Saving Space in a Time Efficient Simulation Algorithm

Silvia Crafa Francesco Ranzato Francesco Tapparo
Dipartimento di Matematica Pura ed Applicata
Università di Padova, Italy

Abstract

A number of algorithms are available for computing the simulation relation on Kripke structures and on labelled transition systems representing concurrent systems. 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. We propose here 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 new simulation algorithm retains a space complexity comparable with Gentilini et al.'s algorithm while improving on Gentilini et al.'s time bound.

1. Introduction

The simulation preorder is widely used both as a behavioural relation in concurrent systems [7], [13], [16] 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. More precisely, the simulation problem consists in computing the greatest simulation relation R_{sim} among the states of a given Kripke structure or of a given labelled transition system (LTS), where for any state s, $R_{\mathrm{sim}}(s)$ gives the set of states that simulate s. Simulation equivalence is then the equivalence relation obtained as symmetric reduction of the preorder R_{sim}, namely $R_{\mathrm{sim}} \cap R_{\mathrm{sim}}^{-1}$. Moreover, it is not hard to reduce the simulation problem for LTSs to that for Kripke sturctures, and conversely, algorithms working over 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 [12], Bloom and Paige [3], Bustan and Grumberg [4], Tan and Cleaveland [17], Gentilini, Piazza and Policriti [9] and Ranzato and Tapparo [14]. Let Σ denote the state space, $\rightarrow$ the transition relation and P_{sim} the partition of Σ induced by simulation equivalence. The algorithm by Ranzato and Tapparo [14], here denoted by RT, runs in $O(|P_{\mathrm{sim}}||{\rightarrow}|)$-time and $O(|P_{\mathrm{sim}}||\Sigma|\log|\Sigma|)$-space and is the best algorithm for what concerns time complexity. On the other hand, the algorithm by Gentilini, Piazza and Policriti [9], that was originally flawed and has been successively corrected by van Glabbeek and Ploeger [10], has the best space complexity $O(|P_{\mathrm{sim}}|^2 + |\Sigma|\log|P_{\mathrm{sim}}|)$. 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, where the simulation relation is symbolically represented by a state partition P and a relation $R \subseteq P \times P$ among the blocks of P. In these algorithms, the partition P is an over-approximation of (i.e., coarser than) simulation equivalence P_{sim}, where two distinct states s and t that will be eventually determined to be equivalent are symbolically represented as belonging to a same block of the partition P instead of being represented separately in an explicit way. On the other hand, the relation R between blocks of P represents the simulation relation between blocks of equivalent states. Space efficiency crucially depends on this kind of representation, since while a relation on the whole 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 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, in any partition refinement iteration, RT stores a piece of information that is useful for the next iteration, thus saving the time to recompute it from scratch. On the other hand, this time efficiency benefit gives rise to the space efficiency loss of RT w.r.t. GPP-GP, that is triggered by the need of RT of maintaining this information between different iterations.

We propose here a simulation algorithm, denoted by SA, that keeps as much as possible the fast approach of RT and improves the space complexity of RT. The main point of SA is to rely on 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 [15]) to P, meaning that Q is bisimulation stable for P but possibly not for itself. 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 has the major drawback of being time-consuming when updated. 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 $E \in P_{\mathrm{sp}}$ and $B \in P_{\mathrm{sim}}$, $(E, B) \in \rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}$ iff there exist $s \in E$ and $t \in B$ such that $s \rightarrow t$. Then, the space complexity of the new simulation algorithm SA is $O((|P_{\mathrm{sim}}||P_{\mathrm{sp}}| + |\Sigma|) \log |P_{\mathrm{sp}}|)$ while its time complexity is $O(|P_{\mathrm{sim}}||{\rightarrow}| + |P_{\mathrm{sim}}|^2|{\rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}}|)$-time. Thus, 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 (see Section 4) 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% (and are often equal). 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 present the notation and the algorithms SA is based on. Section 3 illustrates the new algorithm: it shows an execution on a simple example, proves its correctness and complexity bounds, and hints at how to adapt it to LTSs. A final section reports on the experimental evaluation of SA on a number of concurrent benchmarks, and discusses the trade-off between time and space efficiency between the algorithms SA, RT and GPP-GP.

2. Background

2.1. Basics

$\mathrm{Part}(\Sigma)$ denotes the set of partitions of the state space Σ. If $P_1, P_2 \in \mathrm{Part}(\Sigma)$ then $P_1 \preceq P_2$, i.e. P_2 is coarser than P_1 (or P_1 refines P_2) if $\forall B \in P_1. \exists B' \in P_2. B \subseteq B'$. If $P_1, P_2 \in \mathrm{Part}(\Sigma)$, $P_1 \preceq P_2$ and $B \in P_1$ then $\mathrm{parent}_{P_2}(B)$ denotes the unique block in P_2 that contains B. For a given subset $S \subseteq \Sigma$ called splitter, we denote by $\mathit{Split}(P, S)$ 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). Given a relation $R \subseteq \Sigma \times \Sigma$, the set $R(s) \overset{\mathrm{def}}{=} \{t \in \Sigma \mid (s, t) \in R\}$ is the R-image of $s \in \Sigma$. For a partition $P \in \mathrm{Part}(\Sigma)$, $P(s)$ denotes the block of P that contains s.

We consider finite transition systems $(\Sigma, \rightarrow)$. The predecessor/successor transformers $\mathrm{pre}, \mathrm{post} : \wp(\Sigma) \rightarrow \wp(\Sigma)$ are defined as usual. Notice that both pre and post are additive operators, namely they preserve unions of sets. We denote by $\rightarrow_\exists$ and $\rightarrow_\forall$, respectively, the abstract existential and universal transition relations, 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 $P, Q \in \mathrm{Part}(\Sigma)$ are two partitions then $\rightarrow_\exists^{P,Q} \subseteq P \times Q$ denotes the abstract existential transition relation between the blocks in P and the blocks in Q.

Given a set AP of atomic propositions (of some specification language), a Kripke structure $(\Sigma, \rightarrow, \ell)$ over AP consists of a transition system $(\Sigma, \rightarrow)$ together with a state labeling function $\ell : \Sigma \rightarrow \wp(AP)$. We use the following notation: for any $s \in \Sigma$, $[s]_\ell \overset{\mathrm{def}}{=} \{s' \in \Sigma \mid \ell(s) = \ell(s')\}$ denotes the equivalence class of a state s w.r.t. the labeling ℓ, while $P_\ell \overset{\mathrm{def}}{=} \{[s]_\ell \mid s \in \Sigma\} \in \mathrm{Part}(\Sigma)$ is the 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 $s \in \Sigma$: (1) $R(s) \subseteq [s]_\ell$; (2) for any $t \in \Sigma$, if $s \rightarrow t$ then $R(s) \rightarrow_\forall R(t)$. $R(s)$ is called a simulator set 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 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 the states that simulate s. Simulation equivalence $\sim_{\mathrm{sim}} \subseteq \Sigma \times \Sigma$ is the symmetric reduction of R_{sim}, namely $\sim_{\mathrm{sim}} \overset{\mathrm{def}}{=} R_{\mathrm{sim}} \cap R_{\mathrm{sim}}^{-1}$ and $P_{\mathrm{sim}} \in \mathrm{Part}(\Sigma)$ denotes the partition corresponding to the equivalence $\sim_{\mathrm{sim}}$. The relation R 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 recall the notion of strong (bisimulation) progression [15]. Consider two relations $R, S \subseteq \Sigma \times \Sigma$. The relation R strongly progresses to S when $(s, t) \in R$ implies that (1) if $s \rightarrow v$ then there exists w such that $t \rightarrow w$ and $(v, w) \in S$ and (2) if $t \rightarrow w$ then there exists v such that $s \rightarrow v$ and $(v, w) \in S$.

As a running example, consider the Kripke structure in

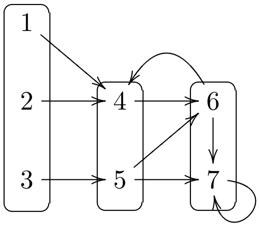

Figure 1. Example

Figure 1, where $\Sigma = \{1,2,3,4,5,6,7\}$ and the labeling induces the partition $P_\ell = \{\{1,2,3\},\{4,5\},\{6,7\}\}$. In this example, the simulation equivalence is given by the partition $P_{\mathrm{sim}} = \{\{1,2,3\},\{4,5\},\{6\},\{7\}\}$, and the simulation preorder is given by $R_{\mathrm{sim}}(1) = R_{\mathrm{sim}}(2) = R_{\mathrm{sim}}(3) = \{1,2,3\}$, $R_{\mathrm{sim}}(4) = R_{\mathrm{sim}}(5) = \{4,5\}$, $R_{\mathrm{sim}}(6) = \{6\}$ and $R_{\mathrm{sim}}(7) = \{6,7\}$. In Section 3 we will show how this simulation preorder is computed by the new algorithm SA.

2.3. Simulation Algorithms

The rest of the section illustrates the two simulation algorithms SA is based on, namely Henzinger, Henzinger and Kopke's algorithm HHK [12], and Ranzato and Tapparo's algorithm RT [14], that can be viewed as a symbolic partition-based version of HHK.

Algorithm HHK**.** The algorithm HHK [12] iteratively refines it a current relation between states until a fixpoint, i.e. the simulation preorder R_{sim}, is reached. In HHK, the current relation between states is explicitly represented through a family of sets of states $\langle \mathit{Sim}(s)\rangle_{s\in\Sigma}$ indexed on the whole state space Σ, where, for any state $s \in \Sigma$, $\mathit{Sim}(s) \subseteq \Sigma$ represents the current set of states that are candidates to simulate s. Thus, the current family of sets $\langle \mathit{Sim}(s)\rangle_{s\in\Sigma}$ is a preorder relation that approximates from above the simulation preorder, i.e., for any state s, $R_{\mathrm{sim}}(s) \subseteq \mathit{Sim}(s)$. At any iteration of HHK, some set $\mathit{Sim}(s)$ is selected and pruned so that the output family of sets provides exactly the simulation preorder R_{sim}. The $O(|\Sigma||{\rightarrow}|)$-time implementation of HHK relies on the following two key points:

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

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

As a consequence, HHK has the serious drawback of a quadratic space complexity. In fact, HHK needs $O(|\Sigma|^2 \log|\Sigma|)$-space because the representation of the matrix Count takes exactly $|\Sigma|^2 \log|\Sigma|$ space and both $\langle \mathit{Sim}(s)\rangle_{s\in\Sigma}$ and $\langle \mathit{Remove}(s)\rangle_{s\in\Sigma}$ need $|\Sigma|^2$ space in the worst case.

```
RT(PartitionRelation $\langle P, \mathit{Rel}\rangle$){
  forall $B \in P$ do $\mathit{Remove}(B) := \Sigma \smallsetminus \mathrm{pre}(\cup \mathit{Rel}(B))$;
  initializeCount();
  while $\exists B \in P$ such that $(\mathit{Remove}(B) \neq \varnothing)$ do
    $\mathit{Remove} := \mathit{Remove}(B)$;
    $\mathit{Remove}(B) := \varnothing$;
    $B_{\mathrm{prev}} := B$;
    $P_{\mathrm{prev}} := P$;
    $P := \mathit{Split}(P, \mathit{Remove})$;
    $\mathit{newBlocks}_P := P \smallsetminus P_{\mathrm{prev}}$;
    if $\mathit{newBlocks}_P \neq \varnothing$ then
      updateRel();
      updateCount();
      updateRemove();
    $\mathit{RemoveList} := \{D \in P \mid D \subseteq \mathit{Remove}\}$;
    forall $C \in P$ such that $C \cap \mathrm{pre}(B_{\mathrm{prev}}) \neq \varnothing$ do
      forall $D \in \mathit{RemoveList}$ do
        if $\mathit{Rel}(C,D)$ then
          $\mathit{Rel}(C,D) := \mathit{ff}$;
          forall $s \in \mathrm{pre}(D)$ do
            $\mathit{Count}(s,C)$−−;
            if $\mathit{Count}(s,C) = 0$ then
              $\mathit{Remove}(C) :=$
                    $\mathit{Remove}(C) \cup \{s\}$;
}
```

Figure 2. RT Simulation Algorithm.

Algorithm RT**.** The algorithm RT [14] is recalled in Figure 2. It is obtained as a modification of HHK that exploits the idea of representing an over-approximation of the simulation preorder R_{sim} through a so-called partition-relation pair $\langle P, \mathit{Rel}\rangle$, where $P \in \mathrm{Part}(\Sigma)$ is a partition of Σ and $\mathit{Rel} \subseteq P \times P$ is a partial order (i.e., reflexive, transitive and antisymmetric) relation on P. In fact, preorders and partition-relation pairs can be related through a bijection:

(1) A preorder $\mathit{Sim} \subseteq \Sigma \times \Sigma$ induces the following partition-relation pair $\langle P, \mathit{Rel}\rangle$: $s_1 \sim_P s_2$ if $\forall s \in \Sigma.\ s_1 \in \mathit{Sim}(s) \Leftrightarrow s_2 \in \mathit{Sim}(s)$, and $(B_1, B_2) \in \mathit{Rel}$ if $B_2 \subseteq \mathit{Sim}(B_1)$.

(2) Conversely, a partition-relation pair $\langle P, \mathit{Rel}\rangle$ induces the following preorder $\mathit{Sim} \subseteq \Sigma \times \Sigma$: $(x,y) \in \mathit{Sim}$ if $(P(x), P(y)) \in \mathit{Rel}$.

Moreover, these two correspondences are one the inverse of the other. Thus, a family of sets $\langle \mathit{Sim}(s)\rangle_{s\in\Sigma}$ that approximates from above the simulation preorder R_{sim} is represented in RT by the corresponding partition-relation pair $\langle P, \mathit{Rel}\rangle$. More precisely, the partition P is maintained as an over-approximation of (i.e., coarser than) simulation equivalence P_{sim}, namely $P_{\mathrm{sim}} \preceq P$, so that if s and t are not in the same block of P then they are not simulation equivalent. On the other hand, the partial order relation Rel is maintained as an over-approximation of the simulation preorder, meaning that if a pair (B_1, B_2) of blocks of P

does not belong to the relation Rel, then no state $s \in B_1$ can be simulated by a state $t \in B_2$. The simulation problem can then be rephrased as the problem of finding the coarsest partition-relation pair representing the simulation preorder, that is a pair $\langle P, \mathit{Rel}\rangle$ such that (1) $P \preceq P_\ell$, and (2) $\forall B_1, B_2 \in P.\ B_1 \rightarrow_\exists B_2 \Rightarrow \cup \mathit{Rel}(B_1) \rightarrow_\forall \cup \mathit{Rel}(B_2)$.

The input partition-relation pair for RT is determined by the state labeling ℓ: P_ℓ is the input partition while the identity $\mathit{Rel}_\ell \overset{\text{def}}{=} \{(B, B) \mid B \in P_\ell\}$ is the input relation. RT iteratively refines the current partition-relation pair $\langle P, \mathit{Rel}\rangle$ until a fixpoint is reached, namely when $\langle P, \mathit{Rel}\rangle$ represents a simulation relation.

In RT, the representation based on partition-relation pairs allows to save both space and time w.r.t. HHK. Intuitively, this depends on the fact that refining a partition and a relation between *blocks* can be more efficient than refining a relation between *states*. Consider for instance the example in Figure 1: the states $1, 2$ and 3 have the same label as well as the same set of simulators, hence they may correctly (and efficiently) be compacted into a single (abstract) state. More precisely, storing the partition-relation $\langle P, \mathit{Rel}\rangle$ rather that the relation $\langle \mathit{Sim}(s)\rangle_{s\in\Sigma}$ on the whole state space, requires $|P_{\mathrm{sim}}|^2$ space instead of $|\Sigma|^2$. In particular, Rel is stored as a resizable boolean matrix so that insert operations take amortized constant time.

Relying on partition-relation pairs, partition blocks play the role of (abstract) states, therefore, similarly to HHK, RT maintains for any block $B \in P$ a set of remove states $\mathit{Remove}(B) \subseteq \Sigma$ and an integer table $\mathit{Count}(s, B)$, indexed on states $s \in \Sigma$ and blocks $B \in P$, where $\mathit{Count}(s, B)$ stores the number of transitions from s to some state in $\cup \mathit{Rel}(B)$, i.e., $\mathit{Count}(s, B) = \sum_{C\in P,(B,C)\in \mathit{Rel}} |\operatorname{post}(s) \cap C|$. The space needed to store these two structures is in the worst case $O(|P_{\mathrm{sim}}||\Sigma|)$ and $O(|\Sigma||P_{\mathrm{sim}}| \log |\Sigma|)$, respectively. Finally, each state must have a pointer to the block of the partition 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 for time complexity, the $O(|P_{\mathrm{sim}}||{\rightarrow}|)$ time bound of RT crucially depends on the following properties.

- The Count table allows to perform the test $\mathit{Count}(s, C) = 0$ at line 22 in constant time. This test logically corresponds to check whether $s \notin \operatorname{pre}(\cup \mathit{Rel}(C))$.
- 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 $\mathit{Rel}(B_i) = \mathit{Rel}(B)$, $\mathit{Remove}(B_i) = \mathit{Remove}(B)$ and $\mathit{Count}(s, B_i) = \mathit{Count}(s, B)$.
- 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 $\mathit{Remove}(B_i) \cap \mathit{Remove}(B_j) = \varnothing$. As a consequence of this property:
  - ◇ A split operation $\mathit{Split}(P, \mathit{Remove})$ at line 9 can be done in $O(|\,\mathit{Remove}\,|)$-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 $\mathit{Remove}(B)$ then for any block $C \subseteq B$ which is selected at some successive iteration it turns out that any block E contained in D cannot be selected at line 17 as a subset of $\mathit{Remove}(C)$. Using this key property, one can prove that the overall time complexity of the for-loop at lines 16-22 is in $O(|P_{\mathrm{sim}}||{\rightarrow}|)$.

3. A New Simulation Algorithm

As discussed above, RT can be viewed 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 explicit: (1) in the family $\{\mathit{Remove}(B)\}_{B\in P}$ any $\mathit{Remove}(B) \subseteq \Sigma$ is represented as a set of explicit states; (2) the rows of the integer table $\{\mathit{Count}(s, B)\}_{s\in\Sigma, B\in P}$ are explicitly indexed on the whole space state 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 structures Remove and Count in terms of blocks of Q. This allows us to save space in the time efficient algorithm 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 in Figure 3.

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 $\mathit{Remove}(B)$ can be represented as 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.

The algorithm RT initializes and then modifies a remove set at lines 2, 6 and 22, whereas the `updateRemove()` function at line 14 does not affect the remove sets. At line 2, $\mathit{Remove}(B)$ is initialized as $\Sigma \smallsetminus \operatorname{pre}(\cup \mathit{Rel}(B))$. Then, after selecting a block $B \in P$, RT empties the corresponding $\mathit{Remove}(B)$ at line 6. Finally, a remove set $\mathit{Remove}(C)$ can be modified at line 22, where the states that are added to $\mathit{Remove}(C)$ are those in the set $\operatorname{pre}(D) \smallsetminus \operatorname{pre}(\cup \mathit{Rel}(C))$ for some $D \in P$. Recall that any set $\cup \mathit{Rel}(B) \subseteq \Sigma$ is a union of blocks of P and that the predecessor operator pre is additive. Thus, in order to represent the remove sets as 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. We 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 [15]. 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 $\mathrm{pre}(B)$ where $B \in P$. 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 14. 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. This is done by the for-loop at lines 17-19.

Hence, this allows us to represent any remove set $Remove(B)$ as a set of blocks of the partition Q. At the end 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}. Therefore, our representation of the remove sets takes $O(|P_{\mathrm{sim}}||P_{\mathrm{sp}}|)$ space in the worst case. Since P_{bis} strongly progresses to P_{sim}, we have that $P_{\mathrm{bis}} \preceq P_{\mathrm{sp}}$ so that $P_{\mathrm{bis}} \preceq P_{\mathrm{sp}} \preceq P_{\mathrm{sim}}$. 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 4) that $|P_{\mathrm{sp}}|$ tends to be much closer to $|P_{\mathrm{sim}}|$ rather than to $|P_{\mathrm{bis}}|$, so that the space complexity $O(|P_{\mathrm{sim}}||P_{\mathrm{sp}}|)$ is comparable to $O(|P_{\mathrm{sim}}|^2)$.

```
1  SA(PartitionRelation $\langle P, Rel \rangle$){
2    $Q := P$;
3    forall $C \in P$ do $Q := Split(Q, \mathrm{pre}(C))$;
4    computePreEE($P$);
5    computeCount();
6    forall $B \in P$ do
7      forall $E \in Q$ do
8        if $Count(E, B) = 0$ then
           $Remove(B).append(E)$;
9    while $\exists B \in P$ such that $(Remove(B) \neq \varnothing)$ do
10     $Remove := Remove(B)$;
11     $Remove(B) := \varnothing$;
12     $B_{\mathrm{prev}} := B$;
13     $P_{\mathrm{prev}} := P$;
14     $P := Split(P, \cup Remove)$;
15     $newBlocks_P := P \smallsetminus P_{\mathrm{prev}}$;
16     $Q_{\mathrm{prev}} := Q$;
17     forall $C \in newBlocks_P$ do
18       $Remove(C) := Remove(\mathrm{parent}_{P_{\mathrm{prev}}}(C))$;
19       $Q := Split(Q, \mathrm{pre}(C))$;
20     $newBlocks_Q := Q \smallsetminus Q_{\mathrm{prev}}$;
21     if $newBlocks_P \neq \varnothing$ then
22       updateRel();
23       if $newBlocks_Q \neq \varnothing$ then
24         computePreEE($P$);
25         updateRemove();
26       else
27         computePreEE($newBlocks_P$);
28       computeCount();
29     $RemoveList := \{D \in P \mid D \subseteq Remove\}$;
30     forall $C \in P$ such that $C \cap pre(B_{\mathrm{prev}}) \neq \varnothing$ do
31       forall $D \in RemoveList$ do
32         if $Rel(C, D)$ then
33           $Rel(C, D) := f\!f$;
34           forall $E \in preEE(D)$ do
35             $Count(E, C)$--;
36             if $Count(E, C) = 0$ then
37               $Remove(C).append(E)$;
38 }
```

Figure 3. A New Simulation Algorithm.

Exploiting abstract transitions. Blocks of the additional partition Q play the role of symbolic states in SA. Hence, our second basic idea consists in replacing the semi-symbolic integer table $\{Count(s, B)\}_{s \in \Sigma, B \in P}$, which takes $O(|\Sigma||P_{\mathrm{sim}}|\log|\Sigma|)$ space, with a fully symbolic integer table $\{Count(E, B)\}_{E \in Q, B \in P}$, where rows are indexed over symbolic states in Q, which takes $O(|P_{\mathrm{sp}}||P_{\mathrm{sim}}|\log|P_{\mathrm{sp}}|)$ space. Let us recall that $Count(s, C)$ stores the number of transitions from the state s to some state in $\cup Rel(C)$ and that this information is used crucially in RT at line 22 to check whether $s \notin \mathrm{pre}(\cup Rel(C))$ in constant time so that the states in $\mathrm{pre}(D) \smallsetminus \mathrm{pre}(\cup Rel(C))$ are added to $Remove(C)$. Notice that in SA, thanks to the representation of remove sets in terms of blocks of Q, if some state $s \in \mathrm{pre}(D) \smallsetminus \mathrm{pre}(\cup Rel(C))$ must be added to $Remove(C)$, then the whole block $Q(s)$ of Q that contains s can be added to $Remove(C)$. Therefore, instead of adding single states to $Remove(C)$ we add 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 $E \cap \mathrm{pre}(\cup\, Rel(C)) = \varnothing$. Hence, what we need to store in the symbolic table $Count$ is some information that allows us to check whether $E \cap \mathrm{pre}(\cup\, Rel(C)) = \varnothing$ in constant time. To this purpose, it is enough to store in $Count(E, C)$ the number of abstract existential transitions from the block $E \in Q$ to some block $B \in Rel(C)$: in fact, with this representation, we have that $Count(E, C) = 0$ iff $E \cap \mathrm{pre}(\cup Rel(C)) = \varnothing$. More precisely, for any block $E \in Q$ and $C \in P$,

$$Count(E, C) \stackrel{\mathrm{def}}{=} |\{B \in P \mid B \in Rel(C),\ E \to_{\exists} B\}|.$$

The advantage of this choice lies in the time complexity of computing the table $Count$. The table $Count$ has to be updated whenever either P or Q are refined. Unfortunately, $Count$ has to be recomputed from scratch because the previous information stored in $Count$ cannot be easily updated. However, this recomputation of $Count$ can be done by traversing the abstract existential transition relation $\rightarrow_\exists \subseteq Q \times P$ between Q and P instead of the concrete transition relation $\rightarrow \subseteq \Sigma \times \Sigma$ and this is the key point that allows us to save time w.r.t. the GPP-GP algorithm. Thus, we also store and maintain the abstract transition relation $\rightarrow_\exists$ as predecessor transformer: for any block $B \in P$, $preEE(B)$ is a list containing all the blocks E in Q such that $E \rightarrow_\exists B$. This data structure takes $O(|\rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}|)$ space in the worst case (notice that $|\rightarrow^{P_{\mathrm{sp}},P_{\mathrm{sim}}}| \leq |\rightarrow|$ always holds).

The functions `computePreEE` and `computeCount` that compute, respectively, the data structures $preEE$ and $Count$ are 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 $B \in L$. A call to the function `computeCount` resizes and initializes $Count$ as a $|Q| \times |P|$ integer table of 0 and then computes the information by traversing the existential predecessors stored in $preEE$.

The `updateRel` function in Figure 4 updates the resizable boolean matrix Rel and works exactly as in RT. It is called at line 22 each time the partition P is refined. The `updateRemove` function in Figure 4 is called at line 25 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 the splitting operation. Notice at lines 17-19 that the partition Q can be refined only when P is refined.

The algorithm at work on an example. We illustrate the execution of SA showing how it computes the simulation preorder for the Kripke structure given in Figure 1. The input is the partition-relation $\langle P_\ell, Rel_\ell \rangle$, where the initial partition corresponds to the state labeling, i.e., $P_\ell = \{B_1, B_2, B_3\}$ with $B_1 = \{1,2,3\}, B_2 = \{4,5\}, B_3 = \{6,7\}$, and the initial relation Rel_ℓ is the identity function over the blocks of P_ℓ.

The initialization phase computes the partition Q by refining P_ℓ w.r.t. the predecessor sets of every block of P_ℓ. The resulting partition is $Q = \{E_1, E_2, E_3, E_4\}$ with $E_1 = \{1,2,3\}, E_2 = \{4,5\}, E_3 = \{6\}, E_4 = \{7\}$, which is strictly finer than P_ℓ. Now, 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 results in the initialization of the following remove sets, that will be emptied during the iterations of the main loop:

$$\begin{aligned} Remove(B_1) &= \{E_1, E_2, E_3, E_4\} \\ Remove(B_2) &= \{E_2, E_4\} \\ Remove(B_3) &= \{E_1\} \end{aligned}$$

```
computePreEE(ListOfBlocks $L$){
  forall $B \in L$ do $preEE(B) := \varnothing$;
  forall $B \in L$ do
    forall $y \in B$ do
      forall $x \in \mathrm{pre}(\{y\})$ do
        if ($Q(x)$ unmarked) then
          $preEE(B).append(Q(x))$;
          mark($Q(x)$);
    forall $E \in preEE(B)$ do unmark($E$);
}

computeCount(){
  Resize the table Count;
  forall $E \in Q$ do
    forall $B \in P$ do $Count(E,B) := 0$;
  forall $C \in P$ do
    forall $E \in preEE(C)$ do
      forall $B \in P$ such that $Rel(B,C)$ do
        $Count(E,B)$++;
}

updateRel(){
  Resize the matrix Rel;
  forall $B \in newBlocks_P$ do
    forall $C \in P$ do
      $Rel(B,C) :=$
          $Rel(\mathrm{parent}_{P_{\mathrm{prev}}}(B), \mathrm{parent}_{P_{\mathrm{prev}}}(C))$;
  forall $C \in newBlocks_P$ do
    forall $B \in P \smallsetminus newBlocks_P$ do
      $Rel(B,C) :=$
          $Rel(\mathrm{parent}_{P_{\mathrm{prev}}}(B), \mathrm{parent}_{P_{\mathrm{prev}}}(C))$;
}

updateRemove(){
  forall $B \in P$ do
    forall $E \in Remove(B)$ do
      replace $E$ with $\{F \in newBlocks_Q \mid F \subseteq E\}$;
}
```

Figure 4. Auxiliary functions.

The first iteration chooses B_1 as pivot, hence the set $Remove(B_1)$ is cleared out and the initial partition is split w.r.t. the set of states corresponding to $\bigcup_{i=1,..,4} E_i$, which does not modify the partition P_ℓ. Moreover, since there is no block $C \in P$ such that $C \cap pre(B_1) \neq \varnothing$, also the relation Rel_ℓ is not modified. The second iteration chooses B_2 as pivot; it clears out the set $Remove(B_2)$ and splits P_ℓ w.r.t. the set $\{4,5,7\} = E_2 \cup E_4$. The resulting partition is $P' = \{B_1, B_2, B_3', B_3''\}$, where the block B_3 splits into $B_3' = \{7\}$ and $B_3'' = \{6\}$. The refinement of the partition P_ℓ induces the refinement of Q into $Q' = \{E_1, E_2', E_2'', E_3, E_4\}$ with $E_2' = \{4\}$, $E_2'' = \{5\}$, and the rearrangement of the

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

Table 1. Space and time complexities.

relation and the remove sets, which are updated as follows:

$$\begin{array}{lclllcl}
\mathit{Rel}(B_1) & = & \{B_1\} & & \mathit{Remove}(B_1) & = & \emptyset\\
\mathit{Rel}(B_2) & = & \{B_2\} & & \mathit{Remove}(B_2) & = & \emptyset\\
\mathit{Rel}(B_3') & = & \{B_3',B_3''\} & & \mathit{Remove}(B_3') & = & \{E_1\}\\
\mathit{Rel}(B_3'') & = & \{B_3',B_3''\} & & \mathit{Remove}(B_3'') & = & \{E_1\}
\end{array}$$

Using B_2 as pivot, the lines 30-37 of the second iteration remove the blocks $\{D\in P' | D\subseteq\{4,5,7\}\}$ from the set of simulators of every $C\in P'$ such that $C\cap \mathit{pre}(B_2)\neq\varnothing$. In particular, this step modifies the simulators of B_3'' as well as its remove set, which become

$$\mathit{Rel}(B_3'') = \{B_3''\} \qquad \mathit{Remove}(B_3'') = \{E_1,E_3,E_4\}$$

The third iteration clears out the set $\mathit{Remove}(B_3')$ and leaves untouched both the partitions and the relation. A final iteration clears out the last set $\mathit{Remove}(B_3'')$ without any modification of the partitions and the relation. The final output is then the following:

$$P_{\rm sim} = \{\{1,2,3\},\{4,5\},\{6\},\{7\}\}$$

$$P_{\rm sp} = \{\{1,2,3\},\{4\},\{5\},\{6\},\{7\}\}$$

$$\mathit{Rel}(\{1,2,3\}) = \{1,2,3\} \quad \mathit{Rel}(\{4,5\}) = \{4,5\}$$

$$\mathit{Rel}(\{6\}) = \{6\} \quad \mathit{Rel}(\{7\}) = \{6,7\}$$

First, notice that in this case $P_{\rm sp}\prec P_{\rm sim}$, and $P_{\rm sp}$ is strictly coarser than $P_{\rm bis}$ that in this example is the identity equivalence over states. Moreover, this example shows the space saving of SA compared to both RT and HHK. Compared to the concrete algorithm HHK, in SA (as in RT) the sets of equivalent states as $\{1,2,3\}$ and $\{4,5\}$ are represented as single blocks, saving the cost of maintaining replicated information. On the other hand, SA obtains a similar space saving over RT by compactly representing the remove sets using blocks (of Q) instead of sets of equivalent states, namely the blocks E_1 and E_2 rather than $\{1,2,3\}$ and $\{4,5\}$.

Correctness and Complexity. The correctness of the SA algorithm is a consequence of the correctness of the RT algorithm and the properties described above. Analogously to RT, the input partition-relation pair of SA is determined by the labeling ℓ of the Kripke structure, namely the input partition-relation pair is $\langle P_\ell, \mathit{Rel}_\ell\rangle$.

Theorem 3.1 (Correctness). *The algorithm* SA *always terminates on a finite Kripke structure $\mathcal{K}$ and outputs a partition-relation pair that induces the simulation preorder $R_{\rm sim}$ on $\mathcal{K}$.*

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

Theorem 3.2 (Complexity). *The algorithm* SA *runs in $O(|P_{\rm sim}||{\to}|+|P_{\rm sim}|^2|{\to}^{P_{\rm sp},P_{\rm sim}}|)$-time and $O((|P_{\rm sp}||P_{\rm sim}|+|\Sigma|)\log|P_{\rm sp}|)$-space.*

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

- The state partitions Q and P that take, resp., $O(|P_{\rm sp}|)$ and $O(|P_{\rm sim}|)$.
- 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, resp., $O(|\Sigma|\log|P_{\rm sp}|)$ and $O(|P_{\rm sp}|\log|P_{\rm sim}|)$.
- The resizable boolean matrix $\{\mathit{Rel}(B,C)\}_{B,C\in P}$ that takes $O(|P_{\rm sim}|^2)$.
- The remove lists $\{\mathit{Remove}(B)\}_{B\in P}$ that takes $O(|P_{\rm sim}||P_{\rm sp}|\log|P_{\rm sp}|)$.
- The integer table $\{\mathit{Count}(E,B)\}_{E\in Q,B\in P}$ that takes $O(|P_{\rm sp}||P_{\rm sim}|\log|P_{\rm sp}|)$.
- The lists of existential predecessors $\{\mathit{preEE}(B)\}_{B\in P}$ that takes $O(|P_{\rm sim}||P_{\rm sp}|\log|P_{\rm sp}|)$.

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. Let us recall that a splitting operation $\mathit{Split}(P,S)$ can be implemented by scanning all the states in S and therefore it takes $O(|S|)$ time. Also, since the transition relation is supposed to be total, we have that $|Q|\leq|{\to}^{Q,P}|$.

- The initializations of Q (line 3), preEE (line 4), Count (line 5) and Remove (lines 6-8) take time, resp., $O(\sum_{B\in P}|\mathrm{pre}(B)|)=O(|{\to}|)$, $O(|{\to}|)$, $O(|P||Q|+|P||{\to}^{Q,P}|)$ and $O(|P||Q|)$. Thus, the initialization phase takes $O(|{\to}|+|P||{\to}^{Q,P}|)$ time, i.e. $O(|{\to}|+|P_{\rm sim}||{\to}^{P_{\rm sp},P_{\rm sim}}|)$.
- As shown in [14] for RT, the overall time complexity of splitting the partition P and updating the resizable matrix Rel (lines 14 and 22) is $O(|P_{\rm sim}||{\to}|)$.

- As shown in [14] for RT, the overall number of newly generated blocks by the splitting operation at line 14 is $2(|P_{\mathrm{sim}}| - |P_{\ell}|)$, namely it is in $O(|P_{\mathrm{sim}}|)$. Moreover, each splitting of the partition Q takes $O(|\operatorname{pre}(C)|)$ 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 is in $O(|P_{\mathrm{sim}}||{\to}|)$-time.
- Each call to `computePreEE` takes $O(|{\to}|)$ time. This function is called at lines 24 and 27 at most $O(|P_{\mathrm{sim}}|)$ time so that the overall time complexity of lines 24 and 27 is in $O(|P_{\mathrm{sim}}||{\to}|)$.
- Each call to `computeCount` takes $O(|P||Q| + |P||{\to}^{Q,P}|)$ time. This function is called at line 28 at most $O(|P_{\mathrm{sim}}|)$ time so that the overall time complexity of line 28 is in $O(|P_{\mathrm{sim}}|^2|P_{\mathrm{sp}}| + |P_{\mathrm{sim}}|^2|{\to}^{P_{\mathrm{sp}},P_{\mathrm{sim}}}|)$.
- Each call to `updateRemove` takes $O(|P||Q|)$ time. This function is called at line 25 at most $O(|P_{\mathrm{sim}}|)$ time so that the overall time complexity of line 25 is in $O(|P_{\mathrm{sim}}|^2|P_{\mathrm{sp}}|)$.
- As shown in [14] for RT, the overall time complexity of lines 30-31 is in $O(|P_{\mathrm{sim}}||{\to}|)$.
- Following the same pattern of proof in [14], one can show that the overall time complexity of lines 32-37 is in $O(|P_{\mathrm{sim}}||{\to}^{P_{\mathrm{sp}},P_{\mathrm{sim}}}|)$. The difference between lines 18-22 in RT and lines 32-37 in SA lies in the fact that in RT at line 20 by means of $s \in \operatorname{pre}(D)$ we traverse the concrete transition relation $\to$ while in SA at line 34 by means of $E \in \mathit{preEE}(D)$ we traverse instead the abstract existential transition relation $\to^{Q,P}$.

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.

Adapting the algorithm to LTSs. The algorithms discussed so far compute the simulation relation on Kripke structures, but they can be easily adapted to work over LTSs. The LTS version of the algorithm RT is given in [1]; the key point is that for each block B, instead of a single set $\mathit{Remove}(B)$, the algorithm maintains a family of sets $\mathit{Remove}_a(B)$ indexed over the labels $a \in \mathcal{L}$ of the LTS. Intuitively, the set $\mathit{Remove}_a(B)$ collects the states that do not have an a-transition going into states that are in B nor to states of any block B' such that $(B, B') \in \mathit{Rel}$. Similarly, the Count table is generalized to the table $\{\mathit{Count}_a(s, B)\}_{s\in\Sigma, B\in P, a\in\mathcal{L}}$ whose additional third dimension takes into account the label of the transitions from s to some state in $\cup\,\mathit{Rel}(B)$. The complexity of this algorithm is equal to that of RT, up to a multiplicative factor corresponding to the cardinality of the set of labels $\mathcal{L}$.

Combining the approach of [1] with that described in this section, also the algorithm SA can be adapted to LTSs. The only additional issue is that, for any $B \in P$ and $a \in \mathcal{L}$, any set $\mathit{Remove}_a(B)$ must still be represented as a set of blocks of the partition Q. In order to satisfy this invariant it is sufficient to take a partition Q such that any a-predecessor set $\operatorname{pre}_a(C)$, for some label $a \in \mathcal{L}$ and some block $C \in P$, is a union of blocks of Q. More precisely, it is sufficient to iterate the split operations in lines 3 and 19 for any a-predecessor set $\operatorname{pre}_a(C)$ where a ranges over the set $\mathcal{L}$ of labels.

Alternatively, in order to compute the simulation relation over the states of an LTS M, it is sufficient to transform M into a Kripke structure M' in such a way that simulation equivalences on M and M' coincide. This transformation (by Dovier et al. [8]) acts as follows: any labeled transition $s_1 \xrightarrow{a} s_2$ is replaced by two unlabeled transitions $s_1 \to n$ and $n \to s_2$, where n is a new state that is labeled with a, while all the original states in M have the same label. This labeling provides an initial partition on M', that can be used as input of the original SA algorithm. This approach is used in the experimental evaluation of SA discussed below.

4. Experimental Evaluation and Discussion

A prototype of the SA algorithm has been developed in C++ and has been evaluated on a number of concurrent benchmarks. Our benchmarks include systems taken from publicly available examples in the VLTS (Very Large Transition Systems) benchmark suite [18], CWB-NC (The Concurrency WorkBench of New Century) [6] and mCRL2 (micro Common Representation Language 2) [11]. In particular, the benchmarks taken from CWB-NC have been also used in [9] for evaluating the GPP-GP algorithm. These models are represented as labelled transition systems, hence we exploit the procedure described above to transforms a LTS M into a Kripke structure M', together with an initial partition P_{init}, that preserves simulation equivalence. Notice that this transformation grows the size of the model 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 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 $\to^{P_{\mathrm{sp}},P_{\mathrm{sim}}}$ 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, resp., 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 $\to^{P_{\mathrm{sp}},P_{\mathrm{sim}}}$ w.r.t. the concrete transition relation $\to$. We may therefore observe that (1) the average reduction by P_{sp} of the concrete state space is 66.74%; (2) the average decrease of P_{sp} w.r.t. P_{sim} is 0.66%, namely the size of P_{sp} tends to be close to

	Input			Output				Experim. Results	
Model	$\lvert\Sigma\rvert$	$\lvert\to\rvert$	$\lvert P_{\text{init}}\rvert$	$\lvert P_{\text{sim}}\rvert$	$\lvert P_{\text{sp}}\rvert$	$\lvert P_{\text{bis}}\rvert$	$\lvert\to^{P_{\text{sp}},P_{\text{sim}}}\rvert$	Time(s)	Space(MB)
ABP-lossy [6]	187	260	4	32	32	32	48	0.003	0.03
ABP-lossy-2 [6]	1821	3280	12	1004	1004	1004	2456	4.264	7.40
ABP-safe [6]	123	148	4	40	40	40	53	0.003	0.03
two-link-netw [6]	8408	13638	4	336	375	448	883	0.670	1.38
cwi_1_2 [18]	4339	4774	27	2401	2401	2401	2701	103.89	3.6
cwi_3_14 [18]	18548	29104	3	123	123	123	122	1.045	1.67
vasy_0_1 [18]	1513	2448	3	21	21	21	32	0.013	0.13
vasy_1_4 [18]	5647	8928	7	87	87	87	118	0.074	0.43
brp [11]	22716	24336	5	591	591	591	648	2.80	3.18
leader [11]	1520	2256	3	47	47	47	46	0.031	0.14
mpsu [11]	202	300	15	145	145	145	229	0.021	0.19
par [11]	209	236	6	58	58	58	67	0.007	0.05
parallel [11]	8000	14000	286	1540	1540	1540	2640	16.158	19.86
tree [11]	2049	2048	3	43	43	43	59	0.014	0.15
cabp [11]	2096	3264	6	210	213	216	411	0.098	0.56
scheduler [11]	32	38	6	30	30	30	36	0.001	0.01
dining-phil-4 [11]	418	600	21	418	418	418	600	0.338	1.64
dining-phil-5 [11]	1642	2500	26	1642	1642	1642	2500	24.463	23.02

Table 2. Results of the experimental evaluation.

	Comparison		
Model	% $\lvert P_{\text{sp}}\rvert/\lvert\Sigma\rvert$	% $\lvert P_{\text{sim}}\rvert/\lvert P_{\text{sp}}\rvert$	% $\lvert\to^{P_{\text{sp}},P_{\text{sim}}}\rvert/\lvert\to\rvert$
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
cwi_1_2 [18]	55.34	100.00	56.58
cwi_3_14 [18]	0.66	100.00	0.42
vasy_0_1 [18]	1.39	100.00	1.31
vasy_1_4 [18]	1.54	100.00	1.32
brp [11]	2.60	100.00	2.66
leader [11]	3.09	100.00	2.04
mpsu [11]	71.78	100.00	76.33
par [11]	27.75	100.00	28.39
parallel [11]	19.25	100.00	18.86
tree [11]	2.10	100.00	2.88
cabp [11]	10.16	98.59	12.59
scheduler [11]	93.75	100.00	94.74
dining-phil-4 [11]	100.00	100.00	100.00
dining-phil-5 [11]	100.00	100.00	100.00
Average	33.26	99.34	35.21

Table 3. Experimental comparison.

that of simulation equivalence P_{sim}; (3) the average decrease of the number of arcs of $\to^{P_{\text{sp}},P_{\text{sim}}}$ w.r.t. $\to$ is 64.79%, i.e. $|\to^{P_{\text{sp}},P_{\text{sim}}}| \approx |\to|/3$.

This suggests us 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 $\to^{P_{\text{sp}},P_{\text{sim}}}$ is smaller than that of the concrete transition relation $\to$. However, 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 the size of $|P_{\text{sp}}| \approx |P_{\text{sim}}|$. 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.

Acknowledgements. Work partially supported by the PRIN 2007 Project "*AIDA2007: Abstract Interpretation Design and Applications*" and by the University of Padova under the Projects "*Formal methods for specifying and verifying behavioural properties of software systems*" and "*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*, ch. 6, pages 391-424, 2001.

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

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

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

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

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

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

[14] F. Ranzato and F. Tapparo. A new efficient simulation equivalence algorithm. In *Proc. 22nd IEEE Symp. on Logic in Computer Science (LICS'07)*, pp. 171–180, IEEE Press, 2007. Extended version available at arXiv as http://arxiv.org/abs/0709.4118.

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

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

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

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