A More Efficient Simulation Algorithm on Kripke Structures

Francesco Ranzato

Dipartimento di Matematica, University of Padova, Italy

Abstract. A number of algorithms for computing the simulation preorder (and equivalence) on Kripke structures are available. Let Σ denote the state space, $\rightarrow$ the transition relation and P_{sim} the partition of Σ induced by simulation equivalence. While some algorithms are designed to reach the best space bounds, whose dominating additive term is $|P_{\mathrm{sim}}|^2$, other algorithms are devised to attain the best time complexity $O(|P_{\mathrm{sim}}||{\rightarrow}|)$. We present a novel simulation algorithm which is both space and time efficient: it runs in $O(|P_{\mathrm{sim}}|^2 \log |P_{\mathrm{sim}}| + |\Sigma| \log |\Sigma|)$ space and $O(|P_{\mathrm{sim}}||{\rightarrow}| \log |\Sigma|)$ time. Our simulation algorithm thus reaches the best space bounds while closely approaching the best time complexity.

1 Introduction

The simulation preorder is a fundamental behavioral relation widely used in process algebra for establishing system correctness and in model checking as a suitable abstraction for reducing the size of state spaces. The problem of efficiently computing the simulation preorder (and consequently simulation equivalence) on finite Kripke structures has been thoroughly investigated and generated a number of simulation algorithms. Both time and space complexities play an important role in simulation algorithms, since in several applications, especially in model checking, memory requirements may become a serious bottleneck as the input transition system grows.

Consider a finite Kripke structure where Σ denotes the state space, $\rightarrow$ the transition relation and P_{sim} the partition of Σ induced by simulation equivalence. The best simulation algorithms are those by, in chronological order, Gentilini, Piazza and Policriti (GPP) [3] (subsequently corrected in [4]), Ranzato and Tapparo (RT) [10,12], Markovski (Mar) [8], Cécé (Space-Céc and Time-Céc) [2]. The simulation algorithms GPP and RT are designed for Kripke structures, while Space-Céc, Time-Céc and Mar are for more general labeled transition systems. Their space and time complexities are summarized in the following table.

Algorithm	Space complexity	Time complexity
Space-Céc [2]	$O(\|P_{\mathrm{sim}}\|^2 + \|{\rightarrow}\| \log \|{\rightarrow}\|)$	$O(\|P_{\mathrm{sim}}\|^2 \|{\rightarrow}\|)$
Time-Céc [2]	$O(\|P_{\mathrm{sim}}\|\|\Sigma\| \log \|\Sigma\| + \|{\rightarrow}\| \log \|{\rightarrow}\|)$	$O(\|P_{\mathrm{sim}}\|\|{\rightarrow}\|)$
GPP [3]	$O(\|P_{\mathrm{sim}}\|^2 \log \|P_{\mathrm{sim}}\| + \|\Sigma\| \log \|\Sigma\|)$	$O(\|P_{\mathrm{sim}}\|^2 \|{\rightarrow}\|)$
Mar [8]	$O((\|\Sigma\| + \|P_{\mathrm{sim}}\|^2) \log \|P_{\mathrm{sim}}\|)$	$O(\|{\rightarrow}\| + \|P_{\mathrm{sim}}\|\|\Sigma\| + \|P_{\mathrm{sim}}\|^3)$
RT [12]	$O(\|P_{\mathrm{sim}}\|\|\Sigma\| \log \|\Sigma\|)$	$O(\|P_{\mathrm{sim}}\|\|{\rightarrow}\|)$
ESim (this paper)	$O(\|P_{\mathrm{sim}}\|^2 \log \|P_{\mathrm{sim}}\| + \|\Sigma\| \log \|\Sigma\|)$	$O(\|P_{\mathrm{sim}}\|\|{\rightarrow}\| \log \|\Sigma\|)$

We remark that all the above space bounds are bit space complexities, i.e., the word size is a single bit. Let us also remark that both articles [3,4] state that the bit space complexity of GPP is in $O(|P_{\mathrm{sim}}|^2 + |\Sigma| \log |P_{\mathrm{sim}}|)$. However, as observed also in [2], this is not precise. In fact, the algorithm GPP [3, Section 4, p. 98] assumes that the states belonging to some block are stored as a doubly linked list, and this entails a bit space complexity in $O(|\Sigma| \log |\Sigma|)$. Furthermore, GPP uses Henzinger, Henzinger and Kopke [5] simulation algorithm (HKK) as a subroutine, whose bit space complexity is in $O(|\Sigma|^2 \log |\Sigma|)$, which is called on a Kripke structure where states are blocks of the current partition. The bit space complexity of GPP must therefore include an additive term $|P_{\mathrm{sim}}|^2 \log |P_{\mathrm{sim}}|$ and therefore results to be $O(|P_{\mathrm{sim}}|^2 \log |P_{\mathrm{sim}}| + |\Sigma| \log |\Sigma|)$. It is worth observing that a space complexity in $O(|P_{\mathrm{sim}}|^2 + |\Sigma| \log |P_{\mathrm{sim}}|)$ can be considered optimal for a simulation algorithm, since this is of the same order as the size of the output, which needs $|P_{\mathrm{sim}}|^2$ space for storing the simulation preorder as a partial order on simulation equivalence classes and $|\Sigma| \log |P_{\mathrm{sim}}|$ space for storing the simulation equivalence class for any state. Hence, the bit space complexities of GPP and Space-Céc can be considered quasi-optimal. As far as time complexity is concerned, the algorithms RT and Time-Céc both feature the best time bound $O(|P_{\mathrm{sim}}||{\to}|)$.

We present here a novel space and time Efficient Simulation algorithm, called ESim, which features a time complexity in $O(|P_{\mathrm{sim}}||{\to}| \log |\Sigma|)$ and a bit space complexity in $O(|P_{\mathrm{sim}}|^2 \log |P_{\mathrm{sim}}| + |\Sigma| \log |\Sigma|)$. Thus, ESim reaches the best space bound of GPP and significantly improves the GPP time bound $O(|P_{\mathrm{sim}}|^2|{\to}|)$ by replacing a multiplicative factor $|P_{\mathrm{sim}}|$ with $\log |\Sigma|$. Furthermore, ESim significantly improves the RT space bound $O(|P_{\mathrm{sim}}||\Sigma| \log |\Sigma|)$ and closely approaches the best time bound $O(|P_{\mathrm{sim}}||{\to}|)$ of RT and Time-Céc.

ESim is a partition refinement algorithm, meaning that it maintains and iteratively refines a so-called partition-relation pair $\langle P, \trianglelefteq \rangle$, where P is a partition of Σ that overapproximates the final simulation partition P_{sim}, while $\trianglelefteq$ is a binary relation over P which overapproximates the final simulation preorder. ESim relies on the following three main points, which in particular allow to attain the above complexity bounds.

(1) Two distinct notions of partition and relation stability for a partition-relation pair are introduced. Accordingly, at a logical level, ESim is designed as a partition refinement algorithm which iteratively performs two clearly distinct refinement steps: the refinement of the current partition P which splits some blocks of P and the refinement of the relation $\trianglelefteq$ which removes some pairs of blocks from $\trianglelefteq$.
(2) ESim exploits a logical characterization of partition refiners, i.e. blocks of P that allow to split the current partition P, which admits an efficient implementation.
(3) ESim only relies on data structures, like lists and matrices, that are indexed on and contain blocks of the current partition P. The hard task here is to devise efficient ways to keep updated these partition-based data structures along the iterations of ESim. We show that this can be done efficiently, in particular by resorting to Hopcroft's "process the smaller half" principle [7] when updating a crucial data structure after a partition split.

Due to lack of space, some auxiliary algorithms and the proofs of all the results are omitted.

2 Background

Notation. If $R \subseteq \Sigma \times \Sigma$ is any relation and $X \subseteq \Sigma$ then $R(X) \triangleq \{x' \in \Sigma \mid \exists x \in X.\ (x, x') \in R\}$. Recall that R is a preorder relation when it is reflexive and transitive. If f is a function defined on $\wp(\Sigma)$ and $x \in \Sigma$ then we often write $f(x)$ to mean $f(\{x\})$. $\mathrm{Part}(\Sigma)$ denotes the set of partitions of Σ. If $P \in \mathrm{Part}(\Sigma)$, $s \in \Sigma$ and $S \subseteq \Sigma$ then $P(s)$ denotes the block of P that contains s while $P(S) = \cup_{s \in S} P(s)$. $\mathrm{Part}(\Sigma)$ is endowed with the standard partial order $\preceq$: $P_1 \preceq P_2$, i.e. P_2 is coarser than P_1, iff for any $s \in \Sigma$, $P_1(s) \subseteq P_2(s)$. If $P_1 \preceq P_2$ and $B \in P_1$ then $P_2(B)$ is a block of P_2 which is also denoted by $\mathrm{parent}_{P_2}(B)$. For a given nonempty 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 $B \cap S$ and $B \smallsetminus S$, where we also allow no splitting, namely $\mathit{Split}(P, S) = P$ (this happens exactly when $P(S) = S$).

Simulation Preorder and Equivalence. A transition system $(\Sigma, \rightarrow)$ consists of a set Σ of states and of a transition relation $\rightarrow \subseteq \Sigma \times \Sigma$. Given a set AP of atoms (of some specification language), a Kripke structure (KS) $\mathcal{K} = (\Sigma, \rightarrow, \ell)$ over AP consists of a transition system $(\Sigma, \rightarrow)$ together with a state labeling function $\ell : \Sigma \rightarrow \wp(AP)$. The state partition induced by ℓ is denoted by $P_\ell \triangleq \{\{s' \in \Sigma \mid \ell(s) = \ell(s')\} \mid s \in \Sigma\}$. The predecessor/successor transformers $\mathrm{pre}, \mathrm{post} : \wp(\Sigma) \rightarrow \wp(\Sigma)$ are defined as usual: $\mathrm{pre}(T) \triangleq \{s \in \Sigma \mid \exists t \in T.\ s{\rightarrow}t\}$ and $\mathrm{post}(S) \triangleq \{t \in \Sigma \mid \exists s \in S.\ s{\rightarrow}t\}$. If $S_1, S_2 \subseteq \Sigma$ then $S_1 {\rightarrow^\exists} S_2$ iff there exist $s_1 \in S_1$ and $s_2 \in S_2$ such that $s_1 {\rightarrow} s_2$.

A relation $R \subseteq \Sigma \times \Sigma$ is a simulation on a Kripke structure $(\Sigma, \rightarrow, \ell)$ if for any $s, s' \in \Sigma$, if $s' \in R(s)$ then:

(A) $\ell(s) = \ell(s')$;
(B) for any $t \in \Sigma$ such that $s{\rightarrow}t$, there exists $t' \in \Sigma$ such that $s'{\rightarrow}t'$ and $t' \in R(t)$.

Given $s, t \in \Sigma$, t simulates s, denoted by $s \leq t$, if there exists a simulation relation R such that $t \in R(s)$. It turns out that the largest simulation on a given KS exists, is a preorder relation called simulation preorder and is denoted by R_{sim}. Thus, for any $s, t \in \Sigma$, $s \leq t$ iff $(s, t) \in R_{\mathrm{sim}}$. Simulation equivalence R_{simeq} is the symmetric reduction of R_{sim}, namely $R_{\mathrm{simeq}} \triangleq R_{\mathrm{sim}} \cap R_{\mathrm{sim}}^{-1}$, so that $(s, t) \in R_{\mathrm{simeq}}$ iff $s \leq t$ and $t \leq s$. $P_{\mathrm{sim}} \in \mathrm{Part}(\Sigma)$ denotes the partition corresponding to the equivalence R_{simeq} and is called the simulation partition.

3 Logical Simulation Algorithm

A partition-relation pair $\mathcal{P} = \langle P, \trianglelefteq \rangle$, PR for short, is a state partition $P \in \mathrm{Part}(\Sigma)$ together with a binary relation $\trianglelefteq \subseteq P \times P$ between blocks of P. We write $B \triangleleft C$ when $B \trianglelefteq C$ and $B \neq C$ and $(B', C') \trianglelefteq (B, C)$ when $B' \trianglelefteq B$ and $C' \trianglelefteq C$. When $\trianglelefteq$ is a preorder/partial order then $\mathcal{P}$ is called, respectively, a preorder/partial order PR.

PRs allow to represent symbolically, i.e. through state partitions, a relation between states. A relation $R \subseteq \Sigma \times \Sigma$ induces a PR $\mathrm{PR}(R) = \langle P, \trianglelefteq \rangle$ defined as follows:

$$\forall s \in \Sigma.\, P(s) \triangleq \{t \in \Sigma \mid R(s) = R(t)\}; \quad \forall s, t \in \Sigma.\, P(s) \trianglelefteq P(t) \text{ iff } t \in R(s).$$

It is easy to note that if R is a preorder then $\mathrm{PR}(R)$ is a partial order PR. On the other hand, a PR $\mathcal{P} = \langle P, \trianglelefteq \rangle$ induces the following relation $\mathrm{Rel}(\mathcal{P}) \subseteq \Sigma \times \Sigma$:

$$(s,t) \in \mathrm{Rel}(\mathcal{P}) \;\Leftrightarrow\; P(s) \trianglelefteq P(t).$$

Here, if $\mathcal{P}$ is a preorder PR then $\mathrm{Rel}(\mathcal{P})$ is clearly a preorder.

A PR $\mathcal{P} = \langle P, \trianglelefteq \rangle$ is defined to be a simulation PR on a KS $\mathcal{K}$ when $\mathrm{Rel}(\mathcal{P})$ is a simulation on $\mathcal{K}$, namely when $\mathcal{P}$ represents a simulation relation between states. Hence, if $\mathcal{P}$ is a simulation PR and $P(s) = P(t)$ then s and t are simulation equivalent, while if $P(s) \trianglelefteq P(t)$ then t simulates s.

Given a PR $\mathcal{P} = \langle P, \trianglelefteq \rangle$, the map $\mu_{\mathcal{P}} : \wp(\Sigma) \to \wp(\Sigma)$ is defined as follows:

$$\text{for any } X \in \wp(\Sigma),\; \mu_{\mathcal{P}}(X) \triangleq \mathrm{Rel}(\mathcal{P})(X) = \cup\{C \in P \mid \exists s \in X.\, P(s) \trianglelefteq C\}.$$

Note that, for any $s \in \Sigma$, $\mu_{\mathcal{P}}(s) = \mu_{\mathcal{P}}(P(s)) = \cup\{C \in P \mid P(s) \trianglelefteq C\}$. For preorder PRs, this map allows us to characterize the property of being a simulation PR as follows.

Theorem 3.1. *Let $\mathcal{P} = \langle P, \trianglelefteq \rangle$ be a preorder PR. Then, $\mathcal{P}$ is a simulation iff*

(i) *if $B \trianglelefteq C$, $b \in B$ and $c \in C$ then $\ell(b) = \ell(c)$;*
(ii) *if $B \to^{\exists} C$ and $B \trianglelefteq D$ then $D \to^{\exists} \mu_{\mathcal{P}}(C)$;*
(iii) *for any $C \in P$, $P = \mathit{Split}(P, \mathrm{pre}(\mu_{\mathcal{P}}(C)))$.*

By Theorem 3.1, assuming that condition (i) holds, there are two possible reasons for a PR $\mathcal{P} = \langle P, \trianglelefteq \rangle$ for not being a simulation:

(1) There exist $B, C, D \in P$ such that $B \to^{\exists} C$, $B \trianglelefteq D$, but $D \not\to^{\exists} \mu_{\mathcal{P}}(C)$; in this case we say that the block C is a *relation refiner* for $\mathcal{P}$.
(2) There exist $B, C \in P$ such that $B \cap \mathrm{pre}(\mu_{\mathcal{P}}(C)) \neq \varnothing$ and $B \smallsetminus \mathrm{pre}(\mu_{\mathcal{P}}(C)) \neq \varnothing$; in this case we say that the block C is a *partition refiner* for $\mathcal{P}$.

We therefore define $\mathrm{RRefiner}(\mathcal{P})$ and $\mathrm{PRefiner}(\mathcal{P})$ as the sets of blocks of P that are, respectively, relation and partition refiners for $\mathcal{P}$. Accordingly, $\mathcal{P}$ is defined to be relation or partition *stable* when, respectively, $\mathrm{RRefiner}(\mathcal{P}) = \varnothing$ or $\mathrm{PRefiner}(\mathcal{P}) = \varnothing$. Then, Theorem 3.1 can be read as follows: $\mathcal{P}$ is a simulation iff $\mathcal{P}$ satisfies condition (i) and is both relation and partition stable.

If $C \in \mathrm{PRefiner}(\mathcal{P})$ then P is first refined to $P' \triangleq \mathit{Split}(P, \mathrm{pre}(\mu_{\mathcal{P}}(C)))$, i.e. P is split w.r.t. the splitter $S = \mathrm{pre}(\mu_{\mathcal{P}}(C))$. Accordingly, the relation $\trianglelefteq$ on P is transformed into the following relation $\trianglelefteq'$ defined on P':

$$\trianglelefteq' \;\triangleq\; \{(D,E) \in P' \times P' \mid \mathrm{parent}_P(D) \trianglelefteq \mathrm{parent}_P(E)\} \qquad (\dagger)$$

Hence, two blocks D and E of the refined partition P' are related by $\trianglelefteq'$ if their parent blocks $\mathrm{parent}_P(D)$ and $\mathrm{parent}_P(E)$ in P were related by $\trianglelefteq$. Hence, if $\mathcal{P}' = \langle P', \trianglelefteq' \rangle$ then for all $D \in P'$, we have that $\mu_{\mathcal{P}'}(D) = \mu_{\mathcal{P}}(\mathrm{parent}_P(D))$. We will show that this refinement of $\langle P, \trianglelefteq \rangle$ is correct because if $B \in P$ is split into $B \smallsetminus S$ and $B \cap S$ then all the states in $B \smallsetminus S$ are not simulation equivalent to all the states in $B \cap S$. Note that if $B \in P$ has been split into $B \cap S$ and $B \smallsetminus S$ then both $B \cap S \trianglelefteq' B \smallsetminus S$ and $B \smallsetminus S \trianglelefteq' B \cap S$ hold, and consequently $\mathcal{P}'$ becomes relation unstable.

```
ESim(PR $\langle P, \trianglelefteq \rangle$)
  $Initialize()$; $PStabilize()$; bool PStable := $RStabilize()$; bool RStable := tt;
  while $\neg$(PStable & RStable) do
    if $\neg$ PStable then {RStable := $PStabilize()$; PStable := tt;}
    if $\neg$ RStable then {PStable := $RStabilize()$; RStable := tt;}

bool $PStabilize()$
  $P_{\mathrm{old}} := P$;
  while $\exists C \in \mathrm{PRefiner}(\mathcal{P})$ do
    $S := \mathrm{pre}(\mu_{\mathcal{P}}(C))$; $P := Split(S)$;
    forall $(D, E) \in P \times P$ do $D \trianglelefteq E := \mathrm{parent}_P(D) \trianglelefteq \mathrm{parent}_P(E)$;
  return $(P = P_{\mathrm{old}})$;

bool $RStabilize()$
  // Precondition: PStable = tt
  $\trianglelefteq_{\mathrm{old}} := \trianglelefteq$; Delete := $\varnothing$;
  while $\exists C \in \mathrm{RRefiner}(\mathcal{P})$ do
    Delete := Delete $\cup \{(B, D) \in P \times P \mid B \trianglelefteq D,\ B \to^{\exists} C,\ D \not\to^{\exists} \mu_{\mathcal{P}}(C)\}$;
  $\trianglelefteq := \trianglelefteq \smallsetminus$ Delete;
  return $(\trianglelefteq = \trianglelefteq_{\mathrm{old}})$;
```

Fig. 1. Logical Simulation Algorithm.

On the other hand, if $\mathcal{P}$ is partition stable and $C \in \mathrm{RRefiner}(\mathcal{P})$ then we will show that $\trianglelefteq$ can be safely refined to the following relation $\trianglelefteq'$:

$$\begin{aligned} \trianglelefteq' &\triangleq \trianglelefteq \smallsetminus \{(B, D) \in P \times P \mid B \to^{\exists} C,\ B \trianglelefteq D,\ D \not\to^{\exists} \mu_{\mathcal{P}}(C)\} \\ &= \{(B, D) \in P \times P \mid B \trianglelefteq D,\ \big(B \to^{\exists} C \Rightarrow D \to^{\exists} \mu_{\mathcal{P}}(C)\big)\} \end{aligned} \qquad (\ddagger)$$

because if $(B, D) \in \trianglelefteq \smallsetminus \trianglelefteq'$ then all the states in D cannot simulate all the states in B.

The above facts lead us to design a basic simulation algorithm ESim described in Figure 1. ESim maintains a PR $\mathcal{P} = \langle P, \trianglelefteq \rangle$, which initially is $\langle P_\ell, \mathrm{id} \rangle$ and is iteratively refined as follows:

$PStabilize()$: If $\langle P, \trianglelefteq \rangle$ is not partition stable then the partition P is split for $\mathrm{pre}(\mu_{\mathcal{P}}(C))$ as long as a partition refiner C for $\mathcal{P}$ exists, and when this happens the relation $\trianglelefteq$ is transformed to $\trianglelefteq'$ as defined by (†); at the end of this process, we obtain a PR $\mathcal{P}' = \langle P', \trianglelefteq' \rangle$ which is partition stable and if P has been actually refined, i.e. $P' \prec P$ then the current PR $\mathcal{P}'$ becomes relation unstable.

$RStabilize()$: If $\langle P, \trianglelefteq \rangle$ is not relation stable then the relation $\trianglelefteq$ is refined to $\trianglelefteq'$ as described by (‡) as long as a relation refiner for $\mathcal{P}$ exists; hence, at the end of this refinement process $\langle P, \trianglelefteq' \rangle$ becomes relation stable but possibly partition unstable.

Moreover, the following properties of the current PR of ESim hold.

Lemma 3.2. *In any run of* ESim, *the following two conditions hold:*

(i) *If $PStabilize()$ is called on a partial order PR $\langle P, \trianglelefteq \rangle$ then at the exit we obtain a PR $\langle P', \trianglelefteq' \rangle$ which is a preorder.*
(ii) *If $RStabilize()$ is called on a preorder PR $\langle P, \trianglelefteq \rangle$ then at the exit we obtain a PR $\langle P, \trianglelefteq' \rangle$ which is a partial order.*

The main loop of ESim terminates when the current PR $\langle P, \trianglelefteq \rangle$ becomes both partition and relation stable. By the above Lemma 3.2, the output PR $\mathcal{P}$ of ESim is a partial order, and hence a preorder, so that Theorem 3.1 can be applied to $\mathcal{P}$ which then results to be a simulation PR. It turns out that this algorithm is correct, meaning that the output PR $\mathcal{P}$ actually represents the simulation preorder.

Theorem 3.3 (Correctness). *Let Σ be finite.* ESim *is correct, i.e.,* ESim *terminates on any input and if $\langle P, \trianglelefteq \rangle$ is the output PR of* ESim *on input $\langle P_\ell, \mathrm{id} \rangle$ then for any $s, t \in \Sigma$, $s \leq t \;\Leftrightarrow\; P(s) \trianglelefteq P(t)$.*

4 Efficient Implementation

4.1 Data Structures

ESim is implemented by relying on the following data structures.

States: A state s is represented by a record that contains the list $\mathrm{post}(s)$ of its successors, a pointer s.block to the block $P(s)$ that contains s and a boolean flag used for marking purposes. The whole state space Σ is represented as a doubly linked list of states. $\{\mathrm{post}(s)\}_{s\in\Sigma}$ therefore represents the input transition system.

Partition: The states of any block B of the current partition P are consecutive in the list Σ, so that B is represented by two pointers begin and end: B.begin is the first state of B in Σ and B.end is the successor of the last state of B in Σ, i.e., $B = [B.\text{begin}, B.\text{end}[$. Moreover, B stores a boolean flag B.intersection and a block pointer B.brother whose meanings are as follows: after a call to $Split(P, S)$ for splitting P w.r.t. a set of states S, if $B_1 = B \cap S$ and $B_2 = B \smallsetminus S$, for some $B \in P$ that has been split by S then B_1.intersection $=$ **tt** and B_2.intersection $=$ **ff**, while B_1.brother points to B_2 and B_2.brother points to B_1. If instead B has not been split by S then B.intersection $=$ **null** and B.brother $=$ **null**. Also, any block B stores in $Rem(B)$ a list of blocks of P, which is used by $RStabilize()$, and in B.preE the list of blocks $C \in P$ such that $C \to^{\exists} B$. Finally, any block B stores in B.size the size of B, in B.count an integer counter bounded by $|P|$ which is used by $PStabilize()$ and a pair of boolean flags used for marking purposes. The current partition P is stored as a doubly linked list of blocks.

Relation: The current relation $\trianglelefteq$ on P is stored as a resizable $|P| \times |P|$ boolean matrix. Recall that insert operations in a resizable array (whose capacity is doubled as needed) take amortized constant time and that a resizable matrix (or table) can be implemented as a resizable array of resizable arrays. The boolean matrix $\trianglelefteq$ is resized by adding a new entry to $\trianglelefteq$, namely a new row and a new column, for any block B that is split into two new blocks $B \smallsetminus S$ and $B \cap S$. The old entry B becomes the entry for the new block $B \smallsetminus S$ while the new entry is used for the new block $B \cap S$.

```
bool $PStabilize()$
  list⟨Block⟩ $split := \varnothing$;
  while $(C := FindPRefiner()) \neq$ null) do
    list⟨State⟩ $S := \text{pre}\mu(C)$; $split := Split(S)$; $updateRel(split)$;
    $updateBCount(split)$; $updatePreE()$; $updateCount(split)$; $updateRem(split)$;
  return $(split = \varnothing)$;

Block $FindPRefiner()$
  forall $B \in P$ do
    list⟨Block⟩ $p := Post(B)$;
    forall $C \in p$ do if $(\text{Count}(B,C) = 1)$ then return $C$;
  return null;

list⟨Block⟩ $Post$(Block $B$)
  list⟨Block⟩ $p := \varnothing$;
  forall $b \in B$, do
    forall $c \in \text{post}(b)$ do
      Block $C := c$.block;
      if unmarked1$(C)$ then {mark1$(C)$; $C$.count $= 0$; $p$.append$(C)$;}
      if unmarked2$(C)$ then {mark2$(C)$; $C$.count++;}
    forall $C \in p$ do unmark2$(C)$;
  forall $C \in p$ do {unmark1$(C)$; if $(C$.count $= B$.size$)$ then $p$.remove$(C)$;}
  return $p$;
```

Fig. 2. $PStabilize()$ Algorithm.

Auxiliary Data Structures: We store and maintain a resizable boolean matrix BCount and a resizable integer matrix Count, both indexed over P, whose meanings are as follows:

$$\text{BCount}(B,C) \triangleq \begin{cases} 1 & \text{if } B \rightarrow^{\exists} C \\ 0 & \text{if } B \not\rightarrow^{\exists} C \end{cases} \qquad \text{Count}(B,C) \triangleq \textstyle\sum_{E \trianglerighteq C} \text{BCount}(B,E).$$

Hence, $\text{Count}(B,C)$ stores the number of blocks E such that $C \trianglelefteq E$ and $B \rightarrow^{\exists} E$. The table Count allows to implement the test $B \not\rightarrow^{\exists} \text{pre}(\mu_{\mathcal{P}}(C))$ in constant time as $\text{Count}(B,C) = 0$.

The data structures BCount, preE, Count and Rem are initialized by a function *Initialize()* at line 2 of ESim, which is here omitted.

4.2 Partition Stability

Our implementation of ESim will exploit the following logical characterization of partition refiners.

Theorem 4.1. *Let* $\langle P, \trianglelefteq \rangle$ *be a partial order PR. Then,* $\text{PRefiner}(\langle P, \trianglelefteq \rangle) \neq \varnothing$ *iff there exist* $B, C \in P$ *such that the following three conditions hold:*

(i) $B \rightarrow^{\exists} C$;
(ii) *for any* $C' \in P$, *if* $C \lhd C'$ *then* $B \not\rightarrow^{\exists} C'$;
(iii) $B \not\subseteq \mathrm{pre}(C)$.

Notice that this characterization of partition refiners requires that the current PR is a partial order relation and, by Lemma 3.2, for any call to *PStabilize*(), this is actually guaranteed by the ESim algorithm.

The algorithm in Figure 2 is an implementation of the *PStabilize*() function that relies on Theorem 4.1 and on the above data structures. The function *FindPRefiner*() implements the conditions of Theorem 4.1: it returns a partition refiner for the current PR $\mathcal{P} = \langle P, \unlhd \rangle$ when this exists, otherwise it returns a null pointer. Given a block $B \in P$, the function *Post*(B) returns a list of blocks $C \in P$ that satisfy conditions (i) and (iii) of Theorem 4.1, i.e., those blocks C such that $B \rightarrow^{\exists} C$ and $B \not\subseteq \mathrm{pre}(C)$. This is accomplished through the counter C.count that at the exit of the for-loop at lines 14-19 in Figure 2 stores the number of states in B having (at least) an outgoing transition to C, i.e., C.count $= |B \cap \mathrm{pre}(C)|$. Hence, we have that:

$$B \rightarrow^{\exists} C \text{ and } B \not\subseteq \mathrm{pre}(C) \;\Leftrightarrow\; 1 \leq C.\text{count} < B.\text{size}.$$

Then, for any candidate partition refiner $C \in Post(B)$, it remains to check condition (ii) of Theorem 4.1. This condition is checked in *FindPRefiner*() by testing whether Count$(B, C) = 1$: this is correct because Count$(B, C) \geq 1$ holds since $C \in Post(B)$ and therefore $B \rightarrow^{\exists} C$, so that

$$\text{Count}(B, C) = 1 \text{ iff } \forall C' \in P. C \lhd C' \Rightarrow B \not\rightarrow^{\exists} C'.$$

Hence, if Count$(B, C) = 1$ holds at line 10 of *FindPRefiner*(), by Theorem 4.1, C is a partition refiner. Once a partition refiner C has been returned by *Post*(B), *PStabilize*() splits the current partition P w.r.t. the splitter $S = \mathrm{pre}(\mu_{\mathcal{P}}(C))$ by calling the function *Split*(S), updates the relation $\unlhd$ as defined by equation (†) in Section 3 by calling *updateRel*(), updates the data structures BCount, preE, Count and *Rem*, and then check again whether a partition refiner exists. At the exit of the main while-loop of *PStabilize*(), the current PR $\langle P, \unlhd \rangle$ is partition stable.

PStabilize() calls the functions preμ() and *Split*() that are here omitted. Recall that the states of a block B of P are consecutive in the list of states Σ, so that B is represented as $B = [B.\text{begin}, B.\text{end}[$. The implementation of *Split*(S) is quite standard (see e.g. [12]): this is based on a linear scan of the states in S and for each state in S performs some constant time operations. Hence, *Split*(S) takes $O(|S|)$ time. Also, *Split*(S) returns the list *split* of blocks $B \smallsetminus S$ such that $\varnothing \subsetneq B \smallsetminus S \subsetneq B$ (i.e., B.intersection = **ff**). Let us remark that a call *Split*(S) may affect the ordering of the states in the list Σ because states are moved from old blocks to newly generated blocks.

We will show that the overall time complexity of *PStabilize*() along a whole run of ESim is in $O(|P_{\mathrm{sim}}||\rightarrow|)$.

4.3 Updating Data Structures

In the function *PStabilize*(), after calling *Split*(S), firstly we need to update the boolean matrix that stores the relation $\unlhd$ in accordance with definition (†) in Section 3. After that, since both P and $\trianglelefteq$ are changed we need to update the data structures BCount, preE, Count and Rem. We omit the implementations of the functions *updateRel*$()$, *updateBCount*$()$, *updatePreE*$()$ and *updateRem*$()$, which are quite straightforward.

The function *updateCount*$()$ is in Figure 3 and deserves special care in order to design a time efficient implementation. The core of the *updateCount*$()$ algorithm follows Hopcroft's "process the smaller half" principle [7] for updating the integer matrix Count. Let P' be the partition which is obtained by splitting the partition P w.r.t. the splitter S. Let B be a block of P that has been split into $B \cap S$ and $B \smallsetminus S$. Thus, we need to update Count$(B \cap S, C)$ and Count$(B \smallsetminus S, C)$ for any $C \in P'$ by knowing Count$(B, \mathrm{parent}_P(C))$. Let us first observe that after lines 4-6 of $updateCount()$, we have that for any $B, C \in P'$, Count$(B, C) =$ Count$(\mathrm{parent}_P(B), \mathrm{parent}_P(C))$. Let X be the block in $\{B \cap S, B \smallsetminus S\}$ with the smaller size, and let Z be the other block, so that $|X| \leq |B|/2$ and $|X| + |Z| = |B|$. Let C be any block in P'. We set Count(X, C) to 0, while Count(Z, C) is left unchanged, namely Count$(Z, C) =$ Count(B, C). We can correctly update both Count(Z, C) and Count(X, C) by just scanning all the outgoing transitions from X. In fact, if $x \in X$, $x \to y$ and the block $P(y)$ is scanned for the first time then for all $C \trianglelefteq P(y)$, Count(X, C) is incremented by 1, while if $Z \not\to^{\exists} P(y)$, i.e. BCount$(Z, P(y)) = 0$, then Count(Z, C) is decremented by 1. The correctness of this procedure goes as follows:

(1) At the end, Count(X, C) is clearly correct because its value has been re-computed from scratch.
(2) At the end, Count(Z, C) is correct because Count(Z, C) initially stores the value Count(B, C), and if there exists some block D such that $C \trianglelefteq D$, $B \to^{\exists} D$ whereas $Z \not\to^{\exists} D$ — this is correctly implemented at line 19 as BCount$(Z, D) = 0$, since the date structure BCount is up to date — then necessarily $X \to^{\exists} D$, because B has been split into X and Z, so that $D = P(y)$ for some $y \in \mathrm{post}(X)$, namely D has been taken into account by some increment Count(X, C)++ and consequently Count(Z, C) is decremented by 1 at line 19.

Moreover, if some block $D \in P' \smallsetminus \{B \cap S, B \smallsetminus S\}$ is such that both $D \to^{\exists} X$ and $D \to^{\exists} Z$ hold then for all the blocks $C \in P$ such that $C \trianglelefteq X$ (or, equivalently, $C \trianglelefteq Z$), we need to increment Count(D, C) by 1. This is done at lines 21-22 by relying on the updated date structures preE and BCount.

Let us observe that the time complexity of a single call of *updateCount*$(split)$ is

$$|P|\big(|split| + \textstyle\sum_{X \in split}\big(|\{(x, y) \mid x \in X, y \in \Sigma, x \to y\}| + |\{(X, D) \mid D \in P, X \to^{\exists} D\}|\big)\big).$$

Hence, let us calculate the overall time complexity of *updateCount*$()$. If X and X' are two blocks that are scanned in two different calls of *updateCount* and $X' \subseteq X$ then $|X'| \leq |X|/2$. Consequently, any transition $x \to y$ at line 16 and $D \to^{\exists} X$ at line 21 can be scanned in some call of *updateCount*$()$ at most $\log_2 |\Sigma|$ times. Thus, the overall time complexity of *updateCount*$()$ is in $O(|P_{\mathrm{sim}}||{\to}| \log |\Sigma|)$.

4.4 Relation Stability

The basic procedure $RStabilize()$ in Figure 1 is implemented by the algorithm in Figure 4. Let $\mathcal{P}^{\mathrm{in}} = \langle P, \trianglelefteq^{\mathrm{in}} \rangle$ be the current PR when calling $RStabilize()$. For each rela-

```
// Precondition: BCount and preE are updated with the current PR
updateCount(list⟨Block⟩ split)
    forall $B \in$ split do addNewEntry($B$) in matrix Count;
    forall $B \in P$, $C \in$ split do
        if ($B$.intersection = tt) then Count($B, C$) := Count($B$.brother, $C$.brother);
        else Count($B, C$) := Count($B$, $C$.brother);
    forall $C \in P$, $B \in$ split do
        if ($C$.intersection = ff) then Count($B, C$) := Count($B$.brother, $C$);
    forall $C \in P$ do unmark($C$);
    forall $B \in$ split do
        // Update Count($B, \cdot$) and Count($B$.brother, $\cdot$)
        Block $X$, $Z$;
        if ($B$.size $\leq$ $B$.brother.size) then {$X$ := $B$; $Z$ := $B$.brother;}
        else {$X$ := $B$.brother; $Z$ := $B$;}
        forall $C \in P$ do {Count($X, C$) := 0; /* Count($Z, C$) := Count($B, C$); */}
        forall $x \in X$, $y \in \mathrm{post}(x)$ such that unmarked($y$.block) do
            mark($y$.block);
            forall $C \in P$ such that $C \trianglelefteq y.block$ do
                Count($X$,$C$)++; if (BCount($Z$, $y$.block) = 0) then Count($Z$,$C$)−−;
        // For all $D \notin \{B, B.\mathrm{brother}\}$, updateCount($D, \cdot$)
        forall $D \in X$.preE such that ($D \neq X$ & $D \neq Z$ & BCount($D, Z$) = 1) do
            forall $C \in P$ such that $C \trianglelefteq X$ do Count($D, C$)++;
```

Fig. 3. *updateCount*() function.

tion refiner $C \in P$, $RStabilize()$ must iteratively refine the initial relation $\trianglelefteq^{\mathrm{in}}$ in accordance with equation (‡) in Section 3. Hence, if $B {\to}^{\exists} C$, $B \trianglelefteq D$ and $D \not\to^{\exists} \mu_{\mathcal{P}^{\mathrm{in}}}(C)$, the entry $B \trianglelefteq D$ of the boolean matrix that represents the relation $\trianglelefteq$ must be set to **ff**. Thus, the idea is to store and incrementally maintain for each block $C \in P$ a list $Rem(C)$ of blocks $D \in P$ such that: (A) If C is a relation refiner for $\mathcal{P}^{\mathrm{in}}$ then $Rem(C) \neq \varnothing$; (B) If $D \in Rem(C)$ then necessarily $D \not\to^{\exists} \mu^{\mathrm{in}}_{\mathcal{P}}(C)$. It turns out that C is a relation refiner for $\mathcal{P}^{\mathrm{in}}$ iff there exist blocks B and D such that $B {\to}^{\exists} C$, $D \in Rem(C)$ and $B \trianglelefteq D$. Hence, the set of blocks $Rem(C)$ is reminiscent of the set of states $remove(s)$ used in Henzinger et al.'s [5] simulation algorithm, since each pair (B, D) which must be removed from the relation $\trianglelefteq$ is such that $D \in Rem(C)$, for some block C.

Initially, namely at the first call of $RStabilize()$ by ESim, $Rem(C)$ is set by the function *Initialize*() to $\{D \in P \mid D {\to}^{\exists} \Sigma,\ D \not\to^{\exists} \mu_{\mathcal{P}}(C)\}$. Hence, $RStabilize()$ scans all the blocks in the current partition P and selects those blocks C such that $Rem(C) \neq \varnothing$, which are therefore candidate to be relation refiners. Then, by scanning all the blocks $B \in C$.preE and $D \in Rem(C)$, if $B \trianglelefteq D$ holds then the entry $B \trianglelefteq D$ must be set to **ff**. However, the removal of the pair (B, D) from the current relation $\trianglelefteq$ may affect the function $\mu_{\mathcal{P}}$. This is avoided by making a copy $oldRem(C)$ of all the $Rem(C)$'s at the beginning of $RStabilize()$ and then using this copy. During the main for-loop of

```
bool $RStabilize()$
    // $\mu^{\mathrm{in}}_{\mathcal{P}} := \mu_{\mathcal{P}}$;
    forall $C \in P$ do $\{oldRem(C) := Rem(C);\ Rem(C) = \varnothing;\}$
    bool Removed := ff;
    forall $C \in P$ such that $oldRem(C) \neq \varnothing$ do
        // Invariant (Inv): $\forall C \in P.\, Rem(C) = \{D \in P \mid D \rightarrow^{\exists} \mu^{\mathrm{in}}_{\mathcal{P}}(C),\ D \not\rightarrow^{\exists} \mu_{\mathcal{P}}(C)\}$
        forall $B \in C$.preE, $D \in oldRem(C)$ such that $(B \trianglelefteq D)$ do
            $B \trianglelefteq D :=$ ff; Removed := tt;
            // update Count and $Rem$
            forall $F \in D$.preE do
                Count$(F, B) :=$ Count$(F, B) - 1$;
                if (Count$(F, B) = 0$ & $Rem(B) = \varnothing$) then $Rem(B)$.append$(F)$;
    return $\neg$Removed;
```

Fig. 4. $RStabilize()$ Algorithm.

$RStabilize()$, $Rem(C)$ must satisfy the following invariant property:

$$\text{(Inv): } \forall C \in P.\, Rem(C) = \{D \in P \mid D \rightarrow^{\exists} \mu^{\mathrm{in}}_{\mathcal{P}}(C),\ D \not\rightarrow^{\exists} \mu_{\mathcal{P}}(C)\}.$$

This means that at the beginning of $RStabilize()$, any $Rem(C)$ is set to empty, and after the removal of a pair (B, D) from $\trianglelefteq$, since $\mu_{\mathcal{P}}(B)$ has changed, we need: (i) to update the matrix Count, for all the entries (F, B) where $F \rightarrow^{\exists} D$, and (ii) to check if there is some block F such that $F \not\rightarrow^{\exists} \mu_{\mathcal{P}}(B)$, because any such F must be added to $Rem(B)$ in order to maintain the invariant property (Inv).

4.5 Complexity

The time complexity of the algorithm ESim relies on the following key properties:

(1) The overall number of partition refiners found by ESim is in $O(|P_{\mathrm{sim}}|)$. Moreover, the overall number of newly generated blocks by the splitting operations performed by calling $Split(S)$ at line 4 of $PStabilize()$ is in $O(|P_{\mathrm{sim}}|)$. In fact, let $\{P_i\}_{i \in [0,n]}$ be the sequence of different partitions computed by ESim where P_0 is the initial partition P_ℓ, P_n is the final partition P_{sim} and for all $i \in [1, n]$, P_i is the partition after the i-th call to $Split(S)$, so that $P_i \prec P_{i-1}$. The number of new blocks which are produced by a call $Split(S)$ that refines P_i to P_{i+1} is $2(|P_{i+1}| - |P_i|)$. Thus, the overall number of newly generated blocks is $\sum_{i=1}^{n} 2(|P_i| - |P_{i-1}|) = 2(|P_{\mathrm{sim}}| - |P_\ell|) \in O(|P_{\mathrm{sim}}|)$.

(2) The invariant (Inv) of the sets $Rem(C)$ guarantees the following property: if C_1 and C_2 are two blocks that are selected by the for-loop at line 5 of $RStabilize()$ in two different calls of $RStabilize()$, and $C_2 \subseteq C_1$ (possibly $C_1 = C_2$) then $(\cup Rem(C_1)) \cap (\cup Rem(C_2)) = \varnothing$.

Theorem 4.2 (Complexity). ESim *runs in* $O(|P_{\mathrm{sim}}|^2 \log |P_{\mathrm{sim}}| + |\Sigma| \log |\Sigma|)$*-space and* $O(|P_{\mathrm{sim}}||{\rightarrow}| \log |\Sigma|)$*-time.*

5 Further Work

We see a couple of interesting avenues for further work. A first natural question arises: can the time complexity of ESim be further improved and reaches the time complexity of RT? This would require to eliminate the multiplicative factor $\log |\Sigma|$ from the time complexity of ESim and, presently, this seems to us quite hard to achieve. More in general, it would be interesting to investigate whether some lower space and time bounds can be stated for the simulation preorder problem. Secondly, ESim is designed for Kripke structures. While an adaptation of a simulation algorithm from Kripke structures to labeled transition systems (LTSs) can be conceptually simple, unfortunately such a shift may lead to some loss in both space and time complexities, as argued in [2]. We mention the works [1,6] and [8] that provide simulation algorithms for LTSs by adapting, respectively, RT and GPP. It is thus worth investigating whether and how ESim can be efficiently adapted to work with LTSs.

Acknowledgements. We acknowledge the contribution of Francesco Tapparo to a preliminary stage of this research which was informally presented in [11]. This work was partially supported by Microsoft Research SEIF 2013 Award and by the University of Padova under the project BECOM.

References

1. P.A. Abdulla, A, Bouajjani, L. Holík, L. Kaati and T. Vojnar. Computing simulations over tree automata. In *Proc. 14th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, LNCS 4963, pp. 93–108, 2008.
2. G. Cécé. Three simulation algorithms for labelled transition systems. Preprint cs.arXiv:1301.1638, arXiv.org, 2013.
3. R. Gentilini, C. Piazza and A. Policriti. From bisimulation to simulation: coarsest partition problems. *J. Automated Reasoning*, 31(1):73-103, 2003.
4. R. van Glabbeek and B. Ploeger. Correcting a space-efficient simulation algorithm. In *Proc. 20th Int. Conf. on Computer Aided Verification (CAV'08)*, LNCS 5123, pp. 517-529, 2008.
5. M.R. Henzinger, T.A. Henzinger and P.W. Kopke. Computing simulations on finite and infinite graphs. In *Proc. 36th IEEE FOCS*, 453-462, 1995.
6. L. Holík and J. Šimáček. Optimizing an LTS-simulation algorithm. *Computing and Informatics*, 29(6+):1337-1348, 2010.
7. J.E. Hopcroft. A $n \log n$ algorithm for minimizing states in a finite automaton. In Z. Kohavi and A. Paz eds., *Theory of Machines and Computations*, pp. 189-176, Academic Press, 1971.
8. J. Markovski. Saving time in a space-efficient simulation algorithm. In *Proc. 11th Int. Conf. on Quality Software*, pp. 244-251, IEEE, 2011.
9. R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973-989, 1987
10. F. Ranzato and F. Tapparo. A new efficient simulation equivalence algorithm. In *Proc. 22nd Annual IEEE Symp. on Logic in Computer Science (LICS'07)*, pp. 171-180, 2007.
11. F. Ranzato and F. Tapparo. A time and space efficient simulation algorithm. Short talk at *24th Annual IEEE Symposium on Logic in Computer Science (LICS'09)*, 2009.
12. F. Ranzato and F. Tapparo. An efficient simulation algorithm based on abstract interpretation. *Inf. Comput.*, 208(1):1-22, 2010.