linux. Table 2 reports the size of the input model, the number of blocks of P_{sim}, P_{sp} and P_{bis}, the size of the existential transition relation $\rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}$ from P_{sp} to P_{sim}, the execution time in seconds and the allocated memory in MB (this has been obtained by means of glibc-memusage). Table 3 provides, respectively, the comparisons between the size of the partition P_{sp} w.r.t. the size of the state space Σ, the size of P_{sim} w.r.t. P_{sp} and the size of the abstract transition relation $\rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}$ w.r.t. the concrete transition relation $\rightarrow$. We may therefore observe that (1) the average reduction by P_{sp} of the concrete state space is 72.28%; (2) the average decrease of P_{sp} w.r.t. P_{sim} is 0.49%, namely the size of P_{sp} tends to be close to that of simulation equivalence P_{sim}; (3) the average decrease of the number of arcs of $\rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}$ w.r.t. $\rightarrow$ is 70.14%, i.e. $|\rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}| \approx |\rightarrow|/3$.

This suggests the following comparison of the theoretical time and space complexities between SA on one side and RT and GPP-GP on the other side, that we summarized above in Table 1. The time complexity of SA may significantly improve the time complexity of GPP-GP as much as the size of the abstract transition relation $\rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}$ is smaller than that of the concrete transition relation $\rightarrow$. However,

Model	Comparison		
	$\% \, \lvert P_{\mathrm{sp}} \rvert / \lvert \Sigma \rvert$	$\% \, \lvert P_{\mathrm{sim}} \rvert / \lvert P_{\mathrm{sp}} \rvert$	$\% \, \lvert \rightarrow^{P_{\mathrm{sp}}, P_{\mathrm{sim}}} \rvert / \lvert \rightarrow \rvert$
ABP-lossy [6]	17.11	100.00	18.46
ABP-lossy-2 [6]	55.13	100.00	74.88
ABP-safe [6]	32.52	100.00	35.81
two-link-netw [6]	4.46	89.60	6.47
one_pump [6]	20.24	100.00	29.82
cwi_1_2 [19]	55.34	100.00	56.58
cwi_3_14 [19]	0.66	100.00	0.42
vasy_0_1 [19]	1.39	100.00	1.31
vasy_1_4 [19]	1.54	100.00	1.32
vasy_8_24 [19]	4.27	100.00	4.51
vasy_8_38 [19]	2.03	100.00	2.06
vasy_10_56 [19]	12.01	100.00	15.41
vasy_18_73 [19]	17.02	100.00	19.15
brp [12]	2.60	100.00	2.66
leader [12]	3.09	100.00	2.04
mpsu [12]	71.78	100.00	76.33
par [12]	27.75	100.00	28.39
parallel [12]	19.25	100.00	18.86
tree [12]	2.10	100.00	2.88
lift3-final [12]	11.05	100.00	12.04
cabp [12]	10.16	98.59	12.59
scheduler [12]	93.75	100.00	94.74
dining-phil-4 [12]	100.00	100.00	100.00
dining-phil-5 [12]	100.00	100.00	100.00
Average	27.72	99.51	29.86

Table 3. Experimental comparison.

the time complexity of RT still continues to be the best. As far as space complexity is concerned, the space complexity of SA may be only moderately worse than the space complexity of GPP-GP because $|P_{\mathrm{sp}}| \approx |P_{\mathrm{sim}}|$. On the other hand, the space complexity of SA may significantly improve the space complexity of RT when P_{sp} provides a notable reduction of the concrete state space Σ. Moreover, in large models where RT runs out of memory, the space efficiency of SA might be crucial for the computation of the simulation preorder. For instance, if we consider the models vasy_10_56 and vasy_18_73, SA is able to compute the simulation preorder, whereas [15, Section 8] shows that RT fails in out of memory.

Acknowledgements. We are grateful to one of the anonymous referees for her/his valuable comments on the implementation of the algorithm SA. This work has been partially supported by the PRIN 2007 Project "*AIDA2007: Abstract Interpretation Design and Applications*" and by the University of Padova Project "*AVIAMO: Analysis, Verification and abstract InterpretAtion of MOdels for concurrency*".

References

[1] P.A. Abdulla, A, Bouajjani, L. Holík, L. Kaati and T. Vojnar. Computing simulations over tree automata. In *Proc. 14th TACAS*, LNCS 4963, pp. 93–108, 2008.

[2] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[3] B. Bloom and R. Paige. Transformational design and implementation of a new efficient solution to the ready simulation problem. *Sci. Comp. Program.*, 24(3):189-220, 1995.

[4] D. Bustan and O. Grumberg. Simulation-based minimization. *ACM Trans. Comput. Log.*, 4(2):181-204, 2003.

[5] E.M. Clarke, O. Grumberg and D.A. Peled. *Model Checking*. The MIT Press, 1999.

[6] R. Cleaveland and S Sims. The NCSU concurrency workbench. In *Proc. 8th CAV*, LNCS 1102, pp. 394-397, 1996. http://www.cs.sunysb.edu/∼cwb.

[7] R. Cleaveland and O. Sokolsky. Equivalence and preorder checking for finite-state systems. In *Handbook of Process Algebra*, chapter 6, pp. 391-424, Elsevier, 2001.

[8] S. Crafa, F. Ranzato and F. Tapparo. Saving space in a time efficient simulation algorithm. In *Proc. 9th ACSD*, pp. 60-69, IEEE Press, 2009.

[9] A. Dovier, C. Piazza and A. Policriti. An efficient algorithm for computing bisimulation equivalence. *Theor. Comput. Sci.*, 311(1-3):221-256, 2004.

[10] R. Gentilini, C. Piazza and A. Policriti. From bisimulation to simulation: coarsest partition problems. *J. Automated Reasoning*, 31(1):73-103, 2003.

[11] R. van Glabbeek and B. Ploeger. Correcting a space-efficient simulation algorithm. In *Proc. 20th CAV*, LNCS 5123, pp. 517-529, 2008.

[12] J.F. Groote, A. Mathijssen, M.A. Reniers, Y.S. Usenko and M. van Weerdenburg. The formal specification language mCRL2. In *Proc. Methods for Modelling Software Systems*, Dagstuhl Seminar Proceedings, 2006. http://www.mcrl2.org.

[13] M.R. Henzinger, T.A. Henzinger and P.W. Kopke. Computing simulations on finite and infinite graphs. In *Proc. 36th FOCS*, pp. 453–462, IEEE Press, 1995.

[14] A. Kučera and P. Jančar. Equivalence-checking on infinite-state systems: Techniques and results. *Theory Pract. Log. Program.*, 6(3):227-264, 2006.

[15] F. Ranzato and F. Tapparo. An efficient simulation algorithm based on abstract interpretation. *Inform. and Comput.*, 208(1):1–22, 2010. Preliminary version in *Proc. 22nd IEEE LICS'07*.

[16] D. Sangiorgi and D. Walker. *The Pi Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[17] R. Segala. The power of simulation relations. In *Proc. 27th PODC*, pp. 462-462, ACM Press, 2008.

[18] L. Tan and R. Cleaveland. Simulation revisited. In *Proc. 7th TACAS*, LNCS 2031, pp. 480–495, 2001.

[19] The VLTS Benchmark Suite. http://www.inrialpes.fr/vasy /cadp/resources/benchmark_bcg.html