A New Efficient Simulation Equivalence Algorithm

Francesco Ranzato Francesco Tapparo
Dipartimento di Matematica Pura ed Applicata
University of Padova, Padova, Italy
{ranzato, tapparo}@math.unipd.it

Abstract

It is well known that simulation equivalence is an appropriate abstraction to be used in model checking because it strongly preserves ACTL^* *and provides a better space reduction than bisimulation equivalence. However, computing simulation equivalence is harder than computing bisimulation equivalence. A number of algorithms for computing simulation equivalence exist. 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 quadratic space complexity that is bounded from below by* $\Omega(|\Sigma|^2)$*. The algorithm by Gentilini, Piazza, Policriti appears to be the best algorithm when both time and space complexities are taken into account. 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 equivalence algorithm that is obtained as a modification of Henzinger et al.'s algorithm and whose correctness is based on some techniques used in recent applications of abstract interpretation to model checking. Our algorithm runs in* $O(|P_{\mathrm{sim}}||\rightarrow|)$*-time and* $O(|P_{\mathrm{sim}}||\Sigma|)$*-space. Thus, while retaining a space complexity which is lower than quadratic, our algorithm improves the best known time bound.*

1. Introduction

Abstraction techniques are widely used in model checking to hide some properties of the concrete model and then to define a reduced abstract model where to run the verification algorithm. Abstraction provides an effective solution to deal with the state-explosion problem that arises in model checking systems with parallel components. 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 be true on the original unabstracted model as well. Ideally, the reduced structure 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 relation on system states that weakly/strongly preserves a given temporal language. Two well-known examples are bisimulation equivalence that strongly preserves branching-time logics such as CTL and CTL^* [2] and the simulation preorder that ensures weak preservation of ACTL and ACTL^* [10]. Simulation equivalence is weaker than bisimulation equivalence but stronger than simulation preorder because it strongly preserves ACTL, ACTL^*, ECTL and ECTL^* as well as the linear-time logic LTL [7, 10, 13]. This is particularly interesting because simulation equivalence can provide a much better state-space reduction w.r.t. bisimulation equivalence while retaining the ability of strongly preserving expressive temporal languages like ACTL^*. This explains why simulation equivalence is a common choice for reducing the concrete model.

State of the Art. It is well known that computing simulation equivalence is harder than computing bisimulation equivalence [12]. 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 some set AP of atomic propositions. Bisimulation equivalence can be computed by the well-known Paige and Tarjan's [14] 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 [11], Bloom and Paige [1], Bustan and Grumberg [3], Tan and Cleaveland [16] and Gentilini, Piazza and Policriti [8]. The algorithms by Henzinger, Henzinger and Kopke [11] and Bloom and Paige [1] run in $O(|\Sigma||\rightarrow|)$-time and, as far as time-complexity is concerned, they are the best algorithms. However, these algorithms have the drawback of a quadratic space complexity that is bounded from below

by $\Omega(|\Sigma|^2)$. This quadratic lower bound on the size of the state space is clearly a critical problem in the context of model checking. There is therefore a strong motivation for designing simulation equivalence algorithms that are less demanding on memory requirements. Bustan and Grumberg's [3] algorithm represents a first solution in this direction. Let P_{sim} denote the partition corresponding to simulation equivalence on $\mathcal{K}$ so that $|P_{\mathrm{sim}}|$ is the number of simulation equivalence classes. Then, Bustan and Grumberg's algorithm has a space complexity in $O(|P_{\mathrm{sim}}|^2 + |\Sigma| \log(|P_{\mathrm{sim}}|))$, although the time complexity in $O(|P_{\mathrm{sim}}|^4(|\rightarrow| + |P_{\mathrm{sim}}|^2) + |P_{\mathrm{sim}}|^2|\Sigma|(|\Sigma| + |P_{\mathrm{sim}}|^2|))$ remains a drawback. The algorithm by Gentilini, Piazza and Policriti [8] appears to be the best algorithm when both time and space complexities are taken into account. Gentilini et al.'s algorithm runs in $O(|P_{\mathrm{sim}}|^2|\rightarrow|)$-time, thus greatly improving on Bustan and Grumberg's algorithm, while the space complexity $O(|P_{\mathrm{sim}}|^2 + |\Sigma| \log(|P_{\mathrm{sim}}|))$ remains the same. Moreover, Gentilini et al. experimentally show that their procedure also improves on Tan and Cleaveland's [16] algorithm both in time and space while the theoretical complexities cannot be easily compared.

Main Results. This work presents a new efficient simulation equivalence algorithm that runs in $O(|P_{\mathrm{sim}}||\rightarrow|)$-time and $O(|P_{\mathrm{sim}}||\Sigma|)$-space. Thus, while retaining a space complexity lower than quadratic, our algorithm improves the best known time bound.

Our simulation equivalence algorithm is designed as a modification of Henzinger, Henzinger and Kopke's [11] algorithm, here denoted by HHK. In HHK, the quadratic lower bound $\Omega(|\Sigma|^2)$ on the space complexity derives from the fact that HHK 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 maintains instead: (i) a partition P of the state space Σ that is always coarser than the final partition P_{sim}, (ii) a relation $Rel \subseteq P \times P$ on the current partition P and (iii) for any block $B \in P$, a set of states $Remove(B) \subseteq \Sigma$. Thus, our space complexity is in $O(|P_{\mathrm{sim}}||\Sigma|)$, so that memory requirements may be much lower than quadratic in the size of the state space Σ.

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 $\mathcal{S} = \{Sim(s)\}_{s\in\Sigma}$, indexed on the whole state space Σ, with the following state partition $P_{\mathcal{S}}$ induced by $\mathcal{S}$: $s_1 \sim_{\mathcal{S}} s_2$ iff for all $s \in \Sigma$, $s_1 \in Sim(s) \Leftrightarrow s_2 \in Sim(s)$. Hence, if $s_1 \sim_{\mathcal{S}} s_2$ then $s_1 \in Sim(s_2)$ and $s_2 \in Sim(s_1)$ so that any block $B \in P_{\mathcal{S}}$ stores states that are currently candidates to be simulation equivalent. Additionally, we store and maintain a reflexive relation $Rel \subseteq P_{\mathcal{S}} \times P_{\mathcal{S}}$ on the partition $P_{\mathcal{S}}$ that gives rise to a so-called partition/relation pair. The logical intuition in this data structure is that if $B_1, B_2 \in P_{\mathcal{S}}$, $(B_1, B_2) \in Rel$ and $s_i \in B_i$ then the simulator set $Sim(s_2)$ is a subset of the simulator set $Sim(s_1)$, namely s_2 is currently a candidate for simulating s_1. In particular, being Rel reflexive, states that belong to a same block $B \in P_{\mathcal{S}}$ have the same current simulator set. It turns out that the information encoded by a partition/relation pair is enough for preserving the logical structure of HHK. In fact, this approach leads us to design an algorithm that resembles the HHK procedure: we follow Henzinger et al.'s approach both for proving the correctness of our algorithm and for devising an efficient implementation where, roughly, the number of states $|\Sigma|$ in HHK is replaced by the number of blocks of the simulation partition $|P_{\mathrm{sim}}|$. It is worth remarking that the correctness of our simulation equivalence algorithm is shown by resorting to abstract interpretation [5, 6]. More specifically, we exploit some recent results [15] that show how standard strong preservation of temporal languages in abstract Kripke structures can be generalized by abstract interpretation and cast as a completeness property of generic abstract domains that play the role of abstract models.

2. Background

2.1. Notation

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. $\mathrm{Part}(\Sigma)$ 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 is omitted) denotes the unique block in P_2 that contains B. For a given subset $S \subseteq \Sigma$ called splitter, we denote by $Split(P, S)$ the partition obtained from P by replacing each block $B \in P$ with the blocks $B \cap S$ and $B \smallsetminus S$, where we also allow no splitting, namely $Split(P, S) = P$.

Transition Systems. A transition system $\mathcal{T} = (\Sigma, \rightarrow)$ consists of a set Σ of states and a transition relation $\rightarrow \subseteq \Sigma \times \Sigma$. As usual in model checking, we assume that the relation $\rightarrow$ is total, i.e., for any $s \in \Sigma$ there exists some $t \in \Sigma$ such that $s{\rightarrow}t$. Hence, note that $|\Sigma| \leq |\rightarrow|$. The predecessor/successor transformers $\mathrm{pre}_{\rightarrow}, \mathrm{post}_{\rightarrow} : \wp(\Sigma) \rightarrow \wp(\Sigma)$ are defined as usual: $\mathrm{pre}_{\rightarrow}(Y) \stackrel{\mathrm{def}}{=} \{a \in \Sigma \mid \exists b \in Y.\ a{\rightarrow}b\}$ and $\mathrm{post}_{\rightarrow}(Y) \stackrel{\mathrm{def}}{=} \{b \in \Sigma \mid \exists a \in Y.\ a{\rightarrow}b\}$. 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 $\ell : \Sigma \rightarrow \wp(AP)$. For any $s \in \Sigma$, $[s]_\ell \stackrel{\mathrm{def}}{=} \{s' \in \Sigma \mid \ell(s) = \ell(s')\}$ denotes the

equivalence class of a state s w.r.t. the labeling ℓ, while $\mathrm{Part}(\Sigma) \ni P_\ell \stackrel{\text{def}}{=} \{[s]_\ell \mid s \in \Sigma\}$ is the partition induced by ℓ.

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

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 (namely, it is reflexive and transitive) called similarity 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 $\sim_{\mathrm{sim}}$.

It is a well known result in model checking [7, 10, 13] that the reduction of $\mathcal{K}$ w.r.t. simulation equivalence $\sim_{\mathrm{sim}}$ allows us to define an abstract Kripke structure that strongly preserves the temporal language ACTL^*: P_{sim} is the abstract state space, $\rightarrow^{\exists\exists}$ is the abstract transition relation between simulation equivalence classes, while a block $B \in P_{\mathrm{sim}}$ is labeled as $\ell(s)$ for any representative $s \in B$.

2.3. Abstract Interpretation

Abstract Domains. In standard abstract interpretation, abstract domains can be equivalently specified either by Galois connections/insertions or by (upper) closure operators (uco's) [6]. These two approaches are equivalent, modulo isomorphic representations of domain's objects. The closure operator approach 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 $\langle \wp(\Sigma), \subseteq \rangle$, i.e. the powerset of Σ ordered by the subset relation, plays here 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 $\langle \mathrm{uco}(\wp(\Sigma)), \sqsubseteq \rangle$ of 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. Each closure $\mu \in \mathrm{uco}(\wp(\Sigma))$ is uniquely determined by its image $\mathrm{img}(\mu) = \{\mu(X) \in \wp(\Sigma) \mid X \in \wp(\Sigma))\}$: for any $X \subseteq \Sigma$, $\mu(X) = \cap\{Y \in \mathrm{img}(\mu) \mid X \subseteq Y\}$. On the other hand, a set of subsets $\mathcal{X} \subseteq \wp(\Sigma)$ is the image of some closure on $\wp(\Sigma)$ iff $\mathcal{X}$ is closed under arbitrary intersections, i.e. $\mathcal{X} = \mathrm{Cl}_\cap(\mathcal{X}) \stackrel{\text{def}}{=} \{\cap\mathcal{S} \mid \mathcal{S} \subseteq \mathcal{X}\}$ (in particular, note that $\mathrm{Cl}_\cap(\mathcal{X})$ always contains $\Sigma = \cap\varnothing$). Also, if $\mu, \rho \in \mathrm{uco}(\wp(\Sigma))$ then $\mu \sqsubseteq \rho$ iff $\mathrm{img}(\rho) \subseteq \mathrm{img}(\mu)$. Often, we will identify closures with their sets of fixpoints since this does not give rise to ambiguity.

Partitions and Abstract Domains. Let us recall from [15] 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: $x \equiv_\mu y \Leftrightarrow \mu(\{x\}) = \mu(\{y\})$.

Example 2.1. Let $\Sigma = \{1, 2, 3, 4\}$ and let us consider the following abstract domains in $\mathrm{uco}(\wp(\Sigma))$ that are given as subsets of $\wp(\Sigma)$ closed under intersections: $\mu = \{\varnothing, \{1, 2\}, \{3\}, \{4\}, \{3, 4\}, \{1, 2, 3, 4\}\}$, $\mu' = \{\varnothing, \{1, 2\}, \{3\}, \{4\}, \{1, 2, 3, 4\}\}$, $\mu'' = \{\{1, 2\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 2, 3, 4\}\}$. These abstract domains all induce the same partition $P = \{\{1, 2\}, \{3\}, \{4\}\} \in \mathrm{Part}(\Sigma)$. For example, $\mu''(\{1\}) = \mu''(\{2\}) = \{1, 2\}$, $\mu''(\{3\}) = \{1, 2, 3\}$, $\mu''(\{4\}) = \{1, 2, 3, 4\}$ so that $\mathrm{par}(\mu'') = P$. □

Forward Completeness. Consider an abstract domain A specified by an abstraction map $\alpha : \wp(\Sigma) \rightarrow A$ and a concretization map $\gamma : A \rightarrow \wp(\Sigma)$ that define a Galois insertion of A into $\wp(\Sigma)$. Let $f : \wp(\Sigma) \rightarrow \wp(\Sigma)$ be some concrete semantic function and $f^\sharp : A \rightarrow A$ be a corresponding abstract function on A. It is well known that $\langle A, f^\sharp \rangle$ is a sound abstract interpretation when $f \circ \gamma \sqsubseteq \gamma \circ f^\sharp$ holds. Forward completeness corresponds to require the following strengthening of soundness: $\langle A, f^\sharp \rangle$ is a forward complete when $f \circ \gamma = \gamma \circ f^\sharp$. The intuition is that $f^\sharp$ is able to mimick f on the abstract domain A without loss of precision. This is called forward completeness because a dual notion of backward completeness involving the abstraction map α may also be considered (see e.g. [9]).

It turns out that the possibility of defining a forward complete abstract interpretation on a given abstract domain A does not depend on the choice of the abstract function $f^\sharp$ but depends only on the abstract domain A, namely forward completeness is an abstract domain property. This allows to formulate forward completeness independently of abstract functions as follows: an abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ is forward complete for f iff $f \circ \mu = \mu \circ f \circ \mu$. Hence, let us note that μ is forward complete for f 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 any $f \in F$.

It turns out [9, 15] that any abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ can be refined to its F-forward complete shell,

namely to the most abstract domain that is forward complete for F and refines μ. This F-forward complete shell of μ is denoted by $\mathcal{S}_F(\mu)$. Moreover, forward complete shells can be constructively characterized as greatest fixpoints of a suitable operator on the lattice $\mathrm{uco}(\wp(\Sigma))$.

Disjunctive Abstract Domains. An abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ is disjunctive (or additive) when μ preserves arbitrary unions and this happens exactly when its image $\mathrm{img}(\mu)$ is closed under arbitrary unions. The intuition is that a disjunctive abstract domain does not loose precision in approximating concrete set unions. We denote by $\mathrm{uco}^{\mathrm{d}}(\wp(\Sigma))$ the set of disjunctive abstract domains.

Given any abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$, it turns out [6] that μ can be refined to its disjunctive completion μ^{d}, namely the most abstract disjunctive domain $\mu^{\mathrm{d}} \in \mathrm{uco}^{\mathrm{d}}(\wp(\Sigma))$ that refines μ exists. Even more, the disjunctive completion μ^{d} can be obtained by closing the image $\mathrm{img}(\mu)$ under arbitrary unions, namely $\mathrm{img}(\mu^{\mathrm{d}}) = \mathrm{Cl}_{\cup}(\mathrm{img}(\mu)) \stackrel{\mathrm{def}}{=} \{\cup \mathcal{S} \mid \mathcal{S} \subseteq \mathrm{img}(\mu)\}$.

It also turns out that an abstract domain μ is disjunctive iff μ is forward complete for the concrete set union, 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, the disjunctive completion μ^{d} of μ coincides with the $\cup$-forward complete shell $\mathcal{S}_{\cup}(\mu)$ of μ.

Finally, let us recall that an abstract domain μ and its disjunctive completion μ^{d} induce the same partition, i.e. $\mathrm{par}(\mu) = \mathrm{par}(\mu^{\mathrm{d}})$.

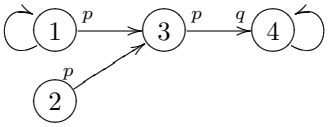
3. Simulation Equivalence as a Forward Complete Shell

Ranzato and Tapparo [15] showed how strong preservation of temporal 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. In this framework, we show that the similarity preorder can be characterized as a forward complete shell as follows. Let $\mathcal{K} = (\Sigma, \rightarrow, \ell)$ be a Kripke structure and let $\mu_\ell \stackrel{\mathrm{def}}{=} \mathrm{Cl}_{\cap}(\{[s]_\ell \mid s \in \Sigma\}) \in \mathrm{uco}(\wp(\Sigma))$ denote the abstract domain induced by the labeling ℓ.

Theorem 3.1. *Let $\mu_{\mathcal{K}} = \mathcal{S}_{\cup,\mathrm{pre}_\rightarrow}(\mu_\ell)$ be the $\{\cup, \mathrm{pre}_\rightarrow\}$-forward complete shell of μ_ℓ. Then, $R_{\mathrm{sim}} = \{(s, s') \in \Sigma^2 \mid \mu_{\mathcal{K}}(\{s'\}) \subseteq \mu_{\mathcal{K}}(\{s\})\}$. Moreover, $P_{\mathrm{sim}} = \mathrm{par}(\mu_{\mathcal{K}})$.*

Thus, simulation equivalence can be obtained as the partition induced by the forward complete shell of the initial abstract domain μ_ℓ induced by the labeling ℓ w.r.t. set union $\cup$ and the predecessor transformer $\mathrm{pre}_\rightarrow$. This result comes as a consequence of the fact that set union and $\mathrm{pre}_\rightarrow$ provide the semantics of, respectively, logical disjunction and existential next operator EX and correspondingly simulation equivalence can be viewed as the most abstract domain that strongly preserves the language $\varphi ::= atom \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \mathrm{EX}\varphi$ (see [15]). Theorem 3.1 is one key result for proving the correctness of our simulation equivalence algorithm $SimEq$ while it is not needed for understanding how $SimEq$ works and for implementing it.

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



Let us denote blocks in a partition and sets in a closure in a compact form without curly brackets and commas. It is simple to observe that $P_{\mathrm{sim}} = \{1, 2, 3, 4\}$ because: (i) while $3{\rightarrow}4$ we have that $1, 2 \notin \mathrm{pre}_\rightarrow(4)$ so that 1 and 2 are not simulation equivalent to 3; (ii) while $1{\rightarrow}1$ we have that $2 \notin \mathrm{pre}_\rightarrow(12)$ so that 1 is not simulation equivalent to 2.

The abstract domain induced by the labeling is $\mu_\ell = \{\varnothing, 123, 4, 1234\} \in \mathrm{uco}(\wp(\Sigma))$. Since the predecessor transformer $\mathrm{pre}_\rightarrow$ clearly preserves set unions, it turns out that the forward complete shell $\mathcal{S}_{\cup,\mathrm{pre}_\rightarrow}(\mu_\ell)$ can be obtained by iteratively closing the image of μ_ℓ under $\mathrm{pre}_\rightarrow$ and then by taking the disjunctive completion.

- $\mu^0 = \mu_\ell$;
- $\mu^1 = \mathrm{Cl}_{\cap}(\mu^0 \cup \mathrm{pre}_\rightarrow(\mu^0)) = \mathrm{Cl}_{\cap}(\mu^0 \cup \{\mathrm{pre}_\rightarrow(\varnothing) = \varnothing, \mathrm{pre}_\rightarrow(123) = 12, \mathrm{pre}_\rightarrow(4) = 34, \mathrm{pre}_\rightarrow(1234) = 1234\}) = \{\varnothing, 12, 3, 123, 4, 34, 1234\}$;
- $\mu^2 = \mathrm{Cl}_{\cap}(\mu^1 \cup \mathrm{pre}_\rightarrow(\mu^1)) = \mathrm{Cl}_{\cap}(\mu^1 \cup \{\mathrm{pre}_\rightarrow(12) = 1, \mathrm{pre}_\rightarrow(3) = 12, \mathrm{pre}_\rightarrow(34) = 1234\}) = \{\varnothing, 1, 12, 3, 123, 4, 34, 1234\}$;
- $\mu^3 = \mathrm{Cl}_{\cap}(\mu^2 \cup \mathrm{pre}_\rightarrow(\mu^2)) = \mu^2$ (fixpoint).

$\mathcal{S}_{\cup,\mathrm{pre}_\rightarrow}(\mu_\ell)$ is thus given by the disjunctive completion of μ^2, i.e., $\mathcal{S}_{\cup,\mathrm{pre}_\rightarrow}(\mu_\ell) = \{\varnothing, 1, 3, 4, 12, 13, 14, 34, 123, 124, 134, 1234\} = \mu_{\mathcal{K}}$. 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 similarity preorder is $R_{\mathrm{sim}} = \{(1,1), (2,2), (2,1), (3,3), (4,4)\}$, while, as expected, $P_{\mathrm{sim}} = \mathrm{par}(\mathcal{S}_{\cup,\mathrm{pre}_\rightarrow}(\mu_\ell)) = \{1, 2, 3, 4\}$. □

4. Partition/Relation Pairs

Let $P \in \mathrm{Part}(\Sigma)$ be a partition and let $R \subseteq P \times P$ be any relation on P. A pair $\langle P, R\rangle$ is called a *partition/relation pair*. We use the following notation: if $B \in P$ then $R(B) \stackrel{\mathrm{def}}{=} \{C \in P \mid (B, C) \in R\}$.

A partition/relation pair $\langle P, R\rangle$ induces a disjunctive closure $\mu_{\langle P,R\rangle} \in \mathrm{uco}^{\mathrm{d}}(\wp(\Sigma))$ as follows: for any $X \in \wp(\Sigma)$,

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

where $R^* \subseteq P \times P$ is the reflexive-transitive closure of 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 equivalence algorithm: in fact, our algorithm maintains a partition/relation pair (with a reflexive relation) and its correctness depends on the fact that this partition/relation pair logically represents a corresponding disjunctive abstract domain. It is not hard to observe that $P \preceq \mathrm{par}(\mu_{\langle P,R\rangle})$ and that if $\langle P, R\rangle$ is a partition/preorder pair then $P = \mathrm{par}(\mu_{\langle P,R\rangle})$.

Example 4.1. Let $\Sigma = \{1,2,3,4\}$, $P = \{12,3,4\} \in \mathrm{Part}(\Sigma)$ (where blocks are denoted without curly brackets and commas) and $R = \{(12,12),(12,3),(3,3),(3,4),(4,4)\}$. The disjunctive abstract domain $\mu_{\langle P,R\rangle}$ is such that $\mu_{\langle P,R\rangle}(\{1\}) = \mu_{\langle P,R\rangle}(\{2\}) = \{1,2,3,4\}$, $\mu_{\langle P,R\rangle}(\{3\}) = \{3,4\}$ and $\mu_{\langle P,R\rangle}(\{4\}) = \{4\}$, so that the image of $\mu_{\langle P,R\rangle}$ is $\{\varnothing, \{4\}, \{3,4\}, \{1,2,3,4\}\}$. □

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:

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

It is not hard to observe that both μ and its disjunctive completion μ^{d} induce the same partition/relation pair, i.e. $\langle P_\mu, R_\mu\rangle = \langle P_{\mu^{\mathrm{d}}}, R_{\mu^{\mathrm{d}}}\rangle$.

Our simulation equivalence algorithm relies on the following key condition on a partition/relation pair $\langle P, R\rangle$ w.r.t. a transition system $\langle \Sigma, \rightarrow\rangle$ which guarantees that the corresponding disjunctive abstract domain $\mu_{\langle P,R\rangle}$ is forward complete for the predecessor transformer $\mathrm{pre}_\rightarrow$.

Lemma 4.2. *Let $(\Sigma, \rightarrow)$ be a transition system. Let $\langle P, R\rangle$ be a partition/relation pair with R reflexive. Assume that for any $B, C \in P$, if $C \cap \mathrm{pre}_\rightarrow(B) \neq \varnothing$ then $\cup R(C) \subseteq \mathrm{pre}_\rightarrow(\cup R(B))$. Then, $\mu_{\langle P,R\rangle}$ is forward complete for* $\mathrm{pre}_\rightarrow$.

5. Henzinger, Henzinger and Kopke's Algorithm

Our simulation equivalence algorithm *SimEquiv* is designed as a modification of Henzinger, Henzinger and Kopke's [11] simulation equivalence algorithm HHK. While HHK maintains for any state a corresponding set of states, *SimEquiv* maintains instead a set of states for all the blocks of a state partition. The HHK algorithm is designed in three incremental steps. The second and third steps are the procedures *RefinedSimilarity* and *EfficientSimilarity* that are recalled in Figure 1.

The idea of the basic HHK algorithm (that is not recalled in Figure 1) is as follows. For each state $v \in \Sigma$, the 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. 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 sharpened to $Sim(u) \smallsetminus \{w\}$.

This simple algorithm is refined by the procedure *RefinedSimilarity* in Figure 1. The key point here is to store for each state $v \in \Sigma$ a further set $prevSim(v)$ that is a superset of $Sim(v)$ (invariant Inv_1) and contains the states that were in $Sim(v)$ in some past iteration. If $u \rightarrow v$ then the invariant Inv_2 allows to sharpen $Sim(u)$ by scrutinizing only the states in $\mathrm{pre}_\rightarrow(prevSim(v))$ instead of all the possible states in Σ. Let us remark that the original *RefinedSimilarity* algorithm appeared in [11] 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 version of *RefinedSimilarity* appeared in [11] 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 $P_{\mathrm{sim}} = \{\{1\},\{2\},\{3\},\{4\}\}$. However, one can check that the original version in [11] of the *RefinedSimilarity* algorithm — 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 states 1 and 2 appear to be simulation equivalent while they are not. The problem with the original version in [11] of the *RefinedSimilarity* algorithm lies in the fact that when $v \in \mathrm{pre}_\rightarrow(\{v\})$ — like in this example — it may happen that during the inner for-loop the set $Sim(v)$ is modified 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 *EfficientSimilarity* algorithm recalled in Figure 1 (the original version of *EfficientSimilarity* in [11] also suffers of a bug that is a direct consequence of the problem in *RefinedSimilarity* explained above). The idea here is that instead of recomputing at each iteration of the while-loop the set $Remove := \mathrm{pre}_\rightarrow(prevSim(v)) \smallsetminus \mathrm{pre}_\rightarrow(Sim(v))$ 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 implementation exploits a matrix $Count(u,v)$, indexed on states $u, v \in \Sigma$, such that $Count(u,v) = |\mathrm{post}_\rightarrow(u) \cap Sim(v)|$ so that the test $w'' \notin \mathrm{pre}_\rightarrow(Rel(u))$ in the innermost for-loop can be done in $O(1)$. This allows an efficient implementation of

```
RefinedSimilarity() {
   for all v in Σ do {prevSim(v) := Σ; Sim(v) := [v]_ℓ; }
   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);
      for all u ∈ pre_→(v) do Sim(u) := Sim(u) ∖ Remove;
   }
}
```

```
EfficientSimilarity() {
   /* for all v in Σ do prevSim(v) := Σ; */
   for all v in Σ do {Sim(v) := [v]_ℓ; Remove(v) := 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) := ∅;
      for all u ∈ pre_→(v) do
         for all w ∈ Remove do
            if (w ∈ Sim(u)) then {
               Sim(u) := Sim(u) ∖ {w};
               for all w'' ∈ pre_→(w) such that (w'' ∉ pre_→(Sim(u))) do
                  Remove(u) := Remove(u) ∪ {w''};
            }
   }
}
```

Figure 1. Henzinger, Henzinger, Kopke's Algorithm.

EfficientSimilarity that runs in $O(|\Sigma||{\to}|)$ time, while the space complexity is clearly quadratic in the size of the state space Σ. Let us remark that the key property for showing the $O(|\Sigma||{\to}|)$ time bound is as follows: if a state v is selected at some iterations i and j of the while-loop with $i < j$ 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 Equivalence Algorithm

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 similarity preorder $R_{\rm sim}$ is obtained from the output $\{Sim(s)\}_{s\in\Sigma}$ as follows: $(s, s') \in R_{\rm sim}$ iff $s' \in Sim(s)$. In turn, the simulation equivalence partition $P_{\rm sim}$ 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 $\langle P, Rel\rangle$ instead 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 $s \in \Sigma$, $s_1 \in Sim(s) \Leftrightarrow s_2 \in Sim(s)$. A reflexive relation $Rel \subseteq P \times P$ gives rise to a partition/relation pair where the intuition is that if $B_1, B_2 \in P$, $(B_1, B_2) \in Rel$ and $s_i \in B_i$ then the simulator set $Sim(s_2)$ is a subset of the simulator set $Sim(s_1)$. Since each $Sim(s)$ always contains s, in this case we have that $s_2 \in Sim(s_1)$, namely s_2 is a current candidate for simulating s_1. In particular, if $B \in P$ and $s, s' \in B$ then s and s' are currently candidates to be simulation equivalent. Thus, the partition/relation pair $\langle P, Rel\rangle$ represents the current approximation of the similarity preorder and in particular P represents the current approximation of simulation equivalence.

The partition P is initialized to the partition P_ℓ induced by the labeling ℓ and is always maintained coarser than the final partition $P_{\rm sim}$. The idea of maintaining a partition/relation pair is also used by Gentilini et al. [8] in their

```
SimEquiv_1(PartitionRelation ⟨P, Rel⟩) {
  while ∃B, C ∈ P such that (C ∩ pre_→(B) ≠ ∅ & ∪Rel(C) ⊈ pre_→(∪Rel(B))) do {
    S := pre_→(∪Rel(B));
    P_prev := P; B_prev := B;
    P := Split(P, S);
    for all C in P ∖ P_prev do Rel(C) := {D ∈ P | D ⊆ ∪Rel(parent_{P_prev}(C))};
    for all C in P such that (C ∩ pre_→(B_prev) ≠ ∅) do Rel(C) := {D ∈ Rel(C) | D ⊆ S};
  }
}
```

Figure 2. Basic Simulation Equivalence Algorithm.

simulation equivalence algorithm. However, the main differences are that (i) by exploiting the results in Section 4, we logically view a partition/relation pair as an abstract domain and (ii) we follow the idea of the HHK procedure of maintaining a remove set to be used for refining the current partition/relation pair.

Following Henzinger et al. [11], we design our simulation equivalence algorithm in three incremental steps. The basic algorithm, called $SimEquiv_1$, is described in Figure 2. By Theorem 3.1, the goal of our algorithm is to compute the $\{\cup, \mathrm{pre}_\rightarrow\}$-forward complete shell of an initial abstract domain through incremental refinements. By Section 4, we know that partition/relation pairs can be viewed as representing disjunctive abstract domains while Lemma 4.2 gives us a condition on a partition/relation pair which guarantees that the corresponding abstract domain is forward complete for $\mathrm{pre}_\rightarrow$. Moreover, this abstract domain is disjunctive as well, being induced by a partition/relation pair. Hence, the idea consists in iteratively and minimally refining an initial partition/relation pair $\langle P, Rel\rangle$ induced by the labeling of a Kripke structure until the condition of Lemma 4.2 is satisfied: for all $B, C \in P$,

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

Note that $C \cap \mathrm{pre}_\rightarrow(B) \neq \varnothing$ holds iff $C \rightarrow^{\exists\exists} B$. The current partition/relation pair $\langle P, Rel\rangle$ is refined by the three following basic steps in $SimEquiv_1$. If B is the block of the current partition selected by the while-loop then:

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

(ii) if C is a newly generated block after splitting P and $\mathrm{parent}_{P_{\mathrm{prev}}}(C)$ is its parent block in the previous partition P_{prev} before the splitting operation then $Rel(C)$ is modified so 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.

For any abstract domain $\mu \in \mathrm{uco}(\wp(\Sigma))$, we write $\mu' = SimEquiv_1(\mu)$ when the algorithm $SimEquiv_1$ on input the partition/relation $\langle P_\mu, R_\mu\rangle$ terminates and outputs a partition/relation pair $\langle P', R'\rangle$ such that $\mu' = \mu_{\langle P', R'\rangle}$. Then, the correctness of $SimEquiv_1$ is as follows.

Theorem 6.1. *Let Σ be finite. Then, $SimEquiv_1$ terminates on any input domain $\mu \in \mathrm{uco}(\wp(\Sigma))$ and $SimEquiv_1(\mu) = \mathcal{S}_{\cup,\mathrm{pre}_\rightarrow}(\mu)$.*

Thus, $SimEquiv_1$ correctly computes the $\{\cup, \mathrm{pre}_\rightarrow\}$-forward complete shell of any input abstract domain. As a particular case, $SimEquiv_1$ allows us to compute simulation equivalence.

Corollary 6.2. *Let $\mathcal{K} = (\Sigma, \rightarrow, \ell)$ be a finite Kripke structure and $\mu_\ell \in \mathrm{uco}(\wp(\Sigma))$ be the abstract domain induced by ℓ. Then, $SimEquiv_1(\mu_\ell) = \langle P', R'\rangle$ where $P' = P_{\mathrm{sim}}$.*

The $SimEquiv_1$ algorithm is refined to $SimEquiv_2$ as described in Figure 3. This is obtained by adapting the ideas of the Henzinger et al.'s *RefinedSimilarity* procedure in Figure 1 to our $SimEquiv_1$ algorithm. The following points explain why this refined algorithm $SimEquiv_2$ remains correct.

- For any block B of the current partition P, we also maintain the "previous" relation $prevRel(B)$. This means that initially $prevRel(B)$ is set to contain all the blocks. Then, when a block B is selected by the while-loop at some iteration i, $prevRel(B)$ is updated to store the current relation $Rel(B)$ of 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 $prevRel(C)$ is set such that $\cup prevRel(C) = \cup prevRel(\mathrm{parent}_{P_{\mathrm{prev}}}(C))$.
- Therefore, since the current relation decreases only — i.e., if i and j are iterations such that j follows i and $B' \subseteq B$ then $\cup Rel_j(B') \subseteq \cup Rel_i(B)$ — at each iteration, the following invariant Inv_1 holds: for any block $B \in P$, $\cup Rel(B) \subseteq \cup prevRel(B)$.

```
SimEquiv_2(PartitionRelation ⟨P, Rel⟩) {
  for all B in P do prevRel(B) := P;
  while ∃B ∈ P such that (pre_→(∪Rel(B)) ≠ pre_→(∪prevRel(B))) do {
    /* Inv_1: ∀B ∈ P. ∪Rel(B) ⊆ ∪prevRel(B) */
    /* Inv_2: ∀B, C ∈ P. C ∩ pre_→(B) ≠ ∅ ⇒ ∪Rel(C) ⊆ pre_→(∪prevRel(B)) */
    S := pre_→(∪Rel(B)); Remove := pre_→(∪prevRel(B)) ∖ S;
    prevRel(B) := Rel(B);
    P_prev := P; B_prev := B;
    P := Split(P, S);
    for all C in P do {
      Rel(C) := {D ∈ P | D ⊆ ∪Rel(parent_{P_prev}(C))};
      prevRel(C) := {D ∈ P | D ⊆ ∪prevRel(parent_{P_prev}(C))};
    }
    for all C in P such that (C ∩ pre_→(B_prev) ≠ ∅) do Rel(C) := {D ∈ Rel(C) | D ∩ Remove = ∅};
  }
}
```

Figure 3. Refined Simulation Equivalence Algorithm.

– The crucial point is the invariant Inv_2: if $C \rightarrow^{\exists\exists} B$ and $D \in Rel(C)$ then $D \subseteq \mathrm{pre}_{\rightarrow}(\cup\, prevRel(B))$. This property is initially true because at the beginning, for each block B, $prevRel(B)$ is set to P. Morever, Inv_2 is maintained at each iteration because $Remove$ is set to $\mathrm{pre}_{\rightarrow}(\cup prevRel(B)) \setminus \mathrm{pre}_{\rightarrow}(\cup Rel(B))$ 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 $Rel(C)$.

– Finally, let us remark that the exit condition of the while-loop, namely $\forall B \in P.\ \mathrm{pre}_{\rightarrow}(\cup Rel(B)) = \mathrm{pre}_{\rightarrow}(\cup prevRel(B))$, is weaker than the exit condition that we would obtain as counterpart of the exit condition of the while-loop of *RefinedSimilarity*, i.e. $\forall B \in P.\ Rel(B) = prevRel(B)$.

If the exit condition of the while-loop of $SimEquiv_2$ is satisfied then, by Inv_2, the exit condition of $SimEquiv_1$ is satisfied as well.

6.1. The Final Algorithm

Following the underyling idea of the refinement of *RefinedSimilarity* to *EfficientSimilarity*, the algorithm $SimEquiv_2$ is further refined to its final version $SimEquiv_3$ in Figure 4. The idea is that instead of recomputing at each iteration of the while-loop the set $Remove := \mathrm{pre}_{\rightarrow}(\cup prevRel(B)) \setminus \mathrm{pre}_{\rightarrow}(\cup Rel(B))$ for the selected block B, we maintain a set of states $Remove(B) \subseteq \Sigma$ for each block B of the current partition. For any block C, $Remove(C)$ is updated in order to satisfy the invariant condition Inv_3: $Remove(C)$ contains exactly the set of states that belong to $\mathrm{pre}_{\rightarrow}(\cup prevRel(C))$ but are not in $\mathrm{pre}_{\rightarrow}(\cup Rel(C))$, where the previous relation $prevRel(C)$ is logically defined as in $SimEquiv_2$, but is not really stored, Moreover, the invariant condition Inv_4 ensures that, for any block C, $\mathrm{pre}_{\rightarrow}(\cup\, prevRel(C))$ is a union of blocks of the current partition. This property allows us to replace the split operation $Split(P, \mathrm{pre}_{\rightarrow}(\cup\, Rel(B)))$ in $SimEquiv_2$ with the equivalent split operation $Split(P, \mathrm{pre}_{\rightarrow}(\cup\, prevRel(B)) \setminus \mathrm{pre}_{\rightarrow}(\cup\, Rel(B)))$. The correctness of such replacement is obtained from invariant Inv_4 by exploiting the following result.

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

The correctness of $SimEquiv_3$ is a consequence of the following observations.

– When a block B_{prev} of the current partition is selected by the while-loop the corresponding remove set $Remove(B_{\mathrm{prev}})$ is set to empty. The invariant Inv_3 is maintained at each iteration because for any block C such that $C \rightarrow^{\exists\exists} B_{\mathrm{prev}}$ the for-loop at line 20 incrementally adds to $Remove(C)$ all the states s that are in $\mathrm{pre}_{\rightarrow}(\cup\, prevRel(C))$ but not in $\mathrm{pre}_{\rightarrow}(\cup\, Rel(C))$.

– 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 $Remove(C)$ is set to $Remove(\mathrm{parent}_{P_{\mathrm{prev}}}(C))$.

– As in $SimEquiv_2$, for any block C such that $C \rightarrow^{\exists\exists} B_{\mathrm{prev}}$, all the blocks that are contained in $Remove(C)$ are removed from $Rel(C)$.

```
1  SimEquiv_3(PartitionRelation ⟨P, Rel⟩) {
2     /* for all B in P do prevRel(B) := P; */
3     for all B in P do Remove(B) := Σ ∖ pre_→(∪Rel(B));
4     while ∃B ∈ P such that (Remove(B) ≠ ∅) do {
5        /* Inv_3: ∀C ∈ P. Remove(C) = pre_→(∪prevRel(C)) ∖ pre_→(∪Rel(C)) */
6        /* Inv_4: ∀C ∈ P. Split(P, pre_→(∪prevRel(C))) = P */
7        /* prevRel(B) := Rel(B); */
8        Remove := Remove(B);
9        Remove(B) := ∅;
10       P_prev := P; B_prev := B;
11       P := Split(P, Remove);
12       for all C in P do {
13          Rel(C) := {D ∈ P | D ⊆ ∪Rel(parent_{P_prev}(C))};
14          Remove(C) := Remove(parent_{P_prev}(C));
15       }
16       for all C in P such that (C ∩ pre_→(B_prev) ≠ ∅) do
17          for all D in P such that (D ⊆ Remove) do
18             if (D ∈ Rel(C)) then {
19                Rel(C) := Rel(C) ∖ {D};
20                for all s ∈ pre_→(D) such that (s ∉ pre_→(∪Rel(C))) do Remove(C) := Remove(C) ∪ {s};
21             }
22    }
23 }
```

Figure 4. Efficient Simulation Equivalence Algorithm.

If the exit condition of the while-loop of $SimEquiv_3$ is satisfied then, by Inv_1 and Inv_3, the exit condition of $SimEquiv_2$ is satisfied as well.

6.2. Time Complexity

As far as time complexity is concerned, we exploit the following key property of $SimEquiv_3$. Let $B \in P_{\mathrm{in}}$ be some block of the initial partition and let $\langle B_i \rangle_{i \in It}$ be a sequence of blocks selected by the while-loop in a sequence of iterations It such that (a) for any $i \in It$, $B_i \subseteq B$ and (b) if an iteration $j \in It$ follows an iteration $i \in It$, i.e. $i < j$, 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, it turns out that the corresponding 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 $D \subseteq Remove(B_i)$ at line 17 is positive at some iteration $i \in It$ then for all the blocks $D' \subseteq D$ and for any successive iteration $j > i$, with $j \in It$, the test $D' \subseteq Remove(B_j)$ will be negative. Moreover, if the test $D \in Rel(C)$ at line 18 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 $j > i$. Let us also observe that the overall number of newly generated blocks by the splitting operation at line 11 is exactly given by $2(|P_{\mathrm{sim}}| - |P_{\mathrm{in}}|)$. From these observations, it is not hard to obtain that the overall time complexity of the code of the for-loops at lines 16-17 is $\sum_B \sum_D \sum_{b \in B} |\, \mathrm{pre}_{\rightarrow}(\{b\})| \leq 2|P_{\mathrm{sim}}||{\rightarrow}|$. Also, the overall time complexity of the code of the if-then statement at lines 18-21 is $\sum_C \sum_D \sum_{d \in D} |\, \mathrm{pre}_{\rightarrow}(\{d\})| \leq 2|P_{\mathrm{sim}}||{\rightarrow}|$. Furthermore, as described below, we use suitable resizable data structures that allow us to execute the tests $D \in Rel(C)$ at line 18 and $s \notin \mathrm{pre}_{\rightarrow}(\cup\, Rel(C))$ at line 20 in $O(1)$ time. A splitting operation $Split(P, Remove)$ can be executed in $O(|\, Remove\,|)$-time so that, by the above observation on the remove sets, the overall cost of all the splitting operations is in $O(|P_{\mathrm{sim}}||\Sigma|)$-time. These properties allow us to show that the total running time is in $O(|P_{\mathrm{sim}}||{\rightarrow}|)$.

Theorem 6.4. *The simulation equivalence algorithm $SimEquiv_3$ runs in $O(|P_{\mathrm{sim}}||{\rightarrow}|)$-time and $O(|P_{\mathrm{sim}}||\Sigma|)$-space.*

6.3. Data Structures

Our implementation uses the following data structures.

– Each state $s \in \Sigma$ (represented as an integer) stores the list of its predecessors in $\mathrm{pre}_{\rightarrow}(\{s\})$. This provides a representation of the input transition system.

– A block B of the current partition is represented by a record that contains a list of of pointers to the states in B. Moreover, any block B stores its corresponding remove set $B.Remove$.

– Any block B also stores two integer arrays indexed over Σ: for any $x \in \Sigma$, $B.BlockCount(x) = |\{(x,y) \mid x{\rightarrow}y,\ y \in B\}|$ while $B.RelCount(x) = \sum_{C \in Rel(B)} C.BlockCount(x)$. The array $RelCount$ allows to implement the test $s \notin \mathrm{pre}_{\rightarrow}(\cup\, Rel(C))$ at line 20 as $C.RelCount(s) = 0$ so that it takes constant time. The array $BlockCount$ is needed to maintain efficiently the array $RelCount$.

– The current partition is stored as a doubly linked list P of blocks. Newly generated blocks are appended to this list. Blocks are scanned from the beginning of this list by performing the test 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.

– The current relation Rel on the current partition P is stored as a resizable $|P| \times |P|$ boolean matrix. The algorithm adds new entries to this matrix as long as a new block is generated from a splitting operation. Hence, the total number of insert operations in the matrix Rel is $|P_{\mathrm{sim}}| - |P_{\mathrm{in}}|$. 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 this matrix is in $O(|P_{\mathrm{sim}}|^2)$-time.

7. Conclusion

We presented a new efficient algorithm for computing simulation equivalence in $O(|P_{\mathrm{sim}}||{\rightarrow}|)$-time and $O(|P_{\mathrm{sim}}||\Sigma|)$-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 [11] and by Bloom and Paige's [1] algorithms that however suffer from a quadratic space complexity that is bounded from below by $\Omega(|\Sigma|^2)$. A better space bound is given by Gentilini et al.'s [8] algorithm whose space complexity is in $O(|P_{\mathrm{sim}}|^2 + |\Sigma| \log(|P_{\mathrm{sim}}|))$, but that runs in $O(|P_{\mathrm{sim}}|^2|{\rightarrow}|)$-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 conduct an experimental evaluation of our algorithm and to compare its performance with the existing implementations of Gentilini et al.'s [8] and Tan and Cleaveland's [16] algorithms. It is also definitely interesting to investigate whether this new efficient algorithm may admit a symbolic version based on BDDs.

References

[1] 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.

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

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

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

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

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

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

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

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

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

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

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

[13] 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.

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

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

[16] L. Tan and R. Cleaveland. Simulation revisited. In *Proc. 7th TACAS*, LNCS 2031:480-495, 2001.