An Efficient Simulation Algorithm based on Abstract Interpretation

FRANCESCO RANZATO FRANCESCO TAPPARO
Dipartimento di Matematica Pura ed Applicata, University of Padova, Italy
{ranzato,tapparo}@math.unipd.it

Abstract

A number of algorithms for computing the simulation preorder are available. Let Σ denote the state space, $\rightarrow$ the transition relation and P_{sim} the partition of Σ induced by simulation equivalence. The algorithms by Henzinger, Henzinger, Kopke and by Bloom and Paige run in $O(|\Sigma||\rightarrow|)$-time and, as far as time-complexity is concerned, they are the best available algorithms. However, these algorithms have the drawback of a space complexity that is more than quadratic in the size of the state space. The algorithm by Gentilini, Piazza, Policriti — subsequently corrected by van Glabbeek and Ploeger — appears to provide the best compromise between time and space complexity. Gentilini et al.'s algorithm runs in $O(|P_{\mathrm{sim}}|^2|\rightarrow|)$-time while the space complexity is in $O(|P_{\mathrm{sim}}|^2 + |\Sigma| \log |P_{\mathrm{sim}}|)$. We present here a new efficient simulation algorithm that is obtained as a modification of Henzinger et al.'s algorithm and whose correctness is based on some techniques used in applications of abstract interpretation to model checking. Our algorithm runs in $O(|P_{\mathrm{sim}}||\rightarrow|)$-time and $O(|P_{\mathrm{sim}}||\Sigma| \log |\Sigma|)$-space. Thus, this algorithm improves the best known time bound while retaining an acceptable space complexity that is in general less than quadratic in the size of the state space. An experimental evaluation showed good comparative results with respect to Henzinger, Henzinger and Kopke's algorithm.

1 Introduction

Abstraction techniques are widely used in model checking to hide some properties of the concrete model in order to define a reduced abstract model where to run the verification algorithm [1, 9]. Abstraction provides an effective solution to deal with the state-explosion problem that arises in model checking systems with parallel components [7]. The reduced abstract structure is required at least to weakly preserve a specification language $\mathcal{L}$ of interest: if a formula $\varphi \in \mathcal{L}$ is satisfied by the reduced abstract model then φ must hold on the original unabstracted model as well. Ideally, the reduced model should be strongly preserving w.r.t. $\mathcal{L}$: $\varphi \in \mathcal{L}$ holds on the concrete model if and only if φ is satisfied by the reduced abstract model. One common approach for abstracting a model consists in defining a logical equivalence or preorder on system states that weakly/strongly preserves a given temporal language. Moreover, this equivalence or preorder often arises as a behavioural relation in the context of process calculi [10]. Two well-known examples are bisimulation equivalence that strongly preserves expressive logics such as CTL^* and the full μ-calculus [5] and the simulation preorder that ensures weak preservation of universal and existential fragments of the μ-calculus like ACTL^* and ECTL^* as well as of linear-time languages like LTL [22, 25]. Simulation equivalence, namely the equivalence relation obtained as symmetric reduction of the simulation preorder, is particularly interesting because it can provide a significantly better state space reduction than bisimulation equivalence while retaining the ability of strongly preserving expressive temporal languages like ACTL^*.

State of the Art. It is known that computing simulation is harder than computing bisimulation [24]. Let $\mathcal{K} = \langle \Sigma, \rightarrow, \ell \rangle$ denote a Kripke structure on the state space Σ, with transition relation $\rightarrow$ and labeling function $\ell : \Sigma \rightarrow \wp(AP)$, for a given set AP of atomic propositions. Bisimulation equivalence can be computed by the well-known Paige and Tarjan's [26] algorithm that runs in $O(|\rightarrow| \log |\Sigma|)$-time. A number of algorithms for computing simulation equivalence exist, the most well known are by Henzinger, Henzinger and Kopke [23], Bloom and Paige [2], Bustan and Grumberg [6], Tan and Cleaveland [29] and Gentilini, Piazza and Policriti [18], this latter subsequently corrected by van Glabbeek and Ploeger [21].

The algorithms by Henzinger, Henzinger, Kopke and by Bloom and Paige run in $O(|\Sigma||\rightarrow|)$-time and, as far as time-complexity is concerned, they are the best available algorithms. However, both these algorithms have the drawback of a space complexity that is bounded from below by $\Omega(|\Sigma|^2)$. This is due to the fact that the simulation preorder is computed in an explicit way, i.e., for any state $s \in \Sigma$, the set of states that simulate s is explicitly given as output. This quadratic lower bound in the size of the state space is clearly a critical issue in model checking. There is therefore a strong motivation for designing simulation algorithms that are less demanding on space requirements. Bustan and Grumberg [6] provide a first solution in this direction. Let $P_{\rm sim}$ denote the partition corresponding to simulation equivalence on $\mathcal{K}$ so that $|P_{\rm sim}|$ is the number of simulation equivalence classes. Then, Bustan and Grumberg's algorithm has a space complexity in $O(|P_{\rm sim}|^2 + |\Sigma| \log |P_{\rm sim}|)$, although the time complexity in $O(|P_{\rm sim}|^4(|\rightarrow| + |P_{\rm sim}|^2) + |P_{\rm sim}|^2|\Sigma|(|\Sigma| + |P_{\rm sim}|^2|))$ remains a serious drawback. The simulation algorithm by Tan and Cleaveland [29] simultaneously computes also the state partition $P_{\rm bis}$ corresponding to bisimulation equivalence. Under the simplifying assumption of dealing with a total transition relation, this procedure has a time complexity in $O(|\rightarrow|(|P_{\rm bis}| + \log |\Sigma|))$ and a space complexity in $O(|\rightarrow|+|P_{\rm bis}|^2+|\Sigma| \log |P_{\rm bis}|)$ (the latter factor $|\Sigma| \log |P_{\rm bis}|$ does not appear in [29] and takes into account the relation that maps each state into its bisimulation equivalence class). The algorithm by Gentilini, Piazza and Policriti [18] appears to provide the best compromise between time and space complexity. Gentilini et al.'s algorithm runs in $O(|P_{\rm sim}|^2|\rightarrow|)$-time, namely it remarkably improves on Bustan and Grumberg's algorithm and is not directly comparable with Tan and Cleaveland's algorithm, while the space complexity $O(|P_{\rm sim}|^2 + |\Sigma| \log |P_{\rm sim}|)$ is the same of Bustan and Grumberg's algorithm and improves on Tan and Cleaveland's algorithm. Moreover, Gentilini et al. show experimentally that in most cases their procedure improves on Tan and Cleaveland's algorithm both in time and space.

Main Contributions. This work presents a new efficient simulation algorithm, called SA, that runs in $O(|P_{\rm sim}||\rightarrow|)$-time and $O(|P_{\rm sim}||\Sigma| \log |\Sigma|)$-space. Thus, while retaining an acceptable space complexity that is in general less than quadratic in the size of the state space, our algorithm improves the best known time bound.

Let us recall that a relation R between states is a simulation if for any $s, s' \in \Sigma$ such that $(s, s') \in R$, $\ell(s) = \ell(s')$ and for any $t \in \Sigma$ such that $s \rightarrow t$, there exists $t' \in \Sigma$ such that $s' \rightarrow t'$ and $(t, t') \in R$. Then, s' simulates s, namely the pair (s, s') belongs to the simulation preorder $R_{\rm sim}$, if there exists a simulation relation R such $(s, s') \in R$. Also, s and s' are simulation equivalent, namely they belong to the same block of the simulation partition $P_{\rm sim}$, if s' simulates s and vice versa.

Our simulation algorithm SA is designed as a modification of Henzinger, Henzinger and Kopke's [23] algorithm, here denoted by HHK. The space complexity of HHK is in $O(|\Sigma|^2 \log |\Sigma|)$. This is a consequence of the fact that HHK computes explicitly the simulation preorder, namely it maintains for any state $s \in \Sigma$ a set of states $Sim(s) \subseteq \Sigma$, called the simulator set of s, which stores states that are currently candidates for simulating s. Our algorithm SA computes instead a symbolic representation of the simulation preorder, namely it maintains: (i) a partition P of the state space Σ that is always coarser than the final simulation partition $P_{\rm sim}$ and (ii) a relation $Rel \subseteq P \times P$ on the current partition P that encodes the simulation relation between blocks of simulation equivalent states. This symbolic representation is the key both for obtaining the $O(|P_{\rm sim}||\rightarrow|)$ time bound and for limiting the space complexity of SA in $O(|P_{\rm sim}||\Sigma| \log |\Sigma|)$, so that memory requirements may be lower than quadratic in the size of the state space.

The basic idea of our approach is to investigate whether the logical structure of the HHK algorithm may be preserved by replacing the family of sets of states $\mathcal{S} = \{Sim(s)\}_{s \in \Sigma}$ with the following state partition P induced by $\mathcal{S}$: two states s_1 and s_2 are equivalent in P iff for all $s \in \Sigma$, $s_1 \in Sim(s) \Leftrightarrow s_2 \in Sim(s)$. Additionally, we store and maintain a preorder relation $Rel \subseteq P \times P$ on the partition P that gives rise to a so-called partition-relation pair $\langle P, Rel \rangle$. The logical meaning of this data structure is that if $B, C \in P$ and $(B, C) \in Rel$ then any state in C is currently candidate to simulate each state in B, while two states s_1 and s_2 in the same block B are currently candidates to be simulation equivalent. Hence, a partition-relation pair $\langle P, Rel \rangle$ represents the current approximation of the simulation preorder and in particular P represents the current approximation of simulation equivalence. It turns out that the information encoded by a partition-relation pair is enough for preserving the logical structure of HHK. In fact, analogously to

the stepwise design of the HHK procedure, this approach leads us to design a basic procedure $\mathrm{BasicSA}$ based on partition-relation pairs which is then refined twice in order to obtain the final simulation algorithm SA. The correctness of SA is proved w.r.t. the basic algorithm $\mathrm{BasicSA}$ and relies on abstract interpretation techniques [12, 13]. More specifically, we exploit some previous results [27] that show how standard strong preservation of temporal languages in abstract Kripke structures can be generalized by abstract interpretation and cast as a so-called completeness property of abstract domains. On the other hand, the simulation algorithm SA is designed as an efficient implementation of the basic procedure $\mathrm{BasicSA}$ where the symbolic representation based on partition-relation pairs allows us to replace the size $|\Sigma|$ of the state space in the time and space bounds of HHK with the size $|P_{\mathrm{sim}}|$ of the simulation partition in the corresponding bounds for SA.

Both HHK and SA have been implemented in C++. This practical evaluation considered benchmarks from the VLTS (Very Large Transition Systems) suite [30] and some publicly available Esterel programs. The experimental results showed that SA outperforms HHK.

2 Background

2.1 Preliminaries

Notations. Let X and Y be sets. If $S \subseteq X$ and X is understood as a universe set then $\neg S = X \smallsetminus S$. If $f : X \to Y$ then the image of f is denoted by $\mathrm{img}(f) = \{f(x) \in Y \mid x \in X\}$. When writing a set S of subsets of a given set of integers, e.g. a partition, S is often written in a compact form like $\{1, 12, 13\}$ or $\{[1], [12], [13]\}$ that stands for $\{\{1\}, \{1, 2\}, \{1, 3\}\}$. If $R \subseteq X \times X$ is any relation then $R^* \subseteq X \times X$ denotes the reflexive and transitive closure of R. Also, if $x \in X$ then $R(x) \stackrel{\text{def}}{=} \{x' \in X \mid (x, x') \in R\}$.

Orders. Let $\langle Q, \leq \rangle$ be a poset, that may also be denoted by $Q_{\leq}$. We use the symbol $\sqsubseteq$ to denote pointwise ordering between functions: If X is any set and $f, g : X \to Q$ then $f \sqsubseteq g$ if for all $x \in X$, $f(x) \leq g(x)$. If $S \subseteq Q$ then $\max(S) \stackrel{\text{def}}{=} \{x \in S \mid \forall y \in S.\ x \leq y \Rightarrow x = y\}$ denotes the set of maximal elements of S in Q. A complete lattice $C_{\leq}$ is also denoted by $\langle C, \leq, \vee, \wedge, \top, \bot \rangle$ where $\vee$, $\wedge$, $\top$ and $\bot$ denote, respectively, lub, glb, greatest element and least element in C. A function $f : C \to D$ between complete lattices is additive when f preserves least upper bounds. Let us recall that a reflexive and transitive relation $R \subseteq X \times X$ on a set X is called a preorder on X.

Partitions. A partition P of a set Σ is a set of nonempty subsets of Σ, called blocks, that are pairwise disjoint and whose union gives Σ. $\mathrm{Part}(\Sigma)$ denotes the set of partitions of Σ. If $P \in \mathrm{Part}(\Sigma)$ and $s \in \Sigma$ then $P(s)$ denotes the block of P that contains s. $\mathrm{Part}(\Sigma)$ is endowed with the following standard partial order $\preceq$: $P_1 \preceq P_2$, i.e. P_2 is coarser than P_1 (or P_1 refines P_2) iff $\forall B \in P_1 . \exists B' \in P_2.\ B \subseteq B'$. If $P_1, P_2 \in \mathrm{Part}(\Sigma)$, $P_1 \preceq P_2$ and $B \in P_1$ then $\mathrm{parent}_{P_2}(B)$ (when clear from the context the subscript P_2 may be omitted) denotes the unique block in P_2 that contains 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 the nonempty sets $B \cap S$ and $B \smallsetminus S$, where we also allow no splitting, namely $\mathit{Split}(P, S) = P$ (this happens exactly when S is a union of some blocks of P).

Kripke Structures. A transition system $(\Sigma, \rightarrow)$ consists of a set Σ of states and a transition relation $\rightarrow \subseteq \Sigma \times \Sigma$. The relation $\rightarrow$ is total when for any $s \in \Sigma$ there exists some $t \in \Sigma$ such that $s \rightarrow t$. The predecessor/successor transformers $\mathrm{pre}_{\rightarrow}, \mathrm{post}_{\rightarrow} : \wp(\Sigma) \to \wp(\Sigma)$ (when clear from the context the subscript $\rightarrow$ may be omitted) are defined as usual:

- $\mathrm{pre}_{\rightarrow}(Y) \stackrel{\text{def}}{=} \{a \in \Sigma \mid \exists b \in Y.\ a \rightarrow b\}$;
- $\mathrm{post}_{\rightarrow}(Y) \stackrel{\text{def}}{=} \{b \in \Sigma \mid \exists a \in Y.\ a \rightarrow b\}$.

Let us remark that $\mathrm{pre}_{\rightarrow}$ and $\mathrm{post}_{\rightarrow}$ are additive operators on the complete lattice $\wp(\Sigma)_{\subseteq}$. If $S_1, S_2 \subseteq \Sigma$ then $S_1 \rightarrow^{\exists\exists} S_2$ iff there exist $s_1 \in S_1$ and $s_2 \in S_2$ such that $s_1 \rightarrow s_2$.

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

2.2 Simulation Preorder and Equivalence

Recall that a relation $R \subseteq \Sigma \times \Sigma$ is a simulation on a Kripke structure $\mathcal{K} = (\Sigma, \rightarrow, \ell)$ over a set AP of atomic propositions if for any $s, s' \in \Sigma$ such that $(s, s') \in R$:

(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 $(t, t') \in R$.

If $(s, s') \in R$ then we say that s' simulates 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}. Simulation equivalence $\sim_{\mathrm{sim}} \subseteq \Sigma \times \Sigma$ is the symmetric reduction of R_{sim}, namely $\sim_{\mathrm{sim}} = R_{\mathrm{sim}} \cap R_{\mathrm{sim}}^{-1}$. $P_{\mathrm{sim}} \in \mathrm{Part}(\Sigma)$ denotes the partition corresponding to $\sim_{\mathrm{sim}}$ and is called simulation partition.

It is a well known result in model checking [14, 22, 25] that the reduction of $\mathcal{K}$ w.r.t. simulation equivalence $\sim_{\mathrm{sim}}$ allows us to define an abstract Kripke structure $\mathcal{A}_{\mathrm{sim}} = \langle P_{\mathrm{sim}}, \rightarrow^{\exists\exists}, \ell^{\exists} \rangle$ that strongly preserves the temporal language ACTL^*, where: P_{sim} is the abstract state space, $\rightarrow^{\exists\exists}$ is the abstract transition relation between simulation equivalence classes, while for any block $B \in P_{\mathrm{sim}}$, $\ell^{\exists}(B) \stackrel{\text{def}}{=} \ell(s)$ for any representative $s \in B$. It turns out that $\mathcal{A}_{\mathrm{sim}}$ strongly preserves ACTL^*, i.e., for any $\varphi \in \mathrm{ACTL}^*$, $B \in P_{\mathrm{sim}}$ and $s \in B$, we have that $s \models^{\mathcal{K}} \varphi$ if and only if $B \models^{\mathcal{A}_{\mathrm{sim}}} \varphi$.

2.3 Abstract Interpretation

Abstract Domains as Closures. In standard abstract interpretation, abstract domains can be equivalently specified either by Galois connections/insertions or by (upper) closure operators (uco's) [13]. These two approaches are equivalent, modulo isomorphic representations of domain's objects. We follow here the closure operator approach: this has the advantage of being independent from the representation of domain's objects and is therefore appropriate for reasoning on abstract domains independently from their representation.

Given a state space Σ, the complete lattice $\wp(\Sigma)_\subseteq$ plays the role of concrete domain. Let us recall that an operator $\mu : \wp(\Sigma) \rightarrow \wp(\Sigma)$ is a uco on $\wp(\Sigma)$, that is an abstract domain of $\wp(\Sigma)$, when μ is monotone, idempotent and extensive (viz., $X \subseteq \mu(X)$). It is well known that the set $\mathrm{uco}(\wp(\Sigma))$ of all uco's on $\wp(\Sigma)$, endowed with the pointwise ordering $\sqsubseteq$, gives rise to the complete lattice $\langle \mathrm{uco}(\wp(\Sigma)), \sqsubseteq, \sqcup, \sqcap, \lambda X.\Sigma, \mathrm{id} \rangle$ of all the abstract domains of $\wp(\Sigma)$. The pointwise ordering $\sqsubseteq$ on $\mathrm{uco}(\wp(\Sigma))$ is the standard order for comparing abstract domains with regard to their precision: $\mu_1 \sqsubseteq \mu_2$ means that the domain μ_1 is a more precise abstraction of $\wp(\Sigma)$ than μ_2, or, equivalently, that the abstract domain μ_1 is a refinement of μ_2.

A closure $\mu \in \mathrm{uco}(\wp(\Sigma))$ is uniquely determined by its image $\mathrm{img}(\mu)$, which coincides with its set of fixpoints, as follows: $\mu = \lambda Y. \cap \{X \in \mathrm{img}(\mu) \mid Y \subseteq X\}$. Also, a set of subsets $\mathcal{X} \subseteq \wp(\Sigma)$ is the image of some closure operator $\mu_{\mathcal{X}} \in \mathrm{uco}(\wp(\Sigma))$ iff $\mathcal{X}$ is a Moore-family of $\wp(\Sigma)$, i.e., $\mathcal{X} = \mathrm{Cl}_\cap(\mathcal{X}) \stackrel{\text{def}}{=} \{\cap S \mid S \subseteq \mathcal{X}\}$ (where $\cap\varnothing = \Sigma \in \mathrm{Cl}_\cap(\mathcal{X})$). In other terms, $\mathcal{X}$ is a Moore-family (or Moore-closed) when $\mathcal{X}$ is closed under arbitrary intersections. In this case, $\mu_{\mathcal{X}} = \lambda Y. \cap \{X \in \mathcal{X} \mid Y \subseteq X\}$ is the corresponding closure operator. For any $\mathcal{X} \subseteq \wp(\Sigma)$, $\mathrm{Cl}_\cap(\mathcal{X})$ is called the Moore-closure of $\mathcal{X}$, i.e., $\mathrm{Cl}_\cap(\mathcal{X})$ is the least set of subsets of Σ which contains all the subsets in $\mathcal{X}$ and is Moore-closed. Moreover, it turns out that for any $\mu \in \mathrm{uco}(\wp(\Sigma))$ and any Moore-family $\mathcal{X} \subseteq \wp(\Sigma)$, $\mu_{\mathrm{img}(\mu)} = \mu$ and $\mathrm{img}(\mu_{\mathcal{X}}) = \mathcal{X}$. Thus, closure operators on $\wp(\Sigma)$ are in bijection with Moore-families of $\wp(\Sigma)$. This allows us to consider a closure operator $\mu \in \mathrm{uco}(\wp(\Sigma))$ both as a function $\mu : \wp(\Sigma) \rightarrow \wp(\Sigma)$ and as a Moore-family $\mathrm{img}(\mu) \subseteq \wp(\Sigma)$. This is particularly useful and does not give rise to ambiguity since one can distinguish the use of a closure μ as function or set according to the context.

Abstract Domains and Partitions. As shown in [27], it turns out that partitions can be viewed as particular abstract domains. Let us recall here that any abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ induces a partition $\mathrm{par}(\mu) \in \mathrm{Part}(\Sigma)$ that corresponds to the following equivalence relation $\equiv_\mu$ on Σ:

$$x \equiv_\mu y \text{ iff } \mu(\{x\}) = \mu(\{y\}).$$

Example 2.1. Let $\Sigma = \{1,2,3,4\}$ and consider the following abstract domains in $\mathrm{uco}(\wp(\Sigma))$ that are given as intersection-closed subsets of $\wp(\Sigma)$: $\mu = \{\varnothing, 3, 4, 12, 34, 1234\}$, $\mu' = \{\varnothing, 3, 4, 12, 1234\}$, $\mu'' = \{12, 123, 124, 1234\}$. These abstract domains all induce the same partition $P = \{[12], [3], [4]\} \in \mathrm{Part}(\Sigma)$. For example, $\mu''(\{1\}) = \mu''(\{2\}) = \{1,2\}$, $\mu''(\{3\}) = \{1,2,3\}$, $\mu''(\{4\}) = \{1,2,4\}$ so that $\mathrm{par}(\mu'') = P$. □

Forward Completeness. Let us consider an abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma)_\subseteq)$, a concrete semantic function $f : \wp(\Sigma) \to \wp(\Sigma)$ and a corresponding abstract semantic function $f^\sharp : \mu \to \mu$ (for simplicity of notation, we consider 1-ary functions). It is well known that the abstract interpretation $\langle \mu, f^\sharp \rangle$ is sound when $f \circ \mu \sqsubseteq f^\sharp \circ \mu$ holds: this means that a concrete computation $f(\mu(X))$ on an abstract object $\mu(X)$ is correctly approximated in μ by $f^\sharp(\mu(X))$, that is, $f(\mu(X)) \subseteq f^\sharp(\mu(X))$. Forward completeness corresponds to require the following strengthening of soundness: $\langle \mu, f^\sharp \rangle$ is forward complete when $f \circ \mu = f^\sharp \circ \mu$: The intuition here is that the abstract function $f^\sharp$ is able to mimic f on the abstract domain μ with no loss of precision. This is called forward completeness because a dual and more standard notion of backward completeness may also be considered (see e.g. [19]).

Example 2.2. As a toy example, let us consider the following abstract domain $Sign$ for representing the sign of an integer variable: $Sign = \{\varnothing, \mathbb{Z}_{\leq 0}, 0, \mathbb{Z}_{\geq 0}, \mathbb{Z}\} \in \mathrm{uco}(\wp(\mathbb{Z})_\subseteq)$. The concrete pointwise addition $+ : \wp(\mathbb{Z}) \times \wp(\mathbb{Z}) \to \wp(\mathbb{Z})$ on sets of integers, that is $X + Y \stackrel{\text{def}}{=} \{x + y \mid x \in X,\ y \in Y\}$, is approximated in $Sign$ by the abstract addition $+^{Sign} : Sign \times Sign \to Sign$ that is defined as expected by the following table:

$+^{Sign}$	$\varnothing$	$\mathbb{Z}_{\leq 0}$	0	$\mathbb{Z}_{\geq 0}$	$\mathbb{Z}$
$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
$\mathbb{Z}_{\leq 0}$	$\varnothing$	$\mathbb{Z}_{\leq 0}$	$\mathbb{Z}_{\leq 0}$	$\mathbb{Z}$	$\mathbb{Z}$
0	$\varnothing$	$\mathbb{Z}_{\leq 0}$	0	$\mathbb{Z}_{\geq 0}$	$\mathbb{Z}$
$\mathbb{Z}_{\geq 0}$	$\varnothing$	$\mathbb{Z}$	$\mathbb{Z}_{\geq 0}$	$\mathbb{Z}_{\geq 0}$	$\mathbb{Z}$
$\mathbb{Z}$	$\varnothing$	$\mathbb{Z}$	$\mathbb{Z}$	$\mathbb{Z}$	$\mathbb{Z}$

It turns out that $\langle Sign, +^{Sign} \rangle$ is forward complete, i.e., for any $a_1, a_2 \in Sign$, $a_1 + a_2 = a_1 +^{Sign} a_2$. □

It turns out that the possibility of defining a forward complete abstract interpretation on a given abstract domain μ does not depend on the choice of the abstract function $f^\sharp$ but depends only on the abstract domain μ. This means that if $\langle \mu, f^\sharp \rangle$ is forward complete then the abstract function $f^\sharp$ indeed coincides with the best correct approximation $\mu \circ f$ of the concrete function f on the abstract domain μ. Hence, for any abstract domain μ and abstract function $f^\sharp$, it turns out that $\langle \mu, f^\sharp \rangle$ is forward complete if and only if $\langle \mu, \mu \circ f \rangle$ is forward complete. This allows us to define the notion of forward completeness independently of abstract functions as follows: an abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ is forward complete for f (or forward f-complete) iff $f \circ \mu = \mu \circ f \circ \mu$. Let us remark that μ is forward f-complete iff the image $\mathrm{img}(\mu)$ is closed under applications of the concrete function f. If F is a set of concrete functions then μ is forward complete for F when μ is forward complete for all $f \in F$.

Forward Complete Shells. It turns out [19, 27] that any abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ can be refined to its forward F-complete shell, namely to the most abstract domain that is forward complete for F and refines μ. This forward F-complete shell of μ is thus defined as

$$\mathcal{S}_F(\mu) \stackrel{\text{def}}{=} \sqcup \{\rho \in \mathrm{uco}(\wp(\Sigma)) \mid \rho \sqsubseteq \mu,\ \rho \text{ is forward } F\text{-complete}\}.$$

Forward complete shells admit a constructive fixpoint characterization. Given $\mu \in \mathrm{uco}(\wp(\Sigma))$, consider the operator $F_\mu : \mathrm{uco}(\wp(\Sigma)) \to \mathrm{uco}(\wp(\Sigma))$ defined by

$$F_\mu(\rho) \stackrel{\text{def}}{=} \mathrm{Cl}_\cap(\mu \cup \{f(X) \mid f \in F,\ X \in \rho\}).$$

Thus, $F_\mu(\rho)$ refines the abstract domain μ by adding the images of ρ for all the functions in F. It turns out that F_μ is monotone and therefore admits the greatest fixpoint, denoted by $\mathrm{gfp}(F_\mu)$, which provides the forward F-complete shell of μ: $\mathcal{S}_F(\mu) = \mathrm{gfp}(F_\mu)$.

Disjunctive Abstract Domains. An abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ is disjunctive (or additive) when μ is additive and this happens exactly when the image $\mathrm{img}(\mu)$ is closed under arbitrary unions. Hence, a disjunctive abstract domain is completely determined by the image of μ on singletons because for any $X \subseteq \Sigma$, $\mu(X) = \cup_{x \in X} \mu(\{x\})$. The intuition is that a disjunctive abstract domain does not lose precision in approximating concrete set unions. We denote by $\mathrm{uco}^{\mathrm{d}}(\wp(\Sigma)) \subseteq \mathrm{uco}(\wp(\Sigma))$ the set of disjunctive abstract domains.

Given any abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$, it turns out [13, 20] that μ can be refined to its disjunctive completion μ^{d}: this is the most abstract disjunctive domain $\mu^{\mathrm{d}} \in \mathrm{uco}^{\mathrm{d}}(\wp(\Sigma))$ that refines μ. The disjunctive completion μ^{d} can be obtained by closing the image $\mathrm{img}(\mu)$ under arbitrary unions, namely $\mathrm{img}(\mu^{\mathrm{d}}) = \mathrm{Cl}_\cup(\mathrm{img}(\mu)) \stackrel{\mathrm{def}}{=} \{\cup \mathcal{S} \mid \mathcal{S} \subseteq \mathrm{img}(\mu)\}$, where $\cup\varnothing = \varnothing \in \mathrm{Cl}_\cup(\mathrm{img}(\mu))$.

It turns out that an abstract domain μ is disjunctive iff μ is forward complete for arbitrary concrete set unions, namely, μ is disjunctive iff for any $\{X_i\}_{i \in I} \subseteq \wp(\Sigma)$, $\cup_{i \in I} \mu(X_i) = \mu(\cup_{i \in I} \mu(X_i))$. Thus, when Σ is finite, the disjunctive completion μ^{d} of μ coincides with the forward $\cup$-complete shell $\mathcal{S}_\cup(\mu)$ of μ. Also, since the predecessor transformer $\mathrm{pre}_\rightarrow$ preserves set unions, it turns out that the forward complete shell $\mathcal{S}_{\cup,\mathrm{pre}_\rightarrow}(\mu)$ for $\{\cup, \mathrm{pre}_\rightarrow\}$ can be obtained by iteratively closing the image of μ under $\mathrm{pre}_\rightarrow$ and then by taking the disjunctive completion, i.e., $\mathcal{S}_{\cup,\mathrm{pre}_\rightarrow}(\mu) = \mathcal{S}_\cup(\mathcal{S}_{\mathrm{pre}_\rightarrow}(\mu))$.

Example 2.3. Let us consider the abstract domain $\mu = \{\varnothing, 3, 4, 12, 34, 1234\}$ in Example 2.1. We have that μ is not disjunctive because $12, 3 \in \mu$ while $12 \cup 3 = 123 \notin \mu$. The disjunctive completion μ^{d} is obtained by closing μ under unions: $\mu^{\mathrm{d}} = \{\varnothing, 3, 4, 12, 34, 123, 124, 1234\}$. □

Some Properties of Abstract Domains. Let us summarize some easy properties of abstract domains that will be used in later proofs.

Lemma 2.4. *Let $\mu \in \mathrm{uco}(\wp(\Sigma))$, $\rho \in \mathrm{uco}^{\mathrm{d}}(\wp(\Sigma))$, $P, Q \in \mathrm{Part}(\Sigma)$ such that $P \preceq \mathrm{par}(\mu)$ and $Q \preceq \mathrm{par}(\rho)$.*

(i) *For any $B \in P$, $\mu(B) = \mu(\mathrm{parent}_{\mathrm{par}(\mu)}(B))$.*

(ii) *For any $X \in \wp(\Sigma)$, $\mu(X) = \cup\{B \in P \mid B \subseteq \mu(X)\}$.*

(iii) *For any $X \in \wp(\Sigma)$, $\rho(X) = \cup\{\rho(B) \mid B \in Q,\ B \cap X \neq \varnothing\}$.*

(iv) $\mathrm{par}(\mu) = \mathrm{par}(\mu^{\mathrm{d}})$.

Proof. (i) In general, by definition of $\mathrm{par}(\mu)$, for any $C \in \mathrm{par}(\mu)$ and $S \subseteq C$, $\mu(S) = \mu(C)$. Hence, since $B \subseteq \mathrm{parent}_{\mathrm{par}(\mu)}(B)$ we have that $\mu(B) = \mu(\mathrm{parent}_{\mathrm{par}(\mu)}(B))$.
(ii) Clearly, $\mu(X) \supseteq \cup\{B \in P \mid B \subseteq \mu(X)\}$. On the other hand, given $z \in \mu(X)$, let $B_z \in P$ be the block in P that contains z. Then, $B_z \subseteq \mu(B_z) = \mu(\{z\}) \subseteq \mu(X)$, so that $z \in \cup\{B \in P \mid B \subseteq \mu(X)\}$.
(iii)

$$\begin{aligned}
\rho(X) = & \quad \text{[as } \rho \text{ is additive]} \\
\cup\{\rho(\{x\}) \mid x \in X\} = & \quad \text{[as } Q \preceq \mathrm{par}(\rho)\text{]} \\
\cup\{\rho(B_x) \mid x \in X,\ B_x \in Q,\ x \in B_x\} = & \\
\cup\{\rho(B) \mid B \in Q,\ B \cap X \neq \varnothing\}. &
\end{aligned}$$

(iv) Since $\mu^{\mathrm{d}} \sqsubseteq \mu$, we have that $\mathrm{par}(\mu^{\mathrm{d}}) \preceq \mathrm{par}(\mu)$. On the other hand, if $B \in \mathrm{par}(\mu)$ then for all $x \in B$, $\mu^{\mathrm{d}}(\{x\}) = \mu(\{x\}) = \mu(B)$, so that $B \in \mathrm{par}(\mu^{\mathrm{d}})$. □

3 Simulation Preorder as a Forward Complete Shell

Ranzato and Tapparo [27] showed how strong preservation of specification languages in standard abstract models like abstract Kripke structures can be generalized by abstract interpretation and cast as a forward completeness property of generic abstract domains that play the role of abstract models. We rely here on this framework in order to show that the simulation preorder can be characterized as a forward complete shell for set union and the predecessor transformer. Let $\mathcal{K} = (\Sigma, \rightarrow, \ell)$ be a Kripke structure. Recall that the labeling function ℓ induces the state partition $P_\ell = \{[s]_\ell \mid s \in \Sigma\}$. This partition can be made an abstract domain $\mu_\ell \in \mathrm{uco}(\wp(\Sigma))$ by considering the Moore-closure of P_ℓ that simply adds to P_ℓ the empty set and the whole state space, namely $\mu_\ell \stackrel{\text{def}}{=} \mathrm{Cl}_\cap(\{[s]_\ell \mid s \in \Sigma\})$.

Theorem 3.1. *Let $\mu_\mathcal{K} = \mathcal{S}_{\cup,\mathrm{pre}}(\mu_\ell)$ be the forward $\{\cup, \mathrm{pre}\}$-complete shell of μ_ℓ. Then, $R_{\mathrm{sim}} = \{(s, s') \in \Sigma \times \Sigma \mid s' \in \mu_\mathcal{K}(\{s\})$ and $P_{\mathrm{sim}} = \mathrm{par}(\mu_\mathcal{K})$.*

Proof. Given a disjunctive abstract domain $\mu \in \mathrm{uco}^{\mathrm{d}}(\wp(\Sigma))$, define $R_\mu \stackrel{\text{def}}{=} \{(s, s') \in \Sigma \times \Sigma \mid s' \in \mu(\{s\})\}$. We prove the following three preliminary facts:

(1) μ is forward complete for pre iff R_μ satisfies the following property: for any $s, t, s' \in \Sigma$ such that $s \rightarrow t$ and $(s, s') \in R_\mu$ there exists $t' \in \Sigma$ such that $s' \rightarrow t'$ and $(t, t') \in R_\mu$. Observe that the disjunctive closure μ is forward complete for pre iff for any $s, t \in \Sigma$, if $s \in \mathrm{pre}(\mu(\{t\}))$ then $\mu(\{s\}) \subseteq \mathrm{pre}(\mu(\{t\}))$, and this happens iff for any $s, t \in \Sigma$, if $s \in \mathrm{pre}(\{t\})$ then $\mu(\{s\}) \subseteq \mathrm{pre}(\mu(\{t\}))$. This latter statement is equivalent to the fact that for any $s, s', t \in \Sigma$ such that $s \rightarrow t$ and $s' \in \mu(\{s\})$, there exists $t' \in \mu(\{t\})$ such that $s' \rightarrow t'$, namely, for any $s, s', t \in \Sigma$ such that $s \rightarrow t$ and $(s, s') \in R_\mu$, there exists $t' \in \Sigma$ such that $(t, t') \in R_\mu$ and $s' \rightarrow t'$.

(2) $\mu \sqsubseteq \mu_\ell$ iff R_μ satisfies the property that for any $s, s' \in \Sigma$, if $(s, s') \in R_\mu$ then $\ell(s) = \ell(s')$: In fact, $\mu \sqsubseteq \mu_\ell \Leftrightarrow \forall s \in \Sigma.\ \mu(\{s\}) \subseteq \mu_\ell(\{s\}) = [s]_\ell \Leftrightarrow \forall s, s' \in \Sigma.\ (s' \in \mu(\{s\})$ implies $s' \in [s]_\ell) \Leftrightarrow \forall s, s' \in \Sigma.\ ((s, s') \in R_\mu$ implies $\ell(s) = \ell(s'))$.

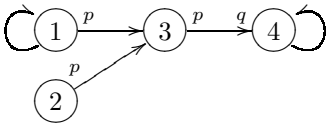
(3) Clearly, given $\mu' \in \mathrm{uco}^{\mathrm{d}}(\wp(\Sigma))$, $\mu \sqsubseteq \mu'$ iff $R_\mu \subseteq R_{\mu'}$.

Let us show that $R_{\mu_\mathcal{K}} = R_{\mathrm{sim}}$. By definition, $\mu_\mathcal{K}$ is the most abstract disjunctive closure that is forward complete for pre and refines μ_ℓ. Thus, by the above points (1) and (2), it turns out that $R_{\mu_\mathcal{K}}$ is a simulation on $\mathcal{K}$. Consider now any simulation S on $\mathcal{K}$ and the function $\mu' \stackrel{\text{def}}{=} \mathrm{post}_{S^*} : \wp(\Sigma) \rightarrow \wp(\Sigma)$. Let us notice that $\mu' \in \mathrm{uco}^{\mathrm{d}}(\wp(\Sigma))$ and $S \subseteq S^* = R_{\mu'}$. Also, the relation S^* is a simulation because S is a simulation. Since S^* is a simulation, we have that $R_{\mu'}$ satisfies the conditions of the above points (1) and (2) so that μ' is forward complete for pre and $\mu' \sqsubseteq \mu_\ell$. Moreover, μ' is disjunctive so that μ' is also forward complete for $\cup$. Thus, $\mu' \sqsubseteq \mathcal{S}_{\cup,\mathrm{pre}}(\mu_\ell) = \mu_\mathcal{K}$. Hence, by point (3) above, $R_{\mu'} \subseteq R_{\mu_\mathcal{K}}$ so that $S \subseteq R_{\mu_\mathcal{K}}$. We have therefore shown that $R_{\mu_\mathcal{K}}$ is the largest simulation on $\mathcal{K}$.
The fact that $P_{\mathrm{sim}} = \mathrm{par}(\mu_\mathcal{K})$ comes as a direct consequence because for any $s, t \in \Sigma$, $s \sim_{\mathrm{sim}} t$ iff $(s, t) \in R_{\mathrm{sim}}$ and $(t, s) \in R_{\mathrm{sim}}$. From $R_{\mu_\mathcal{K}} = R_{\mathrm{sim}}$ we obtain that $s \sim_{\mathrm{sim}} t$ iff $s \in \mu_\mathcal{K}(\{t\})$ and $t \in \mu_\mathcal{K}(\{s\})$ iff $\mu_\mathcal{K}(\{s\}) = \mu_\mathcal{K}(\{t\})$. This holds iff s and t belong to the same block in $\mathrm{par}(\mu_\mathcal{K})$. □

Thus, the simulation preorder is characterized as the forward complete shell of an initial abstract domain μ_ℓ induced by the labeling ℓ w.r.t. set union $\cup$ and the predecessor transformer pre while simulation equivalence is the partition induced by this forward complete shell. Let us observe that set union and the predecessor pre provide the semantics of, respectively, logical disjunction and the existential next operator EX. As shown in [27], simulation equivalence can be also characterized in a precise meaning as the most abstract domain that strongly preserves the language

$$\varphi ::= atom \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \mathrm{EX}\varphi.$$

Example 3.2. Let us consider the Kripke structure $\mathcal{K}$ depicted below where the atoms p and q determine the labeling function ℓ.



It is simple to observe that $P_{\rm sim} = \{1,2,3,4\}$ because: (i) while $3{\rightarrow}4$ we have that $1,2 \notin \mathrm{pre}(4)$ so that 1 and 2 are not simulation equivalent to 3; (ii) while $1{\rightarrow}1$ we have that $2 \notin \mathrm{pre}(12)$ so that 1 is not simulation equivalent to 2.
The abstract domain induced by the labeling is $\mu_\ell = \{\varnothing, 4, 123, 1234\} \in \mathrm{uco}(\wp(\Sigma))$. As observed above, the forward complete shell $\mathcal{S}_{\cup,\mathrm{pre}}(\mu_\ell) = \mathcal{S}_{\cup}(\mathcal{S}_{\mathrm{pre}}(\mu_\ell))$ so that this domain can be obtained by iteratively closing the image of μ_ℓ under pre and then by taking the disjunctive completion:

- $\mu_0 = \mu_\ell$;
- $\mu_1 = \mathrm{Cl}_\cap(\mu_0 \cup \mathrm{pre}(\mu_0)) = \mathrm{Cl}_\cap(\mu_0 \cup \{\mathrm{pre}(\varnothing) = \varnothing,\ \mathrm{pre}(4) = 34,\ \mathrm{pre}(123) = 12,\ \mathrm{pre}(1234) = 1234\}) = \{\varnothing, 3, 4, 12, 34, 123, 1234\}$;
- $\mu_2 = \mathrm{Cl}_\cap(\mu_1 \cup \mathrm{pre}(\mu_1)) = \mathrm{Cl}_\cap(\mu_1 \cup \{\mathrm{pre}(3) = 12,\ \mathrm{pre}(12) = 1,\ \mathrm{pre}(34) = 1234\}) = \{\varnothing, 1, 3, 4, 12, 34, 123, 1234\}$;
- $\mu_3 = \mathrm{Cl}_\cap(\mu_2 \cup \mathrm{pre}(\mu_2)) = \mu_2$ (fixpoint).

$\mathcal{S}_{\cup,\mathrm{pre}}(\mu_\ell)$ is thus given by the disjunctive completion of μ_2, i.e., $\mathcal{S}_{\cup,\mathrm{pre}}(\mu_\ell) = \{\varnothing, 1, 3, 4, 12, 13, 14, 34, 123, 124, 134, 1234\} = \mu_{\mathcal{K}}$. Note that $\mu_{\mathcal{K}}(1) = 1$, $\mu_{\mathcal{K}}(2) = 12$, $\mu_{\mathcal{K}}(3) = 3$ and $\mu_{\mathcal{K}}(4) = 4$. Hence, by Theorem 3.1, the simulation preorder is $R_{\rm sim} = \{(1,1),(2,2),(2,1),(3,3),(4,4)\}$, while $P_{\rm sim} = \mathrm{par}(\mathcal{S}_{\cup,\mathrm{pre}}(\mu_\ell)) = \{1,2,3,4\}$. □

Theorem 3.1 is one key result for proving the correctness of our simulation algorithm SA while it is not needed for understanding how SA works and how to implement it efficiently.

4 Partition-Relation Pairs

Let $P \in \mathrm{Part}(\Sigma)$ and $R \subseteq P \times P$ be any relation on the partition P. One such pair $\langle P, R\rangle$ is called a *partition-relation pair*. A partition-relation pair $\langle P, R\rangle$ induces a disjunctive closure $\mu_{\langle P,R\rangle} \in \mathrm{uco}^{\mathrm{d}}(\wp(\Sigma)_\subseteq)$ as follows: for any $X \in \wp(\Sigma)$,

$$\mu_{\langle P,R\rangle}(X) \stackrel{\mathrm{def}}{=} \cup \{C \in P \mid \exists B \in P.\, B \cap X \neq \varnothing, (B,C) \in R^*\}.$$

It is easily shown that $\mu_{\langle P,R\rangle}$ is indeed a disjunctive uco. Note that, for any $B \in P$ and $x \in B$,

$$\mu_{\langle P,R\rangle}(\{x\}) = \mu_{\langle P,R\rangle}(B) = \cup R^*(B) = \cup\{C \in P \mid (B,C) \in R^*\}.$$

This correspondence is a key logical point for proving the correctness of our simulation algorithm. In fact, our algorithm maintains a partition-relation pair, where the relation is a preorder, and our proof of correctness depends on the fact that this partition-relation pair logically represents a corresponding disjunctive abstract domain.

Example 4.1. Let $\Sigma = \{1,2,3,4\}$, $P = \{12,3,4\} \in \mathrm{Part}(\Sigma)$ and $R = \{(12,3),(3,4),(4,3)\}$. Note that $R^* = \{(12,12),(12,3),(12,4),(3,3),(3,4),(4,3),(4,4)\}$. The disjunctive abstract domain $\mu_{\langle P,R\rangle}$ is such that $\mu_{\langle P,R\rangle}(\{1\}) = \mu_{\langle P,R\rangle}(\{2\}) = \{1,2,3,4\}$ and $\mu_{\langle P,R\rangle}(\{3\}) = \mu_{\langle P,R\rangle}(\{4\}) = \{3,4\}$, so that the image of $\mu_{\langle P,R\rangle}$ is $\{\varnothing, 34, 1234\}$. □

On the other hand, any abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ induces a partition-relation pair $\langle P_\mu, R_\mu\rangle$ as follows:

- $P_\mu \stackrel{\mathrm{def}}{=} \mathrm{par}(\mu)$;
- $R_\mu \stackrel{\mathrm{def}}{=} \{(B,C) \in P_\mu \times P_\mu \mid C \subseteq \mu(B)\}$.

The following properties of partition-relation pairs will be useful in later proofs.

Lemma 4.2. *Let $\langle P, R\rangle$ be a partition-relation pair and $\mu \in \mathrm{uco}(\wp(\Sigma))$.*

(i) $P \preceq \mathrm{par}(\mu_{\langle P,R\rangle})$.

(ii) $\langle P_\mu, R_\mu\rangle = \langle P_{\mu^{\mathrm{d}}}, R_{\mu^{\mathrm{d}}}\rangle$.

Proof. (i) We already observed above that if $B \in P$ and $x \in B$ then $\mu_{\langle P,R\rangle}(\{x\}) = \mu_{\langle P,R\rangle}(B)$, so that $B \subseteq \{y \in \Sigma \mid \mu_{\langle P,R\rangle}(\{x\}) = \mu_{\langle P,R\rangle}(\{y\})\}$ which is a block in $\mathrm{par}(\mu_{\langle P,R\rangle})$.
(ii) By Lemma 2.4 (iv), $P_\mu = \mathrm{par}(\mu) = \mathrm{par}(\mu^{\mathrm{d}}) = P_{\mu^{\mathrm{d}}}$. Moreover,

$$\begin{aligned}
R_\mu = & \quad \text{[by definition]}\\
\{(B,C) \in P_\mu \times P_\mu \mid C \subseteq \mu(B)\} = & \quad \text{[as } P_\mu = P_{\mu^{\mathrm{d}}}\text{]}\\
\{(B,C) \in P_{\mu^{\mathrm{d}}} \times P_{\mu^{\mathrm{d}}} \mid C \subseteq \mu(B)\} = & \quad \text{[as } \mu(B) = \mu^{\mathrm{d}}(B)\text{]}\\
\{(B,C) \in P_{\mu^{\mathrm{d}}} \times P_{\mu^{\mathrm{d}}} \mid C \subseteq \mu^{\mathrm{d}}(B)\} = & \quad \text{[by definition]}\\
R_{\mu^{\mathrm{d}}}. &
\end{aligned}$$

□

It turns out that the above two correspondences between partition-relation pairs and disjunctive abstract domains are inverse of each other when the relation is a partial order.

Lemma 4.3. *For any partition $P \in \mathrm{Part}(\Sigma)$, partial order $R \subseteq P \times P$ and disjunctive abstract domain $\mu \in \mathrm{uco}^{\mathrm{d}}(\wp(\Sigma))$, we have that $\langle P_{\mu_{\langle P,R\rangle}}, R_{\mu_{\langle P,R\rangle}}\rangle = \langle P, R\rangle$ and $\mu_{\langle P_\mu, R_\mu\rangle} = \mu$.*

Proof. Let us show that $\langle P_{\mu_{\langle P,R\rangle}}, R_{\mu_{\langle P,R\rangle}}\rangle = \langle P, R\rangle$. We first prove that $P_{\mu_{\langle P,R\rangle}} = P$, i.e. $\mathrm{par}(\mu_{\langle P,R\rangle}) = P$. On the one hand, by Lemma 4.2 (i), $P \preceq \mathrm{par}(\mu_{\langle P,R\rangle})$. On the other hand, if $x, y \in \Sigma$, $\mu_{\langle P,R\rangle}(\{x\}) = \mu_{\langle P,R\rangle}(\{y\})$ and $x \in B_x \in P$ and $y \in B_y \in P$ then $(B_x, B_y) \in R^*$ and $(B_y, B_x) \in R^*$. Since R is a partial order, we have that $R^* = R$ is a partial order as well, so that $B_x = B_y$, namely $\mathrm{par}(\mu_{\langle P,R\rangle}) \preceq P$. Let us prove now that $R_{\mu_{\langle P,R\rangle}} = R$. In fact, for any $(B,C) \in \mathrm{par}(\mu_{\langle P,R\rangle}) \times \mathrm{par}(\mu_{\langle P,R\rangle})$,

$$\begin{aligned}
(B,C) \in R_{\mu_{\langle P,R\rangle}} \Leftrightarrow & \quad \text{[by definition of } R_{\mu_{\langle P,R\rangle}}\text{]}\\
C \subseteq \mu_{\langle P,R\rangle}(B) \Leftrightarrow & \quad \text{[by definition of } \mu_{\langle P,R\rangle}\text{]}\\
(B,C) \in R^* \Leftrightarrow & \quad \text{[since } R^* = R\text{]}\\
(B,C) \in R.
\end{aligned}$$

Finally, let us show that $\mu_{\langle P_\mu, R_\mu\rangle} = \mu$. Since both $\mu_{\langle P_\mu, R_\mu\rangle}$ and μ are disjunctive it is enough to prove that for all $x \in \Sigma$, $\mu_{\langle P_\mu, R_\mu\rangle}(\{x\}) = \mu(\{x\})$. Given $x \in \Sigma$ consider the block $B_x \in P_\mu = \mathrm{par}(\mu)$ containing x. Then,

$$\begin{aligned}
\mu_{\langle P_\mu, R_\mu\rangle}(\{x\}) = & \quad \text{[by definition of } \mu_{\langle P_\mu, R_\mu\rangle}\text{]}\\
\cup\{C \in P_\mu \mid (B_x, C) \in R_\mu^*\} = & \quad \text{[since } R_\mu^* = R_\mu\text{]}\\
\cup\{C \in P_\mu \mid (B_x, C) \in R_\mu\} = & \quad \text{[by definition of } R_\mu\text{]}\\
\cup\{C \in P_\mu \mid C \subseteq \mu(B_x)\} = & \quad \text{[by Lemma 2.4 (ii)]}\\
\mu(B_x) = & \quad \text{[since } \mu(B_x) = \mu(\{x\})\text{]}\\
\mu(\{x\}). &
\end{aligned}$$

□

Our simulation algorithm relies on the following condition on a partition-relation pair $\langle P, R\rangle$ w.r.t. a transition system $(\Sigma, \to)$ which guarantees that the corresponding disjunctive abstract domain $\mu_{\langle P,R\rangle}$ is forward complete for the predecessor pre.

Lemma 4.4. *Let $(\Sigma, \to)$ be a transition system and $\langle P, R\rangle$ be a partition-relation pair where R is reflexive. Assume that for any $B, C \in P$, if $C \cap \mathrm{pre}(B) \neq \varnothing$ then $\cup R(C) \subseteq \mathrm{pre}(\cup R(B))$. Then, $\mu_{\langle P,R\rangle}$ is forward complete for* pre.

Proof. We preliminarily show the following fact:

(‡) Let $\mu \in \mathrm{uco}^{\mathrm{d}}(\wp(\Sigma))$ and $P \in \mathrm{Part}(\Sigma)$ such that $P \preceq \mathrm{par}(\mu)$. Then, μ is forward complete for pre iff for any $B, C \in P$, if $C \cap \mathrm{pre}(B) \neq \varnothing$ then $\mu(C) \subseteq \mathrm{pre}(\mu(B))$.

($\Rightarrow$) Let $B, C \in P$ such that $C \cap \mathrm{pre}(B) \neq \varnothing$. Since $B \subseteq \mu(B)$ we also have that $C \cap \mathrm{pre}(\mu(B)) \neq \varnothing$. By forward completeness, $\mathrm{pre}(\mu(B)) = \mu(\mathrm{pre}(\mu(B))$. Since $P \preceq \mathrm{par}(\mu)$, $C \in P$ and $C \cap \mu(\mathrm{pre}(\mu(B))) = C \cap \mathrm{pre}(\mu(B)) \neq \varnothing$ we have that $C \subseteq \mu(\mathrm{pre}(\mu(B))) = \mathrm{pre}(\mu(B))$, so that, by applying the monotone map μ, $\mu(C) \subseteq \mu(\mathrm{pre}(\mu(B))) = \mathrm{pre}(\mu(B))$.

($\Leftarrow$) Firstly, we show the following property $(*)$: for any $B, C \in P$, if $C \cap \mathrm{pre}(\mu(B)) \neq \varnothing$ then $\mu(C) \subseteq \mathrm{pre}(\mu(B))$. Since $P \preceq \mathrm{par}(\mu)$, by Lemma 2.4 (ii), $C \cap \mathrm{pre}(\mu(B)) = C \cap \mathrm{pre}(\cup\{D \in P \mid D \subseteq \mu(B)\})$, so that if $C \cap \mathrm{pre}(\mu(B)) \neq \varnothing$ then $C \cap \mathrm{pre}(D) \neq \varnothing$ for some $D \in P$ such that $D \subseteq \mu(B)$. Hence, by hypothesis, $\mu(C) \subseteq \mathrm{pre}(\mu(D))$. Since $\mu(D) \subseteq \mu(B)$, we thus obtain that $\mu(C) \subseteq \mathrm{pre}(\mu(D)) \subseteq \mathrm{pre}(\mu(B))$. Let us now prove that μ is forward complete for pre. We first show the following property $(**)$: for any $B \in P$, $\mu(\mathrm{pre}(\mu(B))) \subseteq \mathrm{pre}(\mu(B))$. In fact, since $P \preceq \mathrm{par}(\mu)$, we have that:

$$\begin{array}{rl} \mu(\mathrm{pre}(\mu(B))) = & \text{[by Lemma 2.4 (iii) because } \mu \text{ is additive]} \\ \cup\{\mu(C) \mid C \in P,\ C \cap \mathrm{pre}(\mu(B)) \neq \varnothing\} \subseteq & \text{[by the above property } (*)\text{]} \\ \mathrm{pre}(\mu(B)). & \end{array}$$

Hence, for any $X \in \wp(\Sigma)$, we have that:

$$\begin{array}{rl} \mu(\mathrm{pre}(\mu(X))) = & \text{[since, by Lemma 2.4 (iii), } \mu(X) = \cup_i \mu(B_i) \text{ for some } \{B_i\} \subseteq P\text{]} \\ \mu(\mathrm{pre}(\cup_i \mu(B_i))) = & \text{[since } \mu \text{ and } \mathrm{pre} \text{ are additive]} \\ \cup_i \mu(\mathrm{pre}(\mu(B_i))) \subseteq & \text{[by the above property } (**)\text{]} \\ \cup_i \mathrm{pre}(\mu(B_i)) = & \text{[since } \mathrm{pre} \text{ is additive]} \\ \mathrm{pre}(\cup_i \mu(B_i)) = & \text{[since } \mu(X) = \cup_i \mu(B_i)\text{]} \\ \mathrm{pre}(\mu(X)). & \end{array}$$

Let us now turn to show the lemma. By Lemma 4.2 (i), we have that $P \preceq \mathrm{par}(\mu_{\langle P,R\rangle})$. By the above fact $(\ddagger)$, in order to prove that $\mu_{\langle P,R\rangle}$ is forward complete for pre it is sufficient to show that for any $B, C \in P$, if $C \cap \mathrm{pre}(B) \neq \varnothing$ then $\mu_{\langle P,R\rangle}(C) \subseteq \mathrm{pre}(\mu_{\langle P,R\rangle}(B))$. Thus, assume that $C \cap \mathrm{pre}(B) \neq \varnothing$. We need to show that $\cup R^*(C) \subseteq \mathrm{pre}(\cup R^*(B))$. Assume that $(C, D) \in R^*$, namely that there exist $\{B_i\}_{i \in [0,k]} \subseteq P$, for some $k \geq 0$, such that $B_0 = C$, $B_k = D$ and for any $i \in [0, k)$, $(B_i, B_{i+1}) \in R$. We show by induction on k that $D \subseteq \mathrm{pre}(\cup R^*(B))$.

($k = 0$) This means that $C = D$. Since R is assumed to be reflexive, we have that $(C, C) \in R$. By hypothesis, $\cup R(C) \subseteq \mathrm{pre}(\cup R(B))$ so that we obtain $D = C \subseteq \cup R(C) \subseteq \mathrm{pre}(\cup R(B)) \subseteq \mathrm{pre}(\cup R^*(B))$.

($k + 1$) Assume that $(C, B_1), (B_1, B_2), ..., (B_k, D) \in R$. By inductive hypothesis, $B_k \subseteq \mathrm{pre}(\cup R^*(B))$. Note that, by additivity of pre, $\mathrm{pre}(\cup R^*(B)) = \cup\{\mathrm{pre}(E) \mid E \in P,\ (B, E) \in R^*\}$. Thus, there exists some $E \in P$ such that $(B, E) \in R^*$ and $B_k \cap \mathrm{pre}(E) \neq \varnothing$. Hence, by hypothesis, $\cup R(B_k) \subseteq \mathrm{pre}(\cup R(E))$. Observe that $\cup R(E) \subseteq \cup R^*(E) \subseteq \cup R^*(B)$ so that $D \subseteq \cup R(B_k) \subseteq \mathrm{pre}(\cup R(E)) \subseteq \mathrm{pre}(\cup R^*(B))$. □

5 Henzinger, Henzinger and Kopke's Algorithm

Our simulation algorithm SA is designed as a symbolic modification of Henzinger, Henzinger and Kopke's simulation algorithm [23]. This algorithm is designed in three incremental steps encoded by the procedures *SchematicSimilarity*, *RefinedSimilarity* and HHK (called *EfficientSimilarity* in [23]) in Figure 1.

Consider any (possibly non total) finite Kripke structure $(\Sigma, \rightarrow, \ell)$. The idea of the basic *SchematicSimilarity* algorithm is simple. For each state $v \in \Sigma$, the simulator set $Sim(v) \subseteq \Sigma$ contains states that are candidates for simulating v. Hence, $Sim(v)$ is initialized with all the states having the same labeling as v, that is $[v]_\ell$. The algorithm then proceeds iteratively as follows: if $u \rightarrow v$, $w \in Sim(u)$ but there is no $w' \in Sim(v)$ such that $w \rightarrow w'$ then w cannot simulate u and therefore $Sim(u)$ is refined to $Sim(u) \smallsetminus \{w\}$.

```
SchematicSimilarity() {
  forall v ∈ Σ do Sim(v) := [v]_ℓ;
  while ∃u, v, w ∈ Σ such that (u→v & w∈Sim(u) & post_→({w}) ∩ Sim(v) = ∅) do
    Sim(u) := Sim(u) ∖ {w};
}

RefinedSimilarity() {
  forall v ∈ Σ do
    prevSim(v) := Σ;
    if post({v}) = ∅ then Sim(v) := [v]_ℓ; else Sim(v) := [v]_ℓ ∩ pre(Σ);
  while ∃v ∈ Σ such that Sim(v) ≠ prevSim(v)) do
    // Inv_1:  ∀v ∈ Σ. Sim(v) ⊆ prevSim(v)
    // Inv_2:  ∀u, v ∈ Σ. u→v ⇒ Sim(u) ⊆ pre(prevSim(v))
    Remove := pre(prevSim(v)) ∖ pre(Sim(v));
    prevSim(v) := Sim(v);
    forall u ∈ pre(v) do Sim(u) := Sim(u) ∖ Remove;
}

HHK() {
  // forall v ∈ Σ do prevSim(v) := Σ;
  forall v ∈ Σ do
    if post({v}) = ∅ then Sim(v) := [v]_ℓ; else Sim(v) := [v]_ℓ ∩ pre(Σ);
    Remove(v) := pre(Σ) ∖ pre(Sim(v));
  while ∃v ∈ Σ such that Remove(v) ≠ ∅ do
    // Inv_3:  ∀v ∈ Σ. Remove(v) = pre(prevSim(v)) ∖ pre(Sim(v))
    // prevSim(v) := Sim(v);
    Remove := Remove(v);
    Remove(v) := ∅;
    forall u ∈ pre(v) do
      forall w ∈ Remove do
        if w ∈ Sim(u) then
          Sim(u) := Sim(u) ∖ {w};
          forall w'' ∈ pre(w) such that w'' ∉ pre(Sim(u) do
            Remove(u) := Remove(u) ∪ {w''};
}
```

Figure 1: HHK Algorithm.

This basic procedure is then refined to the algorithm *RefinedSimilarity*. The key point here is to store for each state $v \in \Sigma$ an additional set of states $prevSim(v)$ that is a superset of $Sim(v)$ (invariant Inv_1) and contains the states that were in $Sim(v)$ in some past iteration where v was selected. If $u{\rightarrow}v$ then the invariant Inv_2 allows to refine $Sim(u)$ by scrutinizing only the states in $\mathrm{pre}(prevSim(v))$ instead of all the possible states in Σ: In fact, while in *SchematicSimilarity*, $Sim(u)$ is reduced to $Sim(u) \smallsetminus (\Sigma \smallsetminus \mathrm{pre}(Sim(v))$, in *RefinedSimilarity*, $Sim(u)$ is reduced in the same way by removing from it the states in $Remove \stackrel{\mathrm{def}}{=} \mathrm{pre}(prevSim(v)) \smallsetminus \mathrm{pre}(Sim(v))$. The initialization of $Sim(v)$ that distinguishes the case $\mathrm{post}(\{v\}) = \varnothing$ allows to initially establish the invariant Inv_2. Let us remark that the original *RefinedSimilarity* algorithm presented in [23] contains the following bug: the statement $prevSim(v) := Sim(v)$ is placed just after the inner for-loop instead of immediately preceding the inner for-loop. It turns out that this is not correct as shown by the following example.

Example 5.1. Let us consider the Kripke structure in Example 3.2. We already observed that the simulation relation is $R_{\mathrm{sim}} = \{(1,1),(2,2),(2,1),(3,3),(4,4)\}$. However, one can check that the original version of the *RefinedSimilarity* algorithm in [23] — where the assignment $prevSim(v) := Sim(v)$ follows the inner for-loop — provides as output $Sim(1) = \{1,2\}$, $Sim(2) = \{1,2\}$, $Sim(3) = \{3\}$, $Sim(4) = \{4\}$, namely the state 2 appears to simulate the state 1 while this is not the case. The problem with the original version in [23] of the *RefinedSimilarity* algorithm lies in the fact that when $v \in \mathrm{pre}(\{v\})$ — like in this example for state 1 — it may happen that during the inner for-loop the set $Sim(v)$ is refined to $Sim(v) \smallsetminus Remove$ so that if the assignment $prevSim(v) := Sim(v)$ follows the inner for-loop then $prevSim(v)$ might be computed as an incorrect subset of the right set. □

RefinedSimilarity is further refined to the final HHK algorithm. The idea here is that instead of recomputing at each iteration of the while-loop the set $Remove := \mathrm{pre}(prevSim(v)) \smallsetminus \mathrm{pre}(Sim(v))$ for the selected state v, a set $Remove(v)$ is maintained and incrementally updated for each state $v \in \Sigma$ in such a way that it satisfies the invariant Inv_3. The original version of HHK in [23] also suffers from a bug that is a direct consequence of the problem in *RefinedSimilarity* described above: within the main while-loop of HHK, the statement $Remove(v) := \varnothing$ is placed just after the outermost for-loop instead of immediately preceding the outermost for-loop. It is easy to show that this is not correct by resorting again to Example 5.1.

The implementation of HHK exploits a matrix $Count(u,v)$, indexed on states $u, v \in \Sigma$, such that $Count(u,v) = |\mathrm{post}(u) \cap Sim(v)|$, i.e., $Count(u,v)$ stores the number of transitions from u to some state $w \in Sim(v)$. Hence, the test $w'' \notin \mathrm{pre}(Sim(u))$ in the innermost for-loop can be done in $O(1)$ by checking whether $Count(w'',u)$ is 0 or not. This provides an efficient implementation of HHK that runs in $O(|\Sigma||{\rightarrow}|)$ time, while the space complexity is in $O(|\Sigma|^2 \log|\Sigma|)$, namely it is more than quadratic in the size of the state space. Let us remark that the key property for showing the $O(|\Sigma||{\rightarrow}|)$ time bound is as follows: if a state v is selected at some iterations i and j of the while-loop and the iteration i precedes the iteration j then $Remove_i(v) \cap Remove_j(v) = \varnothing$, so that the sets in $\{Remove_i(v) \mid v \text{ is selected at some iteration i }\}$ are pairwise disjoints.

6 A New Simulation Algorithm

6.1 The Basic Algorithm

Let us consider any (possibly non total) finite Kripke structure $(\Sigma, {\rightarrow}, \ell)$. As recalled above, the HHK procedure maintains for each state $s \in \Sigma$ a simulator set $Sim(s) \subseteq \Sigma$ and a remove set $Remove(s) \subseteq \Sigma$. The simulation preorder R_{sim} is encoded by the output $\{Sim(s)\}_{s\in\Sigma}$ as follows: $(s,s') \in R_{\mathrm{sim}}$ iff $s' \in Sim(s)$. Hence, the simulation equivalence partition P_{sim} is obtained as follows: s and s' are simulation equivalent iff $s \in Sim(s')$ and $s' \in Sim(s)$. Our algorithm relies on the idea of modifying the HHK procedure in order to maintain a partition-relation pair $\langle P, Rel\rangle$ in place of $\{Sim(s)\}_{s\in\Sigma}$, together with a remove set $Remove(B) \subseteq \Sigma$ for each block $B \in P$. The basic idea is to replace the family of sets $\mathcal{S} = \{Sim(s)\}_{s\in\Sigma}$ with the following state partition P induced by $\mathcal{S}$: $s_1 \sim_{\mathcal{S}} s_2$ iff for all $s \in \Sigma$, $s_1 \in Sim(s) \Leftrightarrow s_2 \in Sim(s)$. Then, a reflexive relation $Rel \subseteq P \times P$ on P gives rise to a partition-relation pair where the intuition is as follows: given a state s and a block $B \in P$ (i) if $s \in B$ then the

```
1 BasicSA(PartitionRelation ⟨P, Rel⟩) {
2   while ∃B, C ∈ P such that (C ∩ pre(B) ≠ ∅ & ∪Rel(C) ⊈ pre(∪Rel(B))) do
3     S := pre(∪Rel(B));
4     P_prev := P; B_prev := B;
5     P := Split(P, S);
6     forall C ∈ P do Rel(C) := {D ∈ P | D ⊆ ∪Rel(parent_{P_prev}(C))};
7     forall C ∈ P such that C ∩ pre(B_prev) ≠ ∅ do Rel(C) := {D ∈ Rel(C) | D ⊆ S};
8 }
```

Figure 2: Basic Simulation Algorithm.

current simulator set for s is a the union of blocks in P that are in relation with B, i.e. $Sim(s) = \cup Rel(B)$; (ii) if $s, s' \in B$ then s and s' are currently candidates to be simulation equivalent. Thus, a partition-relation pair $\langle P, Rel\rangle$ represents the current approximation of the simulation preorder and in particular P represents the current approximation of simulation equivalence.

Partition-relation pairs have been used by Henzinger, Henzinger and Kopke's [23] to compute the simulation preorder on effectively presented infinite transition systems, notably hybrid automata. Henzinger et al. provide a symbolic procedure, called $SymbolicSimilarity$ in [23], that is derived as a symbolization through partition-relation pairs of their basic simulation algorithm $SchematicSimilarity$ in Figure 1. Moreover, partition-relation pairs are also exploited by Gentilini et al. [18] in their simulation algorithm for representing simulation relations. The distinctive feature of our use of partition-relation pairs is that, by relying on the results in Section 4, we logically view partition-relation pairs as abstract domains and therefore we can reason on them by using abstract interpretation.

Following Henzinger et al. [23], our simulation algorithm is designed in three incremental steps. We exploit the following results for designing the basic algorithm.

- Theorem 3.1 tells us that the simulation preorder can be obtained from the forward $\{\cup, \mathrm{pre}\}$-complete shell of an initial abstract domain μ_ℓ induced by the labeling ℓ.
- As shown in Section 4, a partition-relation pair can be viewed as representing a disjunctive abstract domain.
- Lemma 4.4 gives us a condition on a partition-relation pair which guarantees that the corresponding abstract domain is forward complete for pre. Moreover, this abstract domain is disjunctive as well, being induced by a partition-relation pair.

Thus, the idea consists in iteratively and minimally refining an initial partition-relation pair $\langle P, Rel\rangle$ induced by the labeling ℓ until the condition of Lemma 4.4 is satisfied: for all $B, C \in P$,

$$C \cap \mathrm{pre}(B) \neq \varnothing \;\Rightarrow\; \cup Rel(C) \subseteq \mathrm{pre}(\cup Rel(B)).$$

Let us observe that $C \cap \mathrm{pre}(B) \neq \varnothing$ means that $C \rightarrow^{\exists\exists} B$. The basic algorithm, called BasicSA, is in Figure 2. The current partition-relation pair $\langle P, Rel\rangle$ is refined by the following three steps in BasicSA. If B is the block of the current partition P selected by the while-loop then:

(i) the current partition P is split with respect to the set $S = \mathrm{pre}(\cup Rel(B))$;

(ii) if C is a newly generated block after splitting the current partition and $\mathrm{parent}_{P_{\mathrm{prev}}}(C)$ is its parent block in the partition P_{prev} before the splitting operation then $Rel(C)$ is modified so as that $\cup Rel(C) = \cup Rel(\mathrm{parent}_{P_{\mathrm{prev}}}(C))$;

(iii) the current relation Rel is refined for the (new and old) blocks C such that $C \rightarrow^{\exists\exists} B$ by removing from $Rel(C)$ those blocks that are not contained in S; observe that after having split P w.r.t. S it turns out that one such block D either is contained in S or is disjoint with S.

Let us remark that although the symbolic simulation algorithm for infinite graphs $SymbolicSimilarity$ in [23] may appear similar to our $\mathrm{BasicSA}$ algorithm, it is instead inherently different due to the following reason: the role played by the condition: $C{\rightarrow}^{\exists\exists}B \ \& \ \cup Rel(C) \not\subseteq \mathrm{pre}(\cup Rel(B))$ in the while-loop of $\mathrm{BasicSA}$ is played in $SymbolicSimilarity$ by: $C{\rightarrow}^{\exists\exists} \cup Rel(B) \ \& \ \cup Rel(C) \not\subseteq \mathrm{pre}(\cup Rel(B))$, and this latter condition is computationally harder to check.

The following correctness result formalizes that $\mathrm{BasicSA}$ can be viewed as an abstract domain refinement algorithm that allows us to compute forward complete shells for $\{\cup, \mathrm{pre}\}$. For any abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$, we write $\mu' = \mathrm{BasicSA}(\mu)$ when the algorithm $\mathrm{BasicSA}$ on an input partition-relation $\langle P_\mu, R_\mu\rangle$ terminates and outputs a partition-relation pair $\langle P', R'\rangle$ such that $\mu' = \mu_{\langle P', R'\rangle}$.

Theorem 6.1. *Let Σ be finite. Then,* $\mathrm{BasicSA}$ *terminates on any input domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ and* $\mathrm{BasicSA}(\mu) = \mathcal{S}_{\cup,\mathrm{pre}}(\mu)$.

Proof. Let $\langle P_{\mathrm{curr}}, R_{\mathrm{curr}}\rangle$ and $\langle P_{\mathrm{next}}, R_{\mathrm{next}}\rangle$ be, respectively, the current and next partition-relation pair in some iteration of $\mathrm{BasicSA}(\mu)$. By line 5, $P_{\mathrm{next}} \preceq P_{\mathrm{curr}}$ always holds. Moreover, if $P_{\mathrm{next}} = P_{\mathrm{curr}}$ then it turns out that $R_{\mathrm{next}} \subsetneq R_{\mathrm{curr}}$: in fact, if $B, C \in P_{\mathrm{curr}}$, $C \cap \mathrm{pre}(B) \neq \varnothing$ and $\cup R_{\mathrm{curr}}(C) \not\subseteq \mathrm{pre}(\cup R_{\mathrm{curr}}(B))$ then, by lines 6 and 7, $\cup R_{\mathrm{next}}(C) \subsetneq \cup R_{\mathrm{curr}}(C)$ because there exists $x \in \cup R_{\mathrm{curr}}(C)$ such that $x \notin \mathrm{pre}(\cup R_{\mathrm{curr}}(B))$ so that if $B_x \in P_{\mathrm{next}} = P_{\mathrm{curr}}$ is the block that contains x then $B_x \cap (\cup R_{\mathrm{next}}(C)) = \varnothing$ while $B_x \subseteq \cup R_{\mathrm{curr}}(C)$. Thus, either $P_{\mathrm{next}} \prec P_{\mathrm{curr}}$ or $R_{\mathrm{next}} \subsetneq R_{\mathrm{curr}}$, so that, since the state space Σ is finite, the procedure $\mathrm{BasicSA}$ terminates.
Let $\mu' = \mathrm{BasicSA}(\mu)$, namely, let $\mu' = \mu_{\langle P', R'\rangle}$ where $\langle P', R'\rangle$ is the output of $\mathrm{BasicSA}$ on input $\langle P_\mu, R_\mu\rangle$. Let $\{\langle P_i, R_i\rangle\}_{i\in[0,k]}$ be the sequence of partition-relation pairs computed by $\mathrm{BasicSA}$, where $\langle P_0, R_0\rangle = \langle P_\mu, R_\mu\rangle$ and $\langle P_k, R_k\rangle = \langle P', R'\rangle$. Let us first observe that for any $i \in [0, k)$, $P_{i+1} \preceq P_i$ because the current partition is refined by the splitting operation in line 5. Moreover, for any $i \in [0, k)$ and $C \in P_{i+1}$, note that $\cup R_{i+1}(C) \subseteq \cup R_i(\mathrm{parent}_{P_i}(C))$, because the current relation is modified only at lines 6 and 7.
Let us also observe that for any $i \in [0, k]$, R_i is a reflexive relation because R_0 is reflexive and the operations at lines 6-7 preserve the reflexivity of the current relation. Let us show this latter fact. If $C \in P_{\mathrm{next}}$ is such that $C \cap \mathrm{pre}(B_{\mathrm{prev}}) \neq \varnothing$ then because, by hypothesis, $B_{\mathrm{prev}} \in R_{\mathrm{prev}}(B_{\mathrm{prev}})$, we have that $C \cap \mathrm{pre}(\cup R_{\mathrm{prev}}(B_{\mathrm{prev}})) \neq \varnothing$ so that $C \subseteq S = \mathrm{pre}(\cup R_{\mathrm{prev}}(B_{\mathrm{prev}}))$. Hence, if $C \in P_{\mathrm{next}} \cap P_{\mathrm{prev}}$ then $C \in R_{\mathrm{next}}(C)$, while if $C \in P_{\mathrm{next}} \smallsetminus P_{\mathrm{prev}}$ then, by hypothesis, $\mathrm{parent}_{P_{\mathrm{prev}}}(C) \in R_{\mathrm{prev}}(\mathrm{parent}_{P_{\mathrm{prev}}}(C))$ so that, by line 6, $C \in R_{\mathrm{next}}(C)$ also in this case.
For any $B \in P' = P_k$, we have that

$$\begin{aligned}
\mu'(B) = & \quad \text{[by definition of } \mu'\text{]}\\
\cup R^*_k(B) \subseteq & \quad \text{[as } \cup R_k(B) \subseteq \cup R_0(\mathrm{parent}_{P_0}(B))\text{]}\\
\cup R^*_0(\mathrm{parent}_{P_0}(B)) = & \quad \text{[as } P_0 = \mathrm{par}(\mu) \text{ and } R^*_0 = R^*_\mu = R_\mu\text{]}\\
\cup R_\mu(\mathrm{parent}_{\mathrm{par}(\mu)}(B)) = & \quad \text{[by Lemma 4.2 (ii), } \langle \mathrm{par}(\mu), R_\mu\rangle = \langle \mathrm{par}(\mu^{\mathrm{d}}), R_{\mu^{\mathrm{d}}}\rangle\text{]}\\
\cup R_{\mu^{\mathrm{d}}}(\mathrm{parent}_{\mathrm{par}(\mu^{\mathrm{d}})}(B)) = & \quad \text{[by definition of } R_{\mu^{\mathrm{d}}}\text{]}\\
\cup\{C \in \mathrm{par}(\mu^{\mathrm{d}}) \mid C \subseteq \mu^{\mathrm{d}}(\mathrm{parent}_{\mathrm{par}(\mu^{\mathrm{d}})}(B))\} = & \quad \text{[by Lemma 2.4 (ii)]}\\
\mu^{\mathrm{d}}(\mathrm{parent}_{\mathrm{par}(\mu^{\mathrm{d}})}(B)) = & \quad \text{[by Lemma 2.4 (i)]}\\
\mu^{\mathrm{d}}(B). &
\end{aligned}$$

Thus, since, by Lemma 4.2 (i), $P' \preceq \mathrm{par}(\mu')$, by Lemma 2.4 (iv), $P' \preceq P_\mu = \mathrm{par}(\mu^{\mathrm{d}})$ and both μ' and μ^{d} are disjunctive, we have that for any $X \in \wp(\Sigma)$,

$$\begin{aligned}
\mu'(X) = & \quad \text{[by Lemma 2.4 (iii)]}\\
\cup\{\mu'(B) \mid B \in P',\ B \cap X \neq \varnothing\} \subseteq & \quad \text{[as } \mu'(B) \subseteq \mu^{\mathrm{d}}(B)\text{]}\\
\cup\{\mu^{\mathrm{d}}(B) \mid B \in P',\ B \cap X \neq \varnothing\} = & \quad \text{[by Lemma 2.4 (iii)]}\\
\mu^{\mathrm{d}}(X) \subseteq & \quad \text{[as } \mu^{\mathrm{d}} \sqsubseteq \mu\text{]}\\
\mu(X). &
\end{aligned}$$

Thus, μ' is a refinement of μ. We have that $P' \preceq \mathrm{par}(\mu')$, $R' = R_k$ is (as shown above) reflexive and because $\langle P', R' \rangle$ is the output partition-relation pair, for all $B, C \in P'$, if $C \cap \mathrm{pre}(B) \neq \varnothing$ then $\cup R'(C) \subseteq \mathrm{pre}(\cup R'(B))$. Hence, by Lemma 4.4 we obtain that μ' is forward complete for pre. Thus, μ' is a disjunctive refinement of μ that is forward complete for pre so that $\mu' \sqsubseteq \mathcal{S}_{\cup,\mathrm{pre}}(\mu)$.
In order to conclude the proof, let us show that $\mathcal{S}_{\cup,\mathrm{pre}}(\mu) \sqsubseteq \mu'$. We first show by induction that for any $i \in [0,k]$ and $B \in P_i$, we have that $\cup R_i(B) \in \mathrm{img}(\mathcal{S}_{\cup,\mathrm{pre}}(\mu))$:

($i = 0$) We have that $\langle P_0, R_0 \rangle = \langle P_\mu, R_\mu \rangle$ so that for any $B \in P_0$, by Lemma 2.4 (ii), $\cup R_0(B) = \cup\{C \in \mathrm{par}(\mu) \mid C \subseteq \mu(B)\} = \mu(B)$. Hence, $\cup R_0(B) \in \mathrm{img}(\mu) \subseteq \mathrm{img}(\mathcal{S}_{\cup,\mathrm{pre}}(\mu))$.

($i+1$) Let $C \in P_{i+1} = \mathrm{split}(P_i, \mathrm{pre}(\cup R_i(B_i)))$ for some $B_i \in P_i$. If $C \cap \mathrm{pre}(B_i) = \varnothing$ then, by lines 6-7, $\cup R_{i+1}(C) = \cup R_i(\mathrm{parent}_{P_i}(C))$ so that, by inductive hypothesis, $\cup R_{i+1}(C) \in \mathrm{img}(\mathcal{S}_{\cup,\mathrm{pre}}(\mu))$. On the other hand, if $C \cap \mathrm{pre}(B_i) \neq \varnothing$ then, by lines 6-7, $\cup R_{i+1}(C) = \cup R_i(\mathrm{parent}_{P_i}(C)) \cap \mathrm{pre}(\cup R_i(B_i))$. By inductive hypothesis, we have that $\cup R_i(\mathrm{parent}_{P_i}(C)) \in \mathrm{img}(\mathcal{S}_{\cup,\mathrm{pre}}(\mu))$ and $\cup R_i(B_i) \in \mathrm{img}(\mathcal{S}_{\cup,\mathrm{pre}}(\mu))$. Also, since $\mathcal{S}_{\cup,\mathrm{pre}}(\mu)$ is forward complete for pre, $\mathrm{pre}(\cup R_i(B_i)) \in \mathrm{img}(\mathcal{S}_{\cup,\mathrm{pre}}(\mu))$. Hence, $\cup R_{i+1}(C) \in \mathrm{img}(\mathcal{S}_{\cup,\mathrm{pre}}(\mu))$.

As observed above, R_k is reflexive so that for any $B \in P_k$, $B \subseteq \cup R_k(B)$. For any $B \in P'$, we have that

$$\begin{array}{rl}
\mathcal{S}_{\cup,\mathrm{pre}}(\mu)(B) \subseteq & \text{[as } B \subseteq \cup R_k(B)\text{]} \\
\mathcal{S}_{\cup,\mathrm{pre}}(\mu)(\cup R_k(B)) = & \text{[as } \cup R_k(B) \in \mathrm{img}(\mathcal{S}_{\cup,\mathrm{pre}}(\mu))\text{]} \\
\cup R_k(B) \subseteq & \text{[as } R_k \subseteq R_k^*\text{]} \\
\cup R_k^*(B) = & \text{[by definition]} \\
\mu'(B). &
\end{array}$$

Therefore, for any $X \in \wp(\Sigma)$,

$$\begin{array}{rl}
\mathcal{S}_{\cup,\mathrm{pre}}(\mu)(X) \subseteq & \text{[as } X \subseteq \cup\{B \in P' \mid B \cap X \neq \varnothing\}\text{]} \\
\mathcal{S}_{\cup,\mathrm{pre}}(\mu)(\cup\{B \in P' \mid B \cap X \neq \varnothing\}) = & \text{[as } \mathcal{S}_{\cup,\mathrm{pre}}(\mu) \text{ is additive]} \\
\cup\{\mathcal{S}_{\cup,\mathrm{pre}}(\mu)(B) \mid B \in P',\ B \cap X \neq \varnothing\} \subseteq & \text{[as } \mathcal{S}_{\cup,\mathrm{pre}}(\mu)(B) \subseteq \mu'(B)\text{]} \\
\cup\{\mu'(B) \mid B \in P',\ B \cap X \neq \varnothing\} = & \text{[as } \mu' \text{ is disjunctive, by Lemma 2.4 (iii)]} \\
\mu'(X). &
\end{array}$$

We have therefore shown that $\mathcal{S}_{\cup,\mathrm{pre}}(\mu) \sqsubseteq \mu'$. □

Thus, BasicSA computes the forward $\{\cup, \mathrm{pre}\}$-complete shell of any input abstract domain. As a consequence, BasicSA allows us to compute both simulation relation and equivalence when μ_ℓ is the initial abstract domain.

Corollary 6.2. *Let $\mathcal{K} = (\Sigma, \to, \ell)$ be a finite Kripke structure and $\mu_\ell \in \mathrm{uco}(\wp(\Sigma))$ be the abstract domain induced by ℓ. Then,* $\mathrm{BasicSA}(\mu_\ell) = \langle P', R' \rangle$ *where* $P' = P_{\mathrm{sim}}$ *and, for any* $s_1, s_2 \in \Sigma$, $(s_1, s_2) \in R_{\mathrm{sim}} \Leftrightarrow (P_{\mathrm{sim}}(s_1), P_{\mathrm{sim}}(s_2)) \in R'$.

Proof. Let $\mu_{\mathcal{K}} = \mathcal{S}_{\cup,\mathrm{pre}}(\mu_\ell)$. By Theorem 6.1, if $\mathrm{BasicSA}(\mu_\ell) = \langle P', R' \rangle$ then $\mu_{\langle P', R' \rangle} = \mu_{\mathcal{K}}$. By Theorem 3.1, $\mathrm{par}(\mu_{\mathcal{K}}) = P_{\mathrm{sim}}$. By Lemma 4.2 (i), $P' \preceq \mathrm{par}(\mu_{\langle P', R' \rangle}) = \mathrm{par}(\mu_{\mathcal{K}}) = P_{\mathrm{sim}}$. It remains to show that $P_{\mathrm{sim}} = \mathrm{par}(\mu_{\langle P', R' \rangle}) \preceq P'$. Let $\{\langle P_i, R_i \rangle\}_{i \in [0,k]}$ be the sequence of partition-relation pairs computed by BasicSA, where $\langle P_0, R_0 \rangle = \langle P_{\mu_\ell}, R_{\mu_\ell} \rangle$ and $\langle P_k, R_k \rangle = \langle P', R' \rangle$. We show by induction that for any $i \in [0,k]$, we have that $\mathrm{par}(\mu_{\langle P', R' \rangle}) \preceq P_i$.

($i = 0$) Since $\mu_{\langle P', R' \rangle} \sqsubseteq \mu_\ell$, we have that $\mathrm{par}(\mu_{\langle P', R' \rangle}) \preceq \mathrm{par}(\mu_\ell) = P_0$.

($i+1$) Consider $B \in \mathrm{par}(\mu_{\langle P', R' \rangle})$. We have that $P_{i+1} = \mathrm{split}(P_i, \mathrm{pre}_\to(\cup R_i(B_i)))$ for some $B_i \in P_i$. We have shown in the proof of Theorem 6.1 that $\cup R_i(B_i) \in \mu_{\mathcal{K}} = \mu_{\langle P', R' \rangle}$. Since $\mu_{\langle P', R' \rangle}$ is forward complete for pre, we also have that $\mathrm{pre}(\cup R_i(B_i)) \in \mu_{\langle P', R' \rangle}$. Hence, $B \cap \mathrm{pre}_\to(\cup R_i(B_i)) \in \{\varnothing, B\}$. By inductive hypothesis, $\mathrm{par}(\mu_{\langle P', R' \rangle}) \preceq P_i$ so that there exists some $C \in P_i$ such that

```
1  RefinedSA(PartitionRelation ⟨P, Rel⟩) {
2    forall B ∈ P do prePrevRel(B) := Σ;
3    while ∃B ∈ P such that pre(∪Rel(B)) ≠ prePrevRel(B) do
4      //  Inv1:  ∀B ∈ P. pre(∪Rel(B)) ⊆ prePrevRel(B)
5      //  Inv2:  ∀B, C ∈ P. C ∩ pre(B) ≠ ∅ ⇒ ∪Rel(C) ⊆ prePrevRel(B)
6      Remove := prePrevRel(B) ∖ pre(∪Rel(B));
7      prePrevRel(B) := pre(∪Rel(B));
8      P_prev := P; B_prev := B;
9      P := Split(P, prePrevRel(B));
10     forall C ∈ P do
11       Rel(C) := {D ∈ P | D ⊆ ∪Rel(parent_{P_prev}(C))};
12       if C ∈ P ∖ P_prev then prePrevRel(C) := prePrevRel(parent_{P_prev}(C));
13     forall C ∈ P such that C ∩ pre(B_prev) ≠ ∅ do
14       Rel(C) := {D ∈ Rel(C) | D ∩ Remove = ∅};
15 }
```

Figure 3: Refined Simulation Algorithm.

$B \subseteq C$. Since $P_{i+1} = \text{split}(P_i, \text{pre}_{\rightarrow}(\cup R_i(B_i)))$, note that if $C \cap \text{pre}_{\rightarrow}(\cup R_i(B_i)) \neq \varnothing$ then $C \cap \text{pre}_{\rightarrow}(\cup R_i(B_i)) \in P_{i+1}$ and if $C \setminus (\text{pre}_{\rightarrow}(\cup R_i(B_i))) \neq \varnothing$ then $C \setminus (\text{pre}_{\rightarrow}(\cup R_i(B_i))) \in P_{i+1}$. Moreover, if $B \cap \text{pre}_{\rightarrow}(\cup R_i(B_i)) = \varnothing$ then $B \subseteq C \setminus (\text{pre}_{\rightarrow}(\cup R_i(B_i)))$, while if $B \cap \text{pre}_{\rightarrow}(\cup R_i(B_i)) = B$ then $B \subseteq C \cap \text{pre}_{\rightarrow}(\cup R_i(B_i))$. In both cases, there exists some $D \in P_{i+1}$ such that $B \subseteq D$.

Thus, $P' = P_{\text{sim}}$.
The proof of Theorem 6.1 shows that R' is reflexive. Moreover, that proof also shows that for any $B \in P'$, $\cup R'(B) \in \mu_{\mathcal{K}}$. Then, for any $B \in P'$:

$$\begin{aligned}
\cup R'^{*}(B) = & \quad \text{[by definition of } \mu_{\langle P', R' \rangle}\text{]} \\
\mu_{\langle P', R' \rangle}(B) \subseteq & \quad \text{[because } R' \text{ is reflexive]} \\
\mu_{\langle P', R' \rangle}(\cup R'(B)) = & \quad \text{[because } \mu_{\langle P', R' \rangle} = \mu_{\mathcal{K}}\text{]} \\
\mu_{\mathcal{K}}(\cup R'(B)) = & \quad \text{[because } \cup R'(B) \in \mu_{\mathcal{K}}\text{]} \\
\cup R'(B) &
\end{aligned}$$

and therefore R' is transitive. Hence, for any $s_1, s_2 \in \Sigma$,

$$\begin{aligned}
(s_1, s_2) \in R_{\text{sim}} \Leftrightarrow & \quad \text{[by Theorem 3.1]} \\
s_2 \in \mu_{\mathcal{K}}(\{s_1\}) \Leftrightarrow & \quad \text{[because } \mu_{\mathcal{K}} = \mu_{\langle P', R' \rangle}\text{]} \\
s_2 \in \mu_{\langle P', R' \rangle}(\{s_1\}) \Leftrightarrow & \quad \text{[by definition of } \mu_{\langle P', R' \rangle}\text{]} \\
(P'(s_1), P'(s_2)) \in R'^{*} \Leftrightarrow & \quad \text{[because } P' = P_{\text{sim}} \text{ and } R'^{*} = R'\text{]} \\
(P_{\text{sim}}(s_1), P_{\text{sim}}(s_2)) \in R'. &
\end{aligned}$$

□

6.2 Refining the Algorithm

The BasicSA algorithm is refined to the RefinedSA procedure in Figure 3. This is obtained by adapting the ideas of Henzinger et al.'s *RefinedSimilarity* procedure in Figure 1 to our BasicSA algorithm. The following points show that this algorithm RefinedSA remains correct, i.e. the input-output behaviours of BasicSA and RefinedSA are the same.

– For any block B of the current partition P, the predecessors of the blocks in the "previous" relation $Rel_{\text{prev}}(B)$ are maintained as a set $\text{pre}PrevRel(B)$. Initially, at line 2, $\text{pre}PrevRel(B)$ is set to

```
1  SA(PartitionRelation ⟨P, Rel⟩) {
2    // forall B ∈ P do prePrevRel(B) := Σ;
3    forall B ∈ P do Remove(B) := Σ ∖ pre(∪Rel(B));
4    while ∃B ∈ P such that Remove(B) ≠ ∅ do
5      // Inv3: ∀C ∈ P. Remove(C) = prePrevRel(C) ∖ pre(∪Rel(C))
6      // Inv4: ∀C ∈ P. Split(P, prePrevRel(C)) = P
7      // prePrevRel(B) := pre(∪Rel(B));
8      Remove := Remove(B);
9      Remove(B) := ∅;
10     B_prev := B;
11     P_prev := P;
12     P := Split(P, Remove);
13     forall C ∈ P do
14       Rel(C) := {D ∈ P | D ⊆ ∪Rel(parent_{P_prev}(C))};
15       if C ∈ P ∖ P_prev then
16         Remove(C) := Remove(parent_{P_prev}(C));
17         // prePrevRel(C) := prePrevRel(parent_{P_prev}(C));
18     RemoveList := {D ∈ P | D ⊆ Remove};
19     forall C ∈ P such that C ∩ pre(B_prev) ≠ ∅ do
20       forall D ∈ RemoveList do
21         if D ∈ Rel(C) then
22           Rel(C) := Rel(C) ∖ {D};
23           forall s ∈ pre(D) such that s ∉ pre(∪Rel(C)) do
24             Remove(C) := Remove(C) ∪ {s};
25 }
```

Figure 4: The Simulation Algorithm SA.

contain all the states in Σ. Then, when a block B is selected by the while-loop at some iteration i, $\mathrm{pre}PrevRel(B)$ is updated at line 7 in order to save the states in $\mathrm{pre}(\cup Rel(B))$ at this iteration i.

– If C is a newly generated block after splitting P and $\mathrm{parent}_{P_{\mathrm{prev}}}(C)$ is its corresponding parent block in the partition before splitting then $\mathrm{pre}PrevRel(C)$ is set at line 12 as $\mathrm{pre}PrevRel(\mathrm{parent}_{P_{\mathrm{prev}}}(C))$. Therefore, since the current relation Rel decreases only — i.e., if i and j are iterations such that j follows i and B, B' are blocks such that $B' \subseteq B$ then $\cup Rel_j(B') \subseteq \cup Rel_i(B)$ — at each iteration, the following invariant Inv_1 holds: for any block $B \in P$, $\mathrm{pre}(\cup Rel(B)) \subseteq \mathrm{pre}PrevRel(B)$. Initially, Inv_1 is satisfied because for any block B, $\mathrm{pre}PrevRel(B)$ is initialized to Σ at line 2.

– The crucial point is the invariant Inv_2: if $C \rightarrow^{\exists\exists} B$ and $D \in Rel(C)$ then $D \subseteq \mathrm{pre}PrevRel(B)$. Initially, this invariant property is clearly satisfied because for any block B, $\mathrm{pre}PrevRel(B)$ is initialized to Σ. Morever, Inv_2 is maintained at each iteration because at line 6 $Remove$ is set to $\mathrm{pre}PrevRel(B) \setminus \mathrm{pre}(\cup Rel(B))$ and for any block C such that $C \rightarrow^{\exists\exists} B_{\mathrm{prev}}$ if some block D is contained in $Remove$ then D is removed from $Rel(C)$ at line 14.

Thus, if the exit condition of the while-loop of $\mathrm{RefinedSA}$ is satisfied then, by invariant Inv_2, the exit condition of $\mathrm{BasicSA}$ is satisfied as well.

Finally, let us remark that the exit condition of the while-loop, namely $\forall B \in P.\ \mathrm{pre}(\cup Rel(B)) = \mathrm{pre}PrevRel(B)$, is strictly weaker than the exit condition that we would obtain as counterpart of the exit condition of the while-loop of Henzinger et al.'s *RefinedSimilarity* procedure, i.e. $\forall B \in P.\ Rel(B) = Rel_{\mathrm{prev}}(B)$.

6.3 The Final Algorithm

Following the underlying ideas that lead from $\mathit{RefinedSimilarity}$ to HHK, the algorithm $\mathrm{RefinedSA}$ is further refined to its final version SA in Figure 4. The idea is that instead of recomputing at each iteration of the while-loop the set $\mathit{Remove} = \mathrm{pre}\mathit{PrevRel}(B) \setminus \mathrm{pre}(\cup \mathit{Rel}(B))$ for the selected block B, we maintain a set of states $\mathit{Remove}(B) \subseteq \Sigma$ for each block B of the current partition. For any block C, $\mathit{Remove}(C)$ is updated in order to satisfy the invariant condition Inv_3: $\mathit{Remove}(C)$ contains exactly the set of states that belong to $\mathrm{pre}\mathit{PrevRel}(C)$ but are not in $\mathrm{pre}(\cup \mathit{Rel}(C))$, where $\mathrm{pre}\mathit{PrevRel}(C)$ is logically defined as in $\mathrm{RefinedSA}$ but is not really stored. Moreover, the invariant condition Inv_4 ensures that, for any block C, $\mathrm{pre}\mathit{PrevRel}(C)$ is a union of blocks of the current partition. This allows us to replace the operation $\mathit{Split}(P, \mathrm{pre}(\cup \mathit{Rel}(B)))$ in $\mathrm{RefinedSA}$ with the equivalent split operation $\mathit{Split}(P, \mathit{Remove})$. The correctness of such replacement follows from the invariant condition Inv_4 by exploiting the following general remark.

Lemma 6.3. *Let P be a partition, T be a union of blocks in P and $S \subseteq T$. Then, $\mathit{Split}(P, S) = \mathit{Split}(P, T \setminus S)$.*

Proof. Assume that $B \cap T = \varnothing$, so that $B \cap S = \varnothing$. Then,

$$B \cap (T \setminus S) = B \cap (T \cap \neg S) = \varnothing = B \cap S$$

and

$$B \setminus (T \setminus S) = (B \cap \neg T) \cup (B \cap S) = B = B \setminus S$$

so that B is split neither by $T \setminus S$ nor by S.
Otherwise, if $B \cap T \neq \varnothing$, because T is a union of blocks, then $B \subseteq T$. Then,

$$B \cap (T \setminus S) = B \cap (T \cap \neg S) = B \cap \neg S = B \setminus S$$

and

$$B \setminus (T \setminus S) = (B \cap \neg T) \cup (B \cap S) = B \cap S$$

so that B is split by $T \setminus S$ into B_1 and B_2 if and only if B is split by S into B_1 and B_2. We have thus shown that $\mathit{Split}(P, S) = \mathit{Split}(P, T \setminus S)$. □

The equivalence between SA and $\mathrm{RefinedSA}$ is a consequence of the following observations.

- Initially, the invariant properties Inv_3 and Inv_4 clearly hold because for any block B, $\mathrm{pre}\mathit{PrevRel}(B) = \Sigma$.
- When a block B_{prev} of the current partition is selected by the while-loop, the corresponding remove set $\mathit{Remove}(B_{\mathrm{prev}})$ is set to empty at line 9. The invariant Inv_3, namely $\forall C.\ \mathit{Remove}(C) = \mathrm{pre}\mathit{PrevRel}(C) \setminus \mathrm{pre}(\cup \mathit{Rel}(C))$, is maintained at each iteration because for any block C such that $C \rightarrow^{\exists\exists} B_{\mathrm{prev}}$ the for-loop at lines 23-24 incrementally adds to $\mathit{Remove}(C)$ all the states s that are in $\mathrm{pre}\mathit{PrevRel}(C)$ but not in $\mathrm{pre}(\cup \mathit{Rel}(C))$.
- If C is a newly generated block after splitting P and $\mathrm{parent}_{P_{\mathrm{prev}}}(C)$ is its corresponding parent block in the partition before splitting then $\mathit{Remove}(C)$ is set to $\mathit{Remove}(\mathrm{parent}_{P_{\mathrm{prev}}}(C))$ by the for-loop at lines 13-17.
- As in $\mathrm{RefinedSA}$, for any block C such that $C \rightarrow^{\exists\exists} B_{\mathrm{prev}}$, all the blocks that are contained in $\mathit{Remove}(B_{\mathrm{prev}})$ are removed from $\mathit{Rel}(C)$ by the for-loop at lines 20-22.

If the exit condition of the while-loop of SA is satisfied then, by Inv_1 and Inv_3, the exit condition of $\mathrm{RefinedSA}$ is satisfied as well.

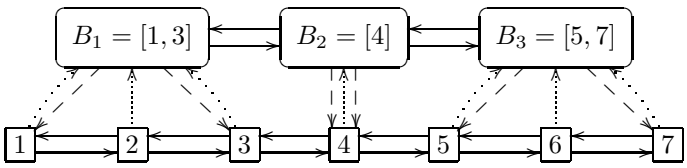


Figure 5: Partition representation.

7 Complexity

7.1 Data Structures

SA is implemented by using the following data structures.

(i) The set of states Σ is represented as a doubly linked list where each state $s \in \Sigma$ (represented as an integer) stores the list of its predecessors in $\mathrm{pre}(\{s\})$. This provides a representation of the input transition system. Any state $s \in \Sigma$ also stores a pointer to the block of the current partition that contains s.

(ii) The states of any block B of the current partition are consecutive in the list Σ, so that B is represented by a record that contains two pointers to the first and to the last state in B (see Figure 5). This structure allows us to move a state from a block to a different block in constant time. Moreover, any block B stores its corresponding remove set $B.Remove$, which is represented as a list of (pointers to) states.

(iii) Any block B additionally stores an integer array $RelCount$ that is indexed over Σ and is defined as follows: for any $x \in \Sigma$, $B.RelCount(x) = \sum_{C \in Rel(B)} |\{(x,y) \mid x \to y,\ y \in C\}|$ is the number of transitions from x to some block $C \in Rel(B)$. The array $RelCount$ allows to implement in constant time the test $s \notin \mathrm{pre}(\cup Rel(C))$ at line 23 as $C.RelCount(s) = 0$.

(iv) The current partition is stored as a doubly linked list P of blocks. Newly generated blocks are appended or prepended to this list. Blocks are scanned from the beginning of this list by checking whether the corresponding remove set is empty or not. If an empty remove set of some block B becomes nonempty then B is moved to the end of P.

(v) The current relation Rel on the current partition P is stored as a resizable $|P| \times |P|$ boolean matrix [11, Section 17.4]. The algorithm adds a new entry to this matrix, namely a new row and a new column, as long as a block B is split at line 12 into two new blocks $B \setminus Remove$ and $B \cap Remove$: the new block $B \setminus Remove$ replaces the old block B in P while a new entry in the matrix Rel corresponds to the new block $B \cap Remove$. We will observe later that the overall number of newly generated blocks by the splitting operation at line 12 is exactly given by $2(|P_{\mathrm{sim}}| - |P_{\mathrm{in}}|)$. Hence, the total number of insert operations in the matrix Rel is $|P_{\mathrm{sim}}| - |P_{\mathrm{in}}| \leq |P_{\mathrm{sim}}|$. Since an insert operation in a resizable array (whose capacity is doubled as needed) takes an amortized constant time, the overall cost of inserting new entries to the matrix Rel is in $O(|P_{\mathrm{sim}}|^2)$-time. Let us recall that the standard C++ vector class implements a resizable array so that a resizable boolean matrix can be easily implemented as a C++ vector of boolean vectors: in this implementation, the algorithm adds a new entry to a $N \times N$ matrix by first inserting a new vector of size $N+1$ containing $false$ values and then by inserting $N+1$ $false$ values in the $N+1$ boolean vectors.

7.2 Space and Time Complexity

Let $B \in P_{\mathrm{in}}$ be some block of the initial partition P_{in} and let $\langle B_i \rangle_{i \in It}$ be the sequence of all the blocks selected by the while-loop in a sequence It of iterations such that:

(a) for any $i \in It$, $B_i \subseteq B$;

(b) if an iteration $j \in It$ follows an iteration $i \in It$, denoted by $i < j$, then B_j is contained in B_i.

Observe that B is the parent block in P_{in} of all the B_i's. Then, one key property of the SA algorithm is that the remove sets in $\{Remove(B_i)\}_{i \in It}$ are pairwise disjoint so that $\sum_{i \in It} | Remove(B_i)| \leq |\Sigma|$. This property guarantees that if the test $D \in RemoveList$ at line 20 is positive at some iteration $i \in It$ then for any block $D' \subseteq D$ and for any successive iteration $j > i$, with $j \in It$, the test $D' \in RemoveList$ will be negative. Moreover, if the test $D \in Rel(C)$ at line 21 is positive at some iteration $i \in It$, so that D is removed from $Rel(C)$, then for all the blocks D' and C' such that $D' \subseteq D$ and $C' \subseteq C$ the test $D' \in Rel(C')$ will be negative for all the iterations $j > i$. As a further consequence, since a splitting operation $Split(P, Remove)$ can be executed in $O(|Remove|)$-time, it turns out that the overall cost of all the splitting operations is in $O(|P_{\text{sim}}||\Sigma|)$-time. Furthermore, by using the data structures described by points (iii) and (v) in Section 7.1, the tests $D \in Rel(C)$ at line 21 and $s \notin \text{pre}(\cup Rel(C))$ at line 23 can be executed in constant time. A careful analysis that exploits these key facts allows us to show that the total running time of SA is in $O(|P_{\text{sim}}||{\to}|)$.

Theorem 7.1. *The algorithm* SA *runs in $O(|P_{\text{sim}}||{\to}|)$-time and $O(|P_{\text{sim}}||\Sigma| \log |\Sigma|)$-space.*

Proof. Let It denote the sequence of iterations of the while-loop for some run of SA, where for any $i, j \in It$, $i < j$ means that j follows i. Moreover, for any $i \in It$, B_i denotes the block selected by the while-loop at line 4, $Remove(B_i) \neq \varnothing$ denotes the corresponding nonempty remove set, $\text{pre}(\cup Rel(B_i))$ denotes the corresponding set for B_i, while $\langle P_i, Rel_i \rangle$ denotes the partition-relation pair at the entry point of the for-loop at line 19.
Consider the set $\mathcal{B} \stackrel{\text{def}}{=} \{B_i \in P_i \mid i \in It\}$ of selected blocks and the following relation on $\mathcal{B}$:

$$B_i \trianglelefteq B_j \;\; \Leftrightarrow \;\; B_i \subsetneq B_j \text{ or } (B_i = B_j \;\&\; i \geq j)$$

It turns out that $\langle \mathcal{B}, \trianglelefteq \rangle$ is a poset. In fact, $\trianglelefteq$ is trivially reflexive. Also, $\trianglelefteq$ is transitive: assume that $B_i \trianglelefteq B_j$ and $B_j \trianglelefteq B_k$; if $B_i = B_j = B_k$ then $i \geq j \geq k$ so that $B_i \trianglelefteq B_k$; otherwise either $B_i \subsetneq B_j$ or $B_j \subsetneq B_k$ so that $B_i \subsetneq B_k$ and therefore $B_i \trianglelefteq B_k$. Finally, $\trianglelefteq$ is antisymmetric: if $B_i \trianglelefteq B_j$ and $B_j \trianglelefteq B_i$ then $B_i = B_j$ and $i \geq j \geq i$ so that $i = j$. Moreover, $B_i \triangleleft B_j$ denotes the corresponding strict order: this happens when either $B_i \subsetneq B_j$ or $B_i = B_j$ and $i > j$.
The time complexity bound is shown incrementally by the following points.

(A) For any $B_i, B_j \in \mathcal{B}$, if $B_i \subseteq B_j$ and $j < i$ then $Remove(B_i) \cap Remove(B_j) = \varnothing$.

Proof. By invariant Inv_3, $Remove(B_j) \cap \text{pre}(\cup Rel_j(B_j)) = \varnothing$. At iteration j, $Remove(B_j)$ is set to $\varnothing$ at line 9. If B_j generates, by the splitting operation at line 12, two new blocks $B_1, B_2 \subseteq B_j$ then their remove sets are set to $\varnothing$ at line 16. Successively, SA may add at line 24 of some iteration $k \geq j$ a state s to the remove set $Remove(C)$ of a block $C \subseteq B_j$ only if $s \in \text{pre}(\cup Rel_k(C))$. We also have that $\cup Rel_k(C) \subseteq \cup Rel_j(B_j)$ so that $\text{pre}(\cup Rel_k(C)) \subseteq \text{pre}(\cup Rel_j(B_j))$. Thus, if $B_i \subseteq B_j$ and $i > j$ then $Remove(B_i) \subseteq \text{pre}(\cup Rel_j(B_j))$. Therefore, $Remove(B_j) \cap Remove(B_i) \subseteq Remove(B_j) \cap \text{pre}(\cup Rel_j(B_j)) = \varnothing$.

(B) The overall number of newly generated blocks by the splitting operation at line 12 is $2(|P_{\text{sim}}| - |P_{\text{in}}|)$.

Proof. Let $\{P_i\}_{i \in [0,n]}$ be the sequence of partitions computed by SA where P_0 is the initial partition P_{in}, P_n is the final partition P_{sim} and for all $i \in [0, n-1]$, $P_{i+1} \preceq P_i$. The number of newly generated blocks by one splitting operation that refines P_i to P_{i+1} is given by $2(|P_{i+1}| - |P_i|)$. Thus, the overall number of newly generated blocks is $\sum_{i=0}^{n-1} 2(|P_{i+1}| - |P_i|) = 2(|P_{\text{sim}}| - |P_{\text{in}}|)$.

(C) The time complexity of the for-loop at line 3 is in $O(|P_{\text{in}}||{\to}|)$.

Proof. For any $B \in P_{\text{in}}$, $\text{pre}(\cup Rel(B))$ is computed in $O(|{\to}|)$-time, so that $\Sigma \setminus \text{pre}(\cup Rel(B))$ is computed in $O(|{\to}|)$-time as well. The time complexity of the initialization of the remove sets is therefore in $O(|P_{\text{in}}||{\to}|)$.

(D) The overall time complexity of lines 8 and 18 is in $O(|P_{\text{sim}}||\Sigma|)$.

Proof. Note that at line 18, $Remove$ is a union of blocks of the current partition P. As described in Section 7.1 (i), each state s also stores a pointer to the block of the current partition that contains

```
1  ListOfBlocks Split(PartitionRelation& P, SetOfStates S) {
2     ListOfBlocks split = empty;
3     forall s in S do {
4        Block B = s.block;
5        if (B.intersection == NULL) then {
6           B.intersection = new Block;
7           if (B.remove == ∅) then P.prepend(B.intersection);
8                              else P.append(B.intersection);
9           split.append(B);
10       }
11       move s from B to B.intersection;
12       if (B == empty) then {
13          B = copy(B.intersection);
14          P.remove(B.intersection);
15          delete B.intersection;
16          split.remove(B);
17       }
18    }
19    return split;
20 }
21
22 SplittingProcedure(P,S) {
23    /* P_prev = P; */
24    ListOfBlocks split = Split(P,S);
25    /* assert(split == {B↘S ∈ P | B↘S ∉ P_prev}) */
26    forall B in split do {
27       Rel.addNewEntry(B.intersection);
28       B.intersection.Remove = copy(B.Remove);
29    }
30    forall B in P do
31       forall C in split do Rel(B,C.intersection) = Rel(B,C);
32    forall B in split do {
33       forall C in P do Rel(B.intersection,C) = Rel(B,C);
34       forall x in Σ do B.intersection.RelCount(x) = B.RelCount(x);
35    }
36 }
```

Figure 6: C++ Pseudocode Implementation of the Splitting Procedure.

s. The list of blocks $RemoveList$ is therefore computed by scanning all the states in $Remove(B_i)$, where B_i is the selected block at iteration i, so that the overall time complexity of lines 8 and 18 is bounded by $2\sum_{i\in It} |\, Remove(B_i)|$. For any block $E \in P_{\mathrm{sim}}$ of the final partition we define the following subset of iterations:

$$It_E \stackrel{\text{def}}{=} \{i \in It \mid E \subseteq B_i\}.$$

Since for any $i \in It$, $P_{\mathrm{sim}} \preceq P_i$, we have that for any $i \in It$ there exists some $E \in P_{\mathrm{sim}}$ such that $i \in It_E$. Note that if $i, j \in It_E$ and $i < j$ then $B_j \subseteq B_i$ and, by point (A), this implies that $Remove(B_i) \cap Remove(B_j) = \varnothing$. Thus,

$$\begin{aligned} 2\textstyle\sum_{i\in It} |\, Remove(B_i)| \leq & \quad \text{[by definition of } It_E\text{]} \\ 2\textstyle\sum_{E\in P_{\mathrm{sim}}} \sum_{i\in It_E} |\, Remove(B_i)| \leq & \quad \text{[as the sets in } \{Remove(B_i)\}_{i\in It_E} \text{ are pairwise disjoint]} \\ 2\textstyle\sum_{E\in P_{\mathrm{sim}}} |\Sigma| = & \\ 2|P_{\mathrm{sim}}||\Sigma|. & \end{aligned}$$

(E) The overall time complexity of line 10, i.e. of copying the list of states of the selected block B, is in $O(|P_{\mathrm{sim}}||\Sigma|)$.

Proof. For any block $E \in P_{\mathrm{sim}}$ of the final partition we define the following subset of iterations:

$$It_E \stackrel{\text{def}}{=} \{i \in It \mid E \subseteq Remove(B_i)\}.$$

Since for any $i \in It$, $P_{\mathrm{sim}} \preceq P_i$ and $Remove(B_i)$ is a union of blocks of P_i, it turns out that for any $i \in It$ there exists some $E \in P_{\mathrm{sim}}$ such that $i \in It_E$. Note that if $i, j \in It_E$ and $i \neq j$ then

$B_j \cap B_i = \varnothing$: this is a consequence of point (A) because $E \subseteq \mathit{Remove}(B_i) \cap \mathit{Remove}(B_j) \neq \varnothing$ implies that $B_j \not\subseteq B_i$ and $B_i \not\subseteq B_j$ so that $B_i \cap B_j = \varnothing$. Thus,

$$\begin{aligned}
\textstyle\sum_{i \in It} |B_i| \leq \quad & \text{[by definition of } It_E\text{]} \\
\textstyle\sum_{E \in P_{\mathrm{sim}}} \sum_{i \in It_E} |B_i| \leq \quad & \text{[as the blocks in } \{B_i\}_{i \in It_E} \text{ are pairwise disjoint]} \\
\textstyle\sum_{E \in P_{\mathrm{sim}}} |\Sigma| = & \\
|P_{\mathrm{sim}}||\Sigma|. &
\end{aligned}$$

(F) The overall time complexity of lines 11-17 is in $O(|P_{\mathrm{sim}}||{\to}|)$.

Proof. Figure 6 describes a C++ pseudocode implementation of lines 11-17. By using the data structures described in Section 7.1, and in particular in Figure 5, all the operations of the procedure Split take constant time so that any call $\mathit{Split}(P, S)$ takes $O(|S|)$ time. Let us now consider $\mathit{SplittingProcedure}$.

- The overall time complexity of the splitting operation at line 24 is in $O(|P_{\mathrm{sim}}||\Sigma|)$. Each call $\mathit{Split}(P, \mathit{Remove}(B_i))$ takes $O(|\mathit{Remove}(B_i)|)$ time. Then, analogously to the proof of point (D), the overall time complexity of line 24 is bounded by $\sum_{i \in It} |\mathit{Remove}(B_i)| \leq |P_{\mathrm{sim}}||\Sigma|$.
- The overall time complexity of the for-loop at lines 26-29 is in $O(|P_{\mathrm{sim}}||\Sigma|)$. It is only worth noticing that since the boolean matrix that stores Rel is resizable, each operation at line 27 that adds a new entry to this resizable matrix has an amortized cost in $O(|P_{\mathrm{sim}}|)$: in fact, the resizable matrix is just a resizable array A of resizable arrays so that when we add a new entry we need to add a new entry to A and then a new entry to each array in A (cf. point (v) in Section 7.1). Thus, the overall time complexity of line 26 is in $O(|P_{\mathrm{sim}}|^2)$.
- The overall time complexity of the for-loop at lines 30-31 is in $O(|P_{\mathrm{sim}}|^2)$.
- The overall time complexity of the for-loop at lines 32-35 is in $O(|P_{\mathrm{sim}}||{\to}|)$. This is a consequence of the fact that the overall time complexity of the for-loops at lines 33 and 34 is in $O(|P_{\mathrm{sim}}||{\to}|)$.

Thus, the overall time complexity of $\mathit{SplittingProcedure}(P, \mathit{Remove})$ is in $O(|P_{\mathrm{sim}}||{\to}|)$.

(G) The overall time complexity of lines 19-21 is in $O(|P_{\mathrm{sim}}||{\to}|)$.

Proof. For any $B_i \in \mathcal{B}$, let $\mathrm{arr}(B_i) \stackrel{\text{def}}{=} \sum_{x \in B_i} |\operatorname{pre}(\{x\})|$ denote the number of transitions that end in some state of B_i and $\mathrm{rem}(B_i) \stackrel{\text{def}}{=} |\{D \in P_i \mid D \subseteq \mathit{Remove}(B_i)\}|$ denote the number of blocks of P_i contained in $\mathit{Remove}(B_i)$. We also define two functions $f_{\lhd}, f_{\trianglelefteq} : \mathcal{B} \to \wp(P_{\mathrm{sim}})$ as follows:

$$\begin{aligned}
f_{\lhd}(B_i) &\stackrel{\text{def}}{=} \{D \in P_{\mathrm{sim}} \mid D \cap (\cup\{\mathit{Remove}(B_j) \mid B_j \in \mathcal{B},\ B_i \lhd B_j\}) = \varnothing\} \\
f_{\trianglelefteq}(B_i) &\stackrel{\text{def}}{=} \{D \in P_{\mathrm{sim}} \mid D \cap (\cup\{\mathit{Remove}(B_j) \mid B_j \in \mathcal{B},\ B_i \trianglelefteq B_j\}) = \varnothing\}
\end{aligned}$$

Let us show the following property:

$$\forall B_i \in \mathcal{B}.\ \mathrm{rem}(B_i) + |f_{\trianglelefteq}(B_i)| \leq |f_{\lhd}(B_i)|. \qquad (\ddagger)$$

We first observe that since $P_{\mathrm{sim}} \preceq P_i$, $\mathrm{rem}(B_i) \leq |\{D \in P_{\mathrm{sim}} \mid D \subseteq \mathit{Remove}(B_i)\}|$. Moreover, the sets $\{D \in P_{\mathrm{sim}} \mid D \subseteq \mathit{Remove}(B_i)\}$ and $f_{\trianglelefteq}(B_i)$ are disjoint and their union gives $f_{\lhd}(B_i)$. Hence,

$$\begin{aligned}
\mathrm{rem}(B_i) + |f_{\trianglelefteq}(B_i)| \leq & \\
|\{D \in P_{\mathrm{sim}} \mid D \subseteq \mathit{Remove}(B_i)\}| + |f_{\trianglelefteq}(B_i)| = & \\
|\{D \in P_{\mathrm{sim}} \mid D \subseteq \mathit{Remove}(B_i)\} \cup f_{\trianglelefteq}(B_i)| = & \\
|f_{\lhd}(B_i)|. &
\end{aligned}$$

Given, $B_k \in \mathcal{B}$, let us show by induction on the height $h(B_k) \geq 0$ of B_k in the poset $\langle \mathcal{B}, \trianglelefteq \rangle$ that

$$\textstyle\sum_{B_i \trianglelefteq B_k} \mathrm{arr}(B_i)\,\mathrm{rem}(B_i) \leq \mathrm{arr}(B_k)|f_{\triangleleft}(B_k)|. \qquad (*)$$

($h(B_k) = 0$): By property (‡), $\mathrm{rem}(B_k) \leq |f_{\triangleleft}(B_k)|$ so that

$$\textstyle\sum_{B_i \trianglelefteq B_k} \mathrm{arr}(B_i)\,\mathrm{rem}(B_i) = \mathrm{arr}(B_k)\,\mathrm{rem}(B_k) \leq \mathrm{arr}(B_k)|f_{\triangleleft}(B_k)|.$$

($h(B_k) > 0$): Let $\max(\{B_i \in \mathcal{B} \mid B_i \triangleleft B_k\}) = \{C_1, ..., C_n\}$. Note that if $i \neq j$ then $C_i \cap C_j = \varnothing$, so that $\sum_i \mathrm{arr}(C_i) \leq \mathrm{arr}(B_k)$, since $\cup_i C_i \subseteq B_k$. Let us observe that for any maximal C_i, $f_{\triangleleft}(C_i) \subseteq f_{\trianglelefteq}(B_k)$ because $\cup\{Remove(B_j) \mid B_j \in \mathcal{B},\ B_k \trianglelefteq B_j\} \subseteq \cup\{Remove(B_j) \mid B_j \in \mathcal{B},\ C_i \triangleleft B_j\}$ since $B_k \trianglelefteq B_j$ and $C_i \triangleleft B_k$ imply $C_i \triangleleft B_j$.

Hence, we have that

$$\begin{aligned}
\textstyle\sum_{B_i \trianglelefteq B_k} \mathrm{arr}(B_i)\,\mathrm{rem}(B_i) = &\quad \text{[by maximality of } C_i\text{'s]}\\
\mathrm{arr}(B_k)\,\mathrm{rem}(B_k) + \textstyle\sum_{C_i}\sum_{D \trianglelefteq C_i} \mathrm{arr}(D)\,\mathrm{rem}(D) \leq &\quad \text{[by inductive hypothesis on } h(C_i) < h(B_k)\text{]}\\
\mathrm{arr}(B_k)\,\mathrm{rem}(B_k) + \textstyle\sum_{C_i} \mathrm{arr}(C_i)|f_{\triangleleft}(C_i)| \leq &\quad \text{[as } f_{\triangleleft}(C_i) \subseteq f_{\trianglelefteq}(B_k)\text{]}\\
\mathrm{arr}(B_k)\,\mathrm{rem}(B_k) + |f_{\trianglelefteq}(B_k)| \textstyle\sum_{C_i} \mathrm{arr}(C_i) \leq &\quad \text{[as } \textstyle\sum_{C_i} \mathrm{arr}(C_i) \leq \mathrm{arr}(B_k)\text{]}\\
\mathrm{arr}(B_k)\,\mathrm{rem}(B_k) + |f_{\trianglelefteq}(B_k)|\,\mathrm{arr}(B_k) = &\\
\mathrm{arr}(B_k)(\mathrm{rem}(B_k) + |f_{\trianglelefteq}(B_k)|) \leq &\quad \text{[by (‡), } \mathrm{rem}(B_k) + |f_{\trianglelefteq}(B_k)| \leq |f_{\triangleleft}(B_k)|\text{]}\\
\mathrm{arr}(B_k)|f_{\triangleleft}(B_k)|. &
\end{aligned}$$

Let us now show that the global time-complexity of lines 19-21 is in $O(|P_{\mathrm{sim}}||{\rightarrow}|)$. Let $\max(\mathcal{B}) = \{M_1, ..., M_k\}$ be the maximal elements in $\mathcal{B}$ so that for any $i \neq j$, $M_i \cap M_j = \varnothing$, and in turn we have that $\sum_{M_i \in \max(\mathcal{B})} \mathrm{arr}(M_i) \leq |{\rightarrow}|$. By using the data structures described in Section 7.1, the test $D \in Rel(C)$ at line 21 takes constant time. Then, the overall complexity of lines 19-21 is

$$\begin{aligned}
\textstyle\sum_{B_i \in \mathcal{B}} \mathrm{arr}(B_i)\,\mathrm{rem}(B_i) = &\quad \text{[as the } M_i\text{'s are maximal in } \mathcal{B}\text{]}\\
\textstyle\sum_{M_i \in \max(\mathcal{B})}\sum_{D \trianglelefteq M_i} \mathrm{arr}(D)\,\mathrm{rem}(D) \leq &\quad \text{[by property } (*) \text{ above]}\\
\textstyle\sum_{M_i \in \max(\mathcal{B})} \mathrm{arr}(M_i)|P_{\mathrm{sim}}| = &\\
|P_{\mathrm{sim}}| \textstyle\sum_{M_i \in \max(\mathcal{B})} \mathrm{arr}(M_i) \leq &\quad \text{[as } \textstyle\sum_{M_i \in \max(\mathcal{B})} \mathrm{arr}(M_i) \leq |{\rightarrow}|\text{]}\\
|P_{\mathrm{sim}}||{\rightarrow}|. &
\end{aligned}$$

(H) The overall time complexity of lines 22-24 is in $O(|P_{\mathrm{sim}}||{\rightarrow}|)$.

Proof. Let $\mathcal{P}$ denote the multiset of pairs of blocks $(C, D) \in P_i$ that are scanned at lines 19-20 at some iteration $i \in It$ such that $D \in Rel_i(C)$. By using the data structures described in Section 7.1, the test $s \notin \mathrm{pre}(\cup Rel(C))$ and the statement $Rel(C) := Rel(C) \setminus \{D\}$ take constant time. Moreover, the statement $Remove(C) := Remove(C) \cup \{s\}$ also takes constant time because if a state s is added to $Remove(C)$ at line 24 then s was not already in $Remove(C)$ so that this operation can be implemented simply by appending s to the list of states that represents $Remove(C)$. Therefore, the overall time complexity of the body of the if-then statement at lines 21-24 is $\sum_{(C,D) \in \mathcal{P}} \mathrm{arr}(D)$. We notice the following fact. Let $i, j \in It$ such that $i < j$ and let (C, D_i) and (C, D_j) be pairs of blocks scanned at lines 19-20, respectively, at iterations i and j such that $D_j \subseteq D_i$. Then, if the test $D_i \in Rel_i(C)$ is true at iteration i then the test $D_j \in Rel_j(C)$ is false at iteration j. This is a consequence of the fact that if $D \in Rel_i(C)$ then D is removed from $Rel_i(C)$ at line 22 and $\cup Rel_j(C) \subseteq \cup Rel_i(C)$ so that $D \cap \cup Rel_j(C) = \varnothing$. Hence, if $(C, D), (C, D') \in \mathcal{P}$ then $D \cap D' = \varnothing$. We define the set $\mathcal{C} \stackrel{\text{def}}{=} \{C \mid \exists D.\ (C, D) \in \mathcal{P}\}$ and given $C \in \mathcal{C}$, the multiset $\mathcal{D}_C \stackrel{\text{def}}{=} \{D \mid (C, D) \in \mathcal{P}\}$. Observe that $|\mathcal{C}|$ is bounded by the number of blocks that appear in

```
Initialize(PartitionRelation P) {
   forall B in P do {
      B.Remove = pre(Σ) ∖ pre(∪{C in P | Rel(B,C)});
      forall x in Σ do B.RelCount(x) = 0;
   }
   forall B in P do
      forall y in B do
         forall x in pre({y}) do
            forall C in P such that Rel(C,B) do C.RelCount(x)++;
}

SA(PartitionRelation P) {
   Initialize(P);
   forall B in P such that (B.Remove ≠ ∅) do {
      Set Remove = B.Remove;
      B.Remove = ∅;
      Set B_prev = B;
      SplittingProcedure(P,Remove);
      ListOfBlocks RemoveList = {D ∈ P | D ⊆ Remove};
      forall C in P such that (C ∩ pre(B_prev) ≠ ∅) do
         forall D in RemoveList do
            if (Rel(C,D)) then {
               Rel(C,D) = 0;
               forall d in D do
                  forall x in pre(d) do {
                     C.RelCount(x)--;
                     if (C.RelCount(x) == 0) then {
                        C.Remove = C.Remove ∪ {x};
                        P.moveAtTheEnd(C);
                     }
                  }
            }
   }
}
```

Figure 7: C++ Pseudocode Implementation of SA.

some partition P_i, so that by point (B), $|\mathcal{C}| \leq 2(|P_{\text{sim}}| - |P_{\text{in}}|) + |P_{\text{in}}| \leq 2|P_{\text{sim}}|$. Moreover, the observation above implies that $\mathcal{D}_C$ is indeed a set and the blocks in $\mathcal{D}_C$ are pairwise disjoint. Thus,

$$\begin{aligned} \textstyle\sum_{(C,D)\in\mathcal{P}} \text{arr}(D) = & \\ \textstyle\sum_{C\in\mathcal{C}} \sum_{D\in\mathcal{D}_C} \text{arr}(D) \leq & \quad \text{[as the blocks in } \mathcal{D}_C \text{ are pairwise disjoint]} \\ \textstyle\sum_{C\in\mathcal{C}} |{\to}| \leq & \quad \text{[as } |\mathcal{C}| \leq 2|P_{\text{sim}}|\text{]} \\ 2|P_{\text{sim}}||{\to}|. & \end{aligned}$$

Summing up, we have shown that the overall time-complexity of SA is in $O(|P_{\text{sim}}||{\to}|)$.
The space complexity is in $O(|\Sigma|\log|P_{\text{sim}}| + |P_{\text{sim}}| + |P_{\text{sim}}|^2 + |P_{\text{sim}}||\Sigma|\log|\Sigma|) = O(|P_{\text{sim}}||\Sigma|\log|\Sigma|)$ where:

- The pointers from any state $s \in \Sigma$ to the block of the current partition that contains s are stored in $O(|\Sigma|\log|P_{\text{sim}}|)$ space.
- The current partition P is stored in $O(|P_{\text{sim}}|)$ space.
- The current relation Rel is stored in $O(|P_{\text{sim}}|^2)$ space.
- Each block of the current partition stores the corresponding remove set in $O(|\Sigma|)$ space and the integer array $RelCount$ in $O(|\Sigma|\log|\Sigma|)$, so that these globally take $O(|P_{\text{sim}}||\Sigma|\log|\Sigma|)$ space. □

8 Experimental Evaluation

A pseudocode implementation of the algorithm SA that shows how the data structures in Section 7.1 are actually used is in Figure 7, where *SplittingProcedure* has been introduced above in Figure 6. We implemented in C++ both our simulation algorithm SA and the HHK algorithm in order to experimentally

compare the time and space performances of SA and HHK. In order to make the comparison as meaningful as possible, these two C++ implementations use the same data structures for storing transitions systems, sets of states and tables.

Our benchmarks include systems from the VLTS (Very Large Transition Systems) benchmark suite [30] and some publicly available Esterel programs. These models are represented as labeled transition systems (LTSs) where labels are attached to transitions. Since the versions of SA and HHK considered in this paper both need as input a Kripke structure, namely a transition system where labels are attached to states, we exploited a procedure by Dovier et al. [16] that transforms a LTS M into a Kripke structure M' in such a way that bisimulation and simulation equivalences on M and M' coincide. This transformation acts as follows: any labeled transition $s_1 \xrightarrow{l} s_2$ is replaced by two unlabeled transitions $s_1 \rightarrow n$ and $n \rightarrow s_2$, where n is a new node that is labeled with l, while all the original states in M have the same label. This labeling provides an initial partition on M' which is denoted by P_{in}. Hence, 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. Also, the models cwi_3_14, vasy_5_9, vasy_25_25 and vasy_8_38 have non total transition relations. The vasy_* and cwi_* systems are taken from the VLTS suite, while the remaining systems are the following Esterel programs: WristWatch and ShockDance are taken from the programming examples of Esterel [17], ObsArbitrer4 and AtLeastOneAck4 are described in the technical report [3], lift, NoAckWithoutReq and one_pump are provided together with the fc2symbmin tool that is used by Xeve, a graphical verification environment for Esterel programs [4, 31].

Our experimental evaluation was carried out on an Intel Core 2 Duo 1.86 GHz PC, with 2 GB RAM, running Linux and GNU g++ 4. The results are summarised in Table 1, where we list the name of the transition system, the number of states and transitions of the transformed transition system, the number of blocks of the initial partition, the number of blocks of the final simulation equivalence partition (that is known when one algorithm terminates), the execution time in seconds and the allocated memory in MB (this has been obtained by means of glibc-memusage) both for HHK and SA, where o.o.m. means that the algorithm ran out of memory (2GB).

The comparative experimental evaluation shows that SA outperforms HHK both in time and in space. In fact, the experiments demonstrate that SA improves on HHK of about two orders of magnitude in time and of one order of magnitude in space. The sum of time and space measures on the eight models where both HHK and SA terminate is 64.555 vs. 1.39 seconds in time and 681.303 vs. 52.102 MB in space. Our experiments considered 18 models: HHK terminates on 8 models while SA terminates on 14 of these 18 models. Also, the size of models (states plus transitions) where SA terminates w.r.t. HHK grows about one order of magnitude.

9 Conclusion

We presented a new efficient algorithm for computing the simulation preorder in $O(|P_{\text{sim}}||{\rightarrow}|)$-time and $O(|P_{\text{sim}}||\Sigma|\log|\Sigma|)$-space, where P_{sim} is the partition induced by simulation equivalence on some Kripke structure $(\Sigma, \rightarrow)$. This improves the best available time bound $O(|\Sigma||{\rightarrow}|)$ given by Henzinger, Henzinger and Kopke's [23] and by Bloom and Paige's [2] simulation algorithms that however suffer from a space complexity that is bounded from below by $\Omega(|\Sigma|^2)$. A better space bound is given by Gentilini et al.'s [18] algorithm — subsequently corrected by van Glabbeek and Ploeger [21] — whose space complexity is in $O(|P_{\text{sim}}|^2+|\Sigma|\log|P_{\text{sim}}|)$, but that runs in $O(|P_{\text{sim}}|^2|{\rightarrow}|)$-time. Our algorithm is designed as an adaptation of Henzinger et al.'s procedure and abstract interpretation techniques are used for proving its correctness.

As future work, we plan to investigate whether the techniques used for designing this new simulation algorithm may be generalized and adapted to other behavioural equivalences like branching simulation equivalence (a weakening of branching bisimulation equivalence [15]). It is also interesting to investigate whether this new algorithm may admit a symbolic version based on BDDs.

Acknowledgements. The authors are grateful to the anonymous referees for their detailed and helpful comments and to Silvia Crafa for many useful discussions. This work was partially supported by the FIRB Project "Abstract interpretation and model checking for the verification of embedded systems", by the PRIN 2007 Project "AIDA2007: Abstract Interpretation Design and Applications" and by the University of

	Input			Output	HHK		SA	
Model	$\lvert\Sigma\rvert$	$\lvert\rightarrow\rvert$	$\lvert P_{\mathrm{in}}\rvert$	$\lvert P_{\mathrm{sim}}\rvert$	Time	Space	Time	Space
cwi_1_2	4339	4774	27	2401	22.761	191	0.76	41
cwi_3_14	18548	29104	3	123	–	o.o.m.	0.96	9
vasy_0_1	1513	2448	3	21	1.303	27	0.03	0.229
vasy_10_56	67005	112312	13	??	–	o.o.m.	–	o.o.m.
vasy_1_4	5647	8928	7	87	37.14	407	0.28	2
vasy_18_73	91789	146086	18	??	–	o.o.m.	–	o.o.m.
vasy_25_25	50433	50432	25217	??	–	o.o.m.	–	o.o.m.
vasy_40_60	100013	120014	4	??	–	o.o.m.	–	o.o.m.
vasy_5_9	15162	19352	32	409	–	o.o.m.	1.63	24
vasy_8_24	33290	48822	12	1423	–	o.o.m.	5.95	182
vasy_8_38	47345	76848	82	963	–	o.o.m.	8.15	176
WristWatch	1453	1685	23	1146	1.425	31	0.15	6
ShockDance	379	459	10	327	0.75	2	0.03	0.547
ObsArbitrer4	17389	21394	10	159	–	o.o.m.	0.3	11
AtLeastOneAck4	435	507	18	112	0.363	2	0.02	0.219
lift	138	163	33	112	0.11	0.303	0.02	0.107
NoAckWithoutReq	1212	1372	18	413	0.703	21	0.1	2
one_pump	15774	17926	22	3193	–	o.o.m.	13.64	194

Table 1: Results of the experimental evaluation.

Padova under the Project “Formal methods for specifying and verifying behavioural properties of software systems”. This paper is an extended and revised version of [28].

References

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

[2] 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.

[3] A. Bouali. Xeve: an Esterel Verification Environment (version v1_3). Rapport Technique 214/1997, INRIA, 1997.

[4] A. Bouali. Xeve: an Esterel verification environment. In *Proc. 10^{th} CAV*, LNCS 1427, pp. 500-504, 1998.

[5] M.C. Browne, E.M. Clarke and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theor. Comp. Sci.*, 59:115-131, 1988.

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

[7] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith. Progress on the state explosion problem in model checking. In *Informatics - 10 Years Back, 10 Years Ahead*. LNCS 2000, pp. 176-194, 2001.

[8] E.M. Clarke, O. Grumberg and D. Long. Model checking and abstraction. *ACM Trans. Program. Lang. Syst.*, 16(5):1512–1542, 1994.

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

[10] R. Cleaveland and O. Sokolsky. Equivalence and preorder checking for finite-state systems. In J.A. Bergstra, A. Ponse, S.A. Smolka eds., *Handbook of Process Algebra*, North-Holland, pp. 391-424, 2001.

[11] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein. *Introduction to Algorithms*. The MIT Press and McGraw-Hill, 2nd ed., 2001.

[12] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4^{th} ACM POPL*, pp. 238–252, 1977.

[13] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. 6^{th} ACM POPL*, pp. 269–282, 1979.

[14] D. Dams, O. Grumberg and R. Gerth. Generation of reduced models for checking fragments of CTL. In *Proc. 5^{th} CAV*, LNCS 697, pp. 479–490, 1993.

[15] R. De Nicola and F. Vaandrager. Three logics for branching bisimulation. *J. ACM*, 42(2):458–487, 1995

[16] A. Dovier, C. Piazza and A. Policriti. An efficient algorithm for computing bisimulation equivalence. *Theor. Comput. Sci.*, 325(1):45-67, 2004.

[17] Esterel Programming Examples. http://www-sop.inria.fr/esterel.org/Html/Downloads/Downloads.htm

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

[19] R. Giacobazzi and E. Quintarelli. Incompleteness, counterexamples and refinements in abstract model checking. In *Proc. 8^{th} SAS*, LNCS 2126, pp. 356-373, 2001.

[20] R. Giacobazzi and F. Ranzato. Optimal domains for disjunctive abstract interpretation. *Sci. Comp. Program.*, 32:177–210, 1998.

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

[22] O. Grumberg and D.E. Long. Model checking and modular verification. *ACM Trans. Program. Lang. Syst.*, 16(3):843–871, 1994.

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

[24] A. Kucera and R. Mayr. Why is simulation harder than bisimulation? In *Proc. 13^{th} CONCUR*, LNCS 2421, pp. 594-610, 2002.

[25] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:1–36, 1995.

[26] R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973-989, 1987

[27] F. Ranzato and F. Tapparo. Generalized strong preservation by abstract interpretation. *J. Logic and Computation*, 17(1):157-197, 2007.

[28] 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.

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

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

[31] Xeve: Esterel Verification Environment. http://www-sop.inria.fr/meije/verification/Xeve