# Probabilistic Bisimulation and Simulation Algorithms by Abstract Interpretation

SILVIA CRAFA FRANCESCO RANZATO

University of Padova, Italy

**Abstract.** We show how bisimulation equivalence and simulation preorder on probabilistic LTSs (PLTSs), namely the main behavioural relations on probabilistic nondeterministic processes, can be characterized by abstract interpretation. Both bisimulation and simulation can be obtained as completions of partitions and preorders, viewed as abstract domains, w.r.t. a pair of concrete functions that encode a PLTS. As a consequence, this approach provides a general framework for designing algorithms for computing bisimulation and simulation on PLTSs. Notably, (i) we show that the standard bisimulation algorithm by Baier et al. can be viewed as an instance of such a framework and (ii) we design a new efficient simulation algorithm that improves the state of the art.

## 1 Introduction

Randomization phenomena in concurrent systems have been widely studied in probabilistic extensions of process algebras like Markov chains and probabilistic labeled transition systems (PLTSs). Most standard tools for studying nonprobabilistic processes, like behavioural equivalences, temporal logics and model checking, have been investigated for these probabilistic models. In particular, bisimulation equivalence and simulation preorder relations, namely the main behavioural relations between concurrent systems, have been extended and studied in a probabilistic setting [5,7,12].

Abstract interpretation [2,3] is a well-known general theory for specifying the approximation of formal semantics. Abstract domains play an essential role in any abstract interpretation design, since they encode in an ordered structure how concrete semantic properties are approximated. A number of behavioural relations, including bisimulation, stuttering bisimulation and simulation, have been characterized in abstract interpretation as complete refinements, called forward complete shells, of abstract domains w.r.t. logical/temporal operators of suitable modal logics [11]. One notable benefit of this approach is that it provides a general framework for designing basic algorithms that compute behavioral relations as forward complete shells of abstract domains. As a remarkable example, this abstract interpretation-based approach led to an efficient algorithm for computing the simulation preorder [10] that features the best time complexity among the simulation algorithms.

This paper extends the aforementioned results to a probabilistic setting. In particular, we consider probabilistic processes specified as PLTSs, a general model that exhibits both non-deterministic choice (as in LTSs) and probabilistic choice (as in Markov chains). In [11], bisimulation in LTSs has been characterized in terms of forward complete shells of partitions w.r.t. the predecessor operator of LTSs. We show that this same

idea scales to the case of PLTSs by considering the probabilistic predecessor operator that defines the transitions of a PLTS together with a probabilistic function that encodes the distributions in the PLTS (this latter operator is somehow reminiscent of a probabilistic connective in Parma and Segala's [9] modal logics for probabilistic bisimulation and simulation). Bisimulation equivalence in PLTSs is then characterized as a domain refinement through a complete shell w.r.t. the above two operators. On the other hand, the simulation preorder in PLTSs turns out to be the same complete shell of abstract domains w.r.t. the same two operators, but using different underlying abstract domains: for bisimulation, the complete shell is computed in a space of abstractions that are state and distribution partitions, while for simulation the same complete shell is instead computed over abstractions that are preorders on states and distributions.

Complete shells of abstract domains may in general be obtained through a simple fixpoint computation. We show how such a basic procedure can be instantiated to obtain two algorithms that iteratively compute bisimulation and simulation on PLTSs. Interestingly, the standard procedure for computing bisimulations in PLTSs, namely Baier-Engelen-Majster's algorithm [1], can be actually viewed as an implementation of our complete shell procedure that characterizes bisimulation. On the other hand, we show that the corresponding complete shell for computing the simulation preorder yields a new efficient probabilistic simulation algorithm that advances the state-of-the-art: in fact, its time and space complexity bounds improve on the best known simulation algorithm for PLTSs by Zhang et al. [13].

## 2 Bisimulation and Simulation in PLTSs

Given a set $X$, $\mathrm{Distr}(X)$ denotes the set of (stochastic) distributions on $X$, i.e., functions $d{:}X \to [0,1]$ such that $\sum_{x\in X} d(x) = 1$. The support of a distribution $d$ is defined by $\mathrm{supp}(d) \triangleq \{x \in X \mid d(x) > 0\}$; also, if $S \subseteq X$, then $d(S) \triangleq \sum_{s\in S} d(s)$.

A probabilistic LTS (PLTS) is a tuple $\mathcal{S} = \langle \Sigma, Act, \to\rangle$ where $\Sigma$ is a set of states, $Act$ is a set of actions and $\to \subseteq \Sigma \times Act \times \mathrm{Distr}(\Sigma)$ is a transition relation, where $(s,a,d) \in \to$ is also denoted by $s \xrightarrow{a} d$. We denote by $\mathrm{Distr} \triangleq \{d \in \mathrm{Distr}(\Sigma) \mid \exists s \in \Sigma.\exists a \in Act\,.\, s \xrightarrow{a} d\}$ the set of target distributions in $\mathcal{S}$. Given $D \subseteq \mathrm{Distr}$, we write $s \xrightarrow{a} D$ when there exists $d \in D$ such that $s \xrightarrow{a} d$. For any $a \in Act$, the predecessor and successor operators $\mathrm{pre}_a : \wp(\mathrm{Distr}) \to \wp(\Sigma)$ and $\mathrm{post}_a : \wp(\Sigma) \to \wp(\mathrm{Distr})$ are defined by $\mathrm{pre}_a(D) \triangleq \{s \in \Sigma \mid s \xrightarrow{a} D\}$ and $\mathrm{post}_a(S) \triangleq \{d \in \mathrm{Distr} \mid \exists s \in S.\, s \xrightarrow{a} d\}$. For any $d \in \mathrm{Distr}$ and $s \in \Sigma$, we define $\mathrm{in}(d) \triangleq \{a \in Act \mid \mathrm{pre}_a(d) \neq \varnothing\}$ and $\mathrm{out}(s) \triangleq \{a \in Act \mid \mathrm{post}_a(s) \neq \varnothing\}$.

***Bisimulation.*** Let $\mathrm{Part}(X)$ denote the set of partitions of a finite set $X$. If $P \in \mathrm{Part}(X)$ and $x \in X$ then $P(x)$ denotes the unique block of $P$ that contains $x$. A partition $P$ induces a mapping $P : \wp(X) \to \wp(X)$ defined as $P(Y) \triangleq \cup_{y\in Y} P(y)$. Any partition $P \in \mathrm{Part}(X)$ induces an equivalence relation (i.e., a partition) over distributions $\equiv_P \in \mathrm{Part}(\mathrm{Distr}(X))$ as follows: for any $d, e \in \mathrm{Distr}(X)$, $d \equiv_P e$ if for any $B \in P$, $d(B) = e(B)$. In words, two distributions are $\equiv_P$-equivalent whenever they give the same probability to the blocks of $P$.

Given a PLTS $\mathbb{S} = \langle \Sigma, Act, \rightarrow \rangle$, a partition $P \in \mathrm{Part}(\Sigma)$ is a bisimulation on $\mathbb{S}$ when for all $s, t \in \Sigma$ and $d \in \mathrm{Distr}$, if $P(s) = P(t)$ and $s \xrightarrow{a} d$ then there exists $e \in \mathrm{Distr}$ such that $t \xrightarrow{a} e$ and $d \equiv_P e$. Bisimilarity $P_{\mathrm{bis}} \in \mathrm{Part}(\Sigma)$ is defined as: $P_{\mathrm{bis}}(s) \triangleq \cup\{P(s) \mid P \text{ is a bisimulation on } \mathbb{S}\}$. $P_{\mathrm{bis}}$ turns out to be the greatest bisimulation on $\mathbb{S}$ which is also called the bisimulation partition on $\mathbb{S}$.

***Simulation.*** Let $\mathrm{PreOrd}(X)$ denote the set of preorders on $X$. If $R \in \mathrm{PreOrd}(X)$ and $S \subseteq X$ then $R(S) \triangleq \{x \in X \mid \exists s \in S.\, (s, x) \in R\}$. Similarly to the case of partitions, any preorder $R \in \mathrm{PreOrd}(X)$ induces a preorder $\leq_R$ on $\mathrm{Distr}(X)$ as follows: for any $d, e \in \mathrm{Distr}(X)$, $d \leq_R e$ if for any $S \subseteq X$, $d(S) \leq e(R(S))$. Such a definition of $\leq_R$ can be equivalently stated in terms of so-called weight functions between distributions and of maximum flows between networks. In particular, it turns out that $d \leq_R e$ iff the maximum flow of a suitable bipartite network built over the states in $\mathrm{supp}(d) \cup \mathrm{supp}(e)$ and over the relation $R$ is 1 (see [1,13]).

A preorder $R \in \mathrm{PreOrd}(\Sigma)$ is a simulation on a PLTS $\mathbb{S}$ when for all $s, t \in \Sigma$ and $d \in \mathrm{Distr}$, if $t \in R(s)$ and $s \xrightarrow{a} d$ then there exists $e \in \mathrm{Distr}$ such that $t \xrightarrow{a} e$ and $d \leq_R e$. The simulation preorder $R_{\mathrm{sim}} \in \mathrm{PreOrd}(\Sigma)$ on $\mathbb{S}$ is defined as follows: $R_{\mathrm{sim}}(s) \triangleq \cup\{R(s) \mid R \text{ is a simulation on } \mathbb{S}\}$. It turns out that $R_{\mathrm{sim}}$ is the greatest simulation preorder on $\mathbb{S}$. Simulation partition $P_{\mathrm{psim}}$ on $\mathbb{S}$ is the kernel of the simulation preorder, i.e., $P_{\mathrm{psim}}(s) = P_{\mathrm{psim}}(t)$ iff $s \in R_{\mathrm{sim}}(t)$ and $t \in R_{\mathrm{sim}}(s)$.

## 3 Shells

***Forward Completeness.*** In standard abstract interpretation [2,3], approximations of a concrete semantic domain are encoded by abstract domains (or abstractions), that are specified by Galois insertions (GIs for short) or, equivalently, by adjunctions. A GI of an abstract domain $\langle A, \leq_A \rangle$ into a concrete domain $\langle C, \leq_C \rangle$ is determined by a surjective abstraction map $\alpha : C \rightarrow A$ and a 1-1 concretization map $\gamma : A \rightarrow C$ such that $\alpha(c) \leq_A a \Leftrightarrow c \leq_C \gamma(a)$, and is denoted by $(\alpha, C, A, \gamma)$. Recall that GIs of a common concrete domain $C$ are preordered w.r.t. their relative precision: $\mathcal{G}_1 = (\alpha_1, C, A_1, \gamma_1) \trianglelefteq \mathcal{G}_2 = (\alpha_2, C, A_2, \gamma_2)$ — i.e. $A_1$/$A_2$ is a refinement/simplification of $A_2$/$A_1$ — iff $\gamma_1 \circ \alpha_1 \sqsubseteq_{C \rightarrow C} \gamma_2 \circ \alpha_2$. Moreover, $\mathcal{G}_1$ and $\mathcal{G}_2$ are equivalent when $\mathcal{G}_1 \trianglelefteq \mathcal{G}_2$ and $\mathcal{G}_2 \trianglelefteq \mathcal{G}_1$. We denote by $\mathrm{Abs}(C)$ the family of abstract domains of $C$ up to the above equivalence. It is well known that $\langle \mathrm{Abs}(C), \trianglelefteq \rangle$ is a complete lattice. Given a family of abstract domains $\mathcal{X} \subseteq \mathrm{Abs}(C)$, their lub $\sqcup\mathcal{X}$ is the most precise domain in $\mathrm{Abs}(C)$ which is a simplification of any domain in $\mathcal{X}$.

Let $f : C \rightarrow D$ be some concrete semantic function, and let $A \in \mathrm{Abs}(C)$ and $B \in \mathrm{Abs}(D)$ be abstractions of the concrete domains $C$ and $D$. Given an abstract function $f^\sharp : A \rightarrow B$, we have that $\langle A, B, f^\sharp \rangle$ is a sound abstract interpretation of $f$ when $f \circ \gamma_{A,C} \sqsubseteq_{A \rightarrow D} \gamma_{B,D} \circ f^\sharp$. Forward completeness [3,4] corresponds to the following strengthening of soundness: $f \circ \gamma_{A,C} = \gamma_{B,D} \circ f^\sharp$, meaning that the abstract function $f^\sharp$ is able to replicate the behaviour of $f$ on the abstract domains $A$ and $B$ with no loss of precision. It turns out that if $\langle A, B, f^\sharp \rangle$ is forward complete then the abstract function $f^\sharp$ indeed coincides with $\alpha_{D,B} \circ f \circ \gamma_{A,C}$, that is the best correct approximation of the concrete function $f$ on the pair of abstractions $\langle A, B \rangle$. Hence, the notion of forward

```
ShellAlgo(F, G, A, B) {
  Initialize();
  while ¬(F-Stable ∧ G-Stable) do
    if ¬F-Stable then G-Stable := Stabilize(F, A, B); F-Stable := true;
    if ¬G-Stable then F-Stable := Stabilize(G, B, A); G-Stable := true;
}

Initialize() {
  F-Stable := CheckStability(F, A, B); G-Stable := CheckStability(G, B, A);
}

bool Stabilize(H, X, Y) {
  Y_old := Y;
  Y := ⊔{Y' ∈ Abs | Y' ⊴ Y, ⟨X, Y'⟩ is H-complete};
  return (Y = Y_old);
}
```

**Fig. 1.** Basic Shell Algorithm.

completeness of an abstract interpretation does not depend on the choice of the abstract function $f^\sharp$ but only depends on the abstract domains $A$ and $B$. Accordingly, a pair of abstract domains $\langle A, B\rangle \in \mathrm{Abs}(C) \times \mathrm{Abs}(D)$ is called forward complete for $f$ (or simply $f$-complete) iff $f \circ \gamma_{A,C} = \gamma_{B,D} \circ (\alpha_{D,B} \circ f \circ \gamma_{A,C})$. Equivalently, $\langle A, B\rangle$ is $f$-complete iff the image of $f$ in $D$, that is $f(\gamma_{A,C}(A))$, is contained in $\gamma_{B,D}(B)$. If $\mathcal{F} \subseteq C \to D$ is a set of concrete functions then $\langle A, B\rangle$ is $\mathcal{F}$-complete when $\langle A, B\rangle$ is $f$-complete for all $f \in \mathcal{F}$.

***Shells of Abstract Domains.*** Given a set of semantic functions $\mathcal{F} \subseteq C \to D$ and a pair of abstractions $\langle A, B\rangle \in \mathrm{Abs}(C) \times \mathrm{Abs}(D)$, the notion of forward complete shell [4] formalizes the problem of finding the most abstract pair $\langle A', B'\rangle$ such that $A' \trianglelefteq A, B' \trianglelefteq B$ and $\langle A', B'\rangle$ is $\mathcal{F}$-complete, which is a particular case of abstraction refinement. It turns out (see [4]) that any pair $\langle A, B\rangle$ can be minimally refined to its forward $\mathcal{F}$-complete shell $\mathrm{Shell}_{\mathcal{F}}(\langle A, B\rangle) \triangleq \sqcup\{\langle A', B'\rangle \in \mathrm{Abs}(C) \times \mathrm{Abs}(D) \mid \langle A', B'\rangle \trianglelefteq \langle A, B\rangle,\ \langle A', B'\rangle \text{ is } \mathcal{F}\text{-complete}\}$. Thus, $\mathrm{Shell}_{\mathcal{F}}(\langle A, B\rangle)$ encodes the least refinement of a pair of abstractions $\langle A, B\rangle$ which is needed in order to obtain forward completeness for $\mathcal{F}$.

Let us now consider a further set of concrete semantic functions $\mathcal{G} \subseteq D \to C$ that operate in the opposite direction w.r.t. $\mathcal{F}$, i.e., from $D$ to $C$. Given $A \in \mathrm{Abs}(C)$ and $B \in \mathrm{Abs}(D)$, it makes sense to consider both forward $\mathcal{F}$-completeness of $\langle A, B\rangle$ and forward $\mathcal{G}$-completeness of the reversed pair $\langle B, A\rangle$. Thus, $\langle A, B\rangle$ is defined to be $\langle \mathcal{F}, \mathcal{G}\rangle$-complete when $\langle A, B\rangle$ is $\mathcal{F}$-complete and $\langle B, A\rangle$ is $\mathcal{G}$-complete. Here again, any pair $\langle A, B\rangle$ can be minimally refined to its $\langle \mathcal{F}, \mathcal{G}\rangle$-complete shell $\mathrm{Shell}_{\langle \mathcal{F},\mathcal{G}\rangle}(\langle A, B\rangle) \triangleq \sqcup\{\langle A', B'\rangle \in \mathrm{Abs}(C) \times \mathrm{Abs}(D) \mid \langle A', B'\rangle \trianglelefteq \langle A, B\rangle,\ \langle A', B'\rangle \text{ is } \langle \mathcal{F}, \mathcal{G}\rangle\text{-complete}\}$.

The combined shell $\mathrm{Shell}_{\langle \mathcal{F},\mathcal{G}\rangle}(\langle A, B\rangle)$ can be obtained through the ShellAlgo() procedure described in Figure 1. This procedure ShellAlgo() crucially relies on the Stabilize() function: given a set of functions $\mathcal{H}$ and a pair of abstractions $\langle X, Y\rangle$,

we have that Stabilize$(\mathcal{H}, X, Y)$ refines the abstraction $Y$ to $Y_{\mathrm{stable}} \triangleq \sqcup \{Y' \mid Y' \trianglelefteq Y,\ \langle X, Y'\rangle$ is $\mathcal{H}$-complete$\}$, so that $\langle X, Y_{\mathrm{stable}}\rangle$ becomes $\mathcal{H}$-stable (i.e., $\mathcal{H}$-complete). For instance, Stabilize$(\mathcal{F}, A, B)$ minimally refines $B$ to $B'$ so that $\langle A, B'\rangle$ is $\mathcal{F}$-complete. Hence, while the abstraction $B$ is refined, the abstraction $A$ is left unchanged. Note that if $B$ is actually refined into $B' \triangleleft B$, then the $\mathcal{G}$-Stable flag is set to false so that ShellAlgo() proceeds by $\mathcal{G}$-stabilizing $\langle B', A\rangle$, i.e., by calling Stabilize$(\mathcal{G}, B, A)$. Thus, ShellAlgo$(\mathcal{F}, \mathcal{G}, A, B)$ works by iteratively refining the abstractions $A$ and $B$ separately, namely it refines $B$ w.r.t. $\mathcal{F}$ while $A$ is kept fixed and then it refines $A$ w.r.t. $\mathcal{G}$ while $B$ is kept fixed.

**Theorem 3.1.** ShellAlgo$(\mathcal{F}, \mathcal{G}, A, B) = \mathrm{Shell}_{\langle \mathcal{F}, \mathcal{G}\rangle}(\langle A, B\rangle)$.

## 4 Bisimulation as a Shell

Bisimulation is commonly computed by coarsest partition refinement algorithms [1,8] that iteratively refine a current partition until it becomes the bisimulation partition. Coarsest partition refinements can be cast as shells of partitions: given a property of partitions $\mathbb{P} \subseteq \mathrm{Part}(X)$, the $\mathbb{P}$-shell of $Q \in \mathrm{Part}(X)$ corresponds to the coarsest partition refinement of $Q$ that satisfies $\mathbb{P}$, when this exists. In this section we show how bisimulation in PLTSs can be equivalently stated in terms of forward complete shells of partitions w.r.t. suitable concrete semantic functions. We also show how the above basic shell algorithm ShellAlgo() can be instantiated to compute bisimulations on PLTSs.

***Shells of Partitions.*** Let us first recall that, given a finite set $X$, $\langle \mathrm{Part}(X), \preceq, \curlyvee, \curlywedge\rangle$ is a (finite) lattice where $P_1 \preceq P_2$ (i.e., $P_2$ is coarser than $P_1$ or $P_1$ refines $P_2$) iff $\forall x. P_1(x) \subseteq P_2(x)$, and its top element is $\top_{\mathrm{Part}(X)} = \{X\}$. By following the approach in [11], any partition $P \in \mathrm{Part}(X)$ can be viewed as an abstraction of $\wp(X)_{\subseteq}$ where any set $S \subseteq X$ is approximated through its minimal cover in the partition $P$. This is formalized by the abstract domain $\mathrm{closed}(P) \triangleq \{S \subseteq X \mid P(S) = S\}$ so that $S \in \mathrm{closed}(P)$ iff $S = \cup_{i\in I} B_i$ for some blocks $\{B_i\}_{i\in I} \subseteq P$. Note that $\varnothing, X \in \mathrm{closed}(P)$ and that $\langle \mathrm{closed}(P), \subseteq, \cup, \cap\rangle$ is a lattice. It turns out that $\langle \mathrm{closed}(P), \subseteq\rangle$ is an abstraction in $\mathrm{Abs}(\wp(X)_{\subseteq})$, where any set $S \subseteq X$ is approximated through the blocks in $P$ covering $S$, namely by $\cup\{B \in P \mid B \cap S \neq \varnothing\} \in \mathrm{closed}(P)$.

The above embedding of partitions as abstract domains allows us to define a notion of forward completeness for partitions. Let $f : \wp(X) \to \wp(Y)$ be a concrete semantic function. Then, a pair of partitions $\langle P, Q\rangle \in \mathrm{Part}(X) \times \mathrm{Part}(Y)$ is (forward) $f$-complete when for any union $U \in \mathrm{closed}(P)$ of blocks of $P$, $f(U)$ is a union of blocks of $Q$, namely $f(U) \in \mathrm{closed}(Q)$. Also, if we additionally consider $g : \wp(Y) \to \wp(X)$ then $\langle P, Q\rangle$ is $\langle f, g\rangle$-complete when $\langle P, Q\rangle$ is $f$-complete and $\langle Q, P\rangle$ is $g$-complete. As in Section 3, forward complete shells of partitions exist. Given $\mathcal{F} \subseteq \wp(X) \to \wp(Y)$ and $\mathcal{G} \subseteq \wp(Y) \to \wp(X)$, $\mathrm{Shell}_{\langle \mathcal{F}, \mathcal{G}\rangle}(\langle P, Q\rangle)$ is the coarsest pair of partitions that (component-wise) refines the pair $\langle P, Q\rangle$ and is $\langle \mathcal{F}, \mathcal{G}\rangle$-complete, namely $\mathrm{Shell}_{\langle \mathcal{F}, \mathcal{G}\rangle}(\langle P, Q\rangle) \triangleq \curlyvee\{\langle P', Q'\rangle \in \mathrm{Part}(X) \times \mathrm{Part}(Y) \mid \langle P', Q'\rangle \preceq \langle P, Q\rangle, \langle P', Q'\rangle$ is $\langle \mathcal{F}, \mathcal{G}\rangle$-complete$\}$.

***Bisimulation on PLTSs.*** In [11] it is shown that bisimulation on LTSs can be equivalently defined in terms of forward complete shells of partitions w.r.t. the predecessor operator. This same idea scales to the case of PLTSs taking into account that: (i) in a PLTS the target of the transition relation is a set of distributions rather than a set of states, and (ii) bisimulation on the states of a PLTS induces an equivalence over distributions that depends on the probability that the distributions associate to the blocks of bisimilar states. Let $\mathcal{S} = \langle \Sigma, Act, \rightarrow \rangle$ be a PLTS and consider the following two functions, where $a \in Act$ and $p \in [0,1]$:

$$\begin{array}{l} \mathrm{pre}_a \ : \wp(\mathrm{Distr}) \to \wp(\Sigma),\ \mathrm{pre}_a(D) \triangleq \{s \in \Sigma \mid s \xrightarrow{a} D\} \\ \mathrm{prob}_p : \wp(\Sigma) \to \wp(\mathrm{Distr}),\ \mathrm{prob}_p(S) \triangleq \{d \in \mathrm{Distr} \mid d(S) \geq p\} \end{array}$$

$\mathrm{pre}_a$ is the $a$-predecessor function in the PLTS $\mathcal{S}$ while $\mathrm{prob}_p(S)$ returns the distributions whose probability on the set $S$ is higher than $p$. Let $\mathrm{pre} \triangleq \{\mathrm{pre}_a\}_{a \in Act}$ and $\mathrm{prob} \triangleq \{\mathrm{prob}_p\}_{p \in [0,1]}$. It is worth noticing that this pair of sets of functions provides an encoding of the PLTS $\mathcal{S}$: pre encodes the transition relation $\rightarrow$, while any distribution $d$ in $\mathcal{S}$ can be encoded through prob. For instance, the support of a distribution $d \in \mathrm{Distr}$ is given by the minimal set of states $S$ such that $d \in \mathrm{prob}_1(S)$, while, for any $s \in \Sigma$, $d(s) = \sup\{p \in [0,1] \mid d \in \mathrm{prob}_p(\{s\})\}$.

**Lemma 4.1.** *Consider $\langle P, \mathcal{P} \rangle \in \mathrm{Part}(\Sigma) \times \mathrm{Part}(\mathrm{Distr})$. $\langle P, \mathcal{P} \rangle$ is $\langle \mathrm{prob}, \mathrm{pre} \rangle$-complete if and only if the following two conditions hold:* (i) *if $s \xrightarrow{a} d$ and $t \in P(s)$ then $t \xrightarrow{a} \mathcal{P}(d)$;* (ii) *if $e \in \mathcal{P}(d)$ then $d \equiv_P e$.*

Consequently, a partition $P \in \mathrm{Part}(\Sigma)$ is a bisimulation on $\mathcal{S}$ if and only if $\langle P, \equiv_P \rangle$ is $\langle \mathrm{prob}, \mathrm{pre} \rangle$-complete. In turn, the coarsest bisimulation $P_{\mathrm{bis}}$ on $\mathcal{S}$ can be obtained as a forward complete shell of partitions.

**Theorem 4.2.** $\langle P_{\mathrm{bis}}, \equiv_{P_{\mathrm{bis}}} \rangle = \mathrm{Shell}_{\langle \mathrm{prob}, \mathrm{pre} \rangle}(\top_{\mathrm{Part}(\Sigma)}, \top_{\mathrm{Part}(\mathrm{Distr})})$.

***Bisimulation Algorithm.*** By Theorem 4.2, $P_{\mathrm{bis}}$ can be computed as a partition shell by instantiating the basic shell algorithm in Figure 1 to $\mathcal{F} = \{\mathrm{prob}_p\}_{p \in [0,1]}$ and $\mathcal{G} = \{\mathrm{pre}_a\}_{a \in Act}$, and by viewing partitions in $\mathrm{Part}(\Sigma) \times \mathrm{Part}(\mathrm{Distr})$ as abstract domains. This leads to design a bisimulation algorithm called PBis that maintains a pair of state and distribution partitions $\langle P, \mathcal{P} \rangle \in \mathrm{Part}(\Sigma) \times \mathrm{Part}(\mathrm{Distr})$ and whose initialization and stabilization functions are given in Figure 2.

The function call preStabilize($\langle \mathcal{P}, P \rangle$) refines the state partition $P$ into $P'$ so that $\langle \mathcal{P}, P' \rangle$ is pre-complete. Note (cf. Lemma 4.1) that in order to get pre-completeness it is sufficient to minimally refine $P$ so that for any block of distributions $C \in \mathcal{P}$, and for any incoming label $a \in \mathrm{in}(C)$, $\mathrm{pre}_a(C)$ is a union of blocks of $P$. If $\mathrm{pre}_a(C)$ is not a union of blocks of $P$ then $\mathrm{pre}_a(C) \subseteq \Sigma$ is called a splitter of $P$, and we denote by $\mathrm{Split}(P, \mathrm{pre}_a(C))$ the partition obtained from $P$ by replacing each block $B \in P$ with the nonempty sets $B \cap \mathrm{pre}_a(C)$ and $B \smallsetminus \mathrm{pre}_a(C)$. Notice that when some $\mathrm{pre}_a(C)$ is already a union of blocks of $P$ we have that $\mathrm{Split}(P, \mathrm{pre}_a(C)) = P$, i.e., we also allow no splitting. Hence, preStabilize() refines $P$ by iteratively splitting $P$ w.r.t. $\mathrm{pre}_a(C)$, for all $C \in \mathcal{P}$ and $a \in \mathrm{in}(C)$. On the other hand, the function call probStabilize($\langle P, \mathcal{P} \rangle$) refines the current distribution partition $\mathcal{P}$ into $\mathcal{P}'$ so that $\langle P, \mathcal{P}' \rangle$ is prob-complete. It

```
Initialize() {
  forall s ∈ Σ do P(s) := Σ;  forall d ∈ Distr do 𝒫(d) := Distr;
  preStabilize(⟨𝒫, P⟩);  preStable := probStabilize(⟨P, 𝒫⟩);  probStable := true;
}
bool preStabilize(⟨𝒫, P⟩) {
  P_old := P;
  forall C ∈ 𝒫 do  forall a ∈ in(C) do  P := Split(P, pre_a(C));
  return (P ≠ P_old)
}
bool probStabilize(⟨P, 𝒫⟩) {
  𝒫_old := 𝒫;
  forall B ∈ P do  forall d ∈ Distr do  𝒫 := Split(𝒫, {e ∈ Distr | e(B) = d(B)});
  return (𝒫 ≠ 𝒫_old)
}
```

**Fig. 2.** Bisimulation Algorithm PBis.

turns out that $\langle P, \mathcal{P}\rangle$ is prob-complete when for any block $B \in P$ and any distribution $d \in \mathrm{Distr}$, $\{e \in \mathrm{Distr} \mid e(B) = d(B)\}$ is a union of blocks of $\mathcal{P}$. Thus, probStabilize() iteratively splits the distribution partition $\mathcal{P}$ w.r.t. $\{e \in \mathrm{Distr} \mid e(B) = d(B)\}$, for all $B \in P$ and $d \in \mathrm{Distr}$.

**Theorem 4.3.** *For a finite PLTS* $\mathbb{S}$, $\mathrm{PBis}(\mathbb{S})$ *terminates and is correct, i.e., if* $\langle P, \mathcal{P}\rangle$ *is the output of* $\mathrm{PBis}(\mathbb{S})$ *then* $P = P_{\mathrm{bis}}$ *and* $\mathcal{P} = \equiv_{P_{\mathrm{bis}}}$.

***Implementation.*** Baier-Engelen-Majster's two-phased partitioning algorithm [1] is the standard procedure for computing the bisimulation $P_{\mathrm{bis}}$. This bisimulation algorithm can be essentially viewed as an implementation of the above PBis algorithm, since the two phases of Baier et al.'s algorithm (see [1, Figure 9]) coincide with our preStabilize() and probStabilize() functions. The only remarkable difference is that instead of using a single partition over all the distributions in Distr, Baier et al.'s algorithm maintains a so-called step partition, namely, a family of partitions $\{M_a\}_{a \in Act}$ such that, for any $a \in Act$, $M_a$ is a partition of the distributions in $\mathrm{post}_a(\Sigma)$, i.e., the distributions that have an incoming edge labeled with $a$. As a consequence, in the phase that corresponds to probStabilize(), any partition $M_a$ is split w.r.t. all the splitters $\{e \in \mathrm{post}_a(\Sigma) \mid e(B) = d(B)\}$, where $B \in P$ and $d \in \mathrm{post}_a(\Sigma)$. Baier et al.'s algorithm is implemented by exploiting Hopcroft's "process the smaller half" principle when splitting the state partition w.r.t. a splitter $\mathrm{pre}_a(C)$ and this allows to obtain a procedure that computes bisimulation in $O(|{\to}||\Sigma|(\log|{\to}| + \log|\Sigma|))$ time and $O(|{\to}||\Sigma|)$ space.

## 5 Simulation as a Shell

***Shells of Preorders.*** Recall that, given any finite set $X$, $\langle \mathrm{PreOrd}(X), \subseteq, \cup^t, \cap\rangle$ is a lattice, where $R_1 \cup^t R_2$ is the transitive closure of $R_1 \cup R_2$ and the top element

is $\top_{\mathrm{PreOrd}(X)} \triangleq X \times X$. Analogously to partitions, any preorder $R \in \mathrm{PreOrd}(X)$ can be viewed as an abstraction of $\wp(X)_{\subseteq}$, where any set $S \subseteq X$ is approximated by its $R$-closure $R(S)$. Formally, a preorder $R \in \mathrm{PreOrd}(X)$ can be viewed as the abstract domain $\mathrm{closed}(R) \triangleq \{S \subseteq X \mid R(S) = S\}$. Observe that $S \in \mathrm{closed}(R)$ iff $S = \cup_{i \in I} R(x_i)$ for some set $\{x_i\}_{i \in I} \subseteq X$ and that $\langle \mathrm{closed}(R), \subseteq, \cup, \cap \rangle$ is a lattice (note that $\varnothing, X \in \mathrm{closed}(R)$). It turns out that $\mathrm{closed}(R) \in \mathrm{Abs}(\wp(X)_{\subseteq})$: this means that any set $S \subseteq X$ is approximated by its $R$-closure, namely by $R(S) \in \mathrm{closed}(R)$.

Given the functions $\langle \mathcal{F}, \mathcal{G} \rangle \subseteq (\wp(X) \to \wp(Y)) \times (\wp(Y) \to \wp(X))$, a pair of preorders $\langle R, S \rangle \in \mathrm{PreOrd}(X) \times \mathrm{PreOrd}(Y)$ is (forward) $\langle \mathcal{F}, \mathcal{G} \rangle$-complete when for any $f \in \mathcal{F}$ and $g \in \mathcal{G}$, if $\langle U, V \rangle \in \mathrm{closed}(R) \times \mathrm{closed}(S)$ then $\langle f(U), g(V) \rangle \in \mathrm{closed}(S) \times \mathrm{closed}(R)$. Forward complete shells of preorders are therefore defined as follows: $\mathrm{Shell}_{\langle \mathcal{F}, \mathcal{G} \rangle}(\langle R, S \rangle)$ is the largest pair of preorders $\langle R', S' \rangle \subseteq \langle R, S \rangle$ which is $\langle \mathcal{F}, \mathcal{G} \rangle$-complete.

***Simulation on PLTSs.*** Similarly to the case of bisimulation, simulation can be equivalently expressed in terms of forward completeness w.r.t. $\mathrm{prob} = \{\mathrm{prob}_p\}_{p \in [0,1]}$ and $\mathrm{pre} = \{\mathrm{pre}_a\}_{a \in Act}$.

**Lemma 5.1.** *Let $\langle R, \mathcal{R} \rangle \in \mathrm{PreOrd}(\Sigma) \times \mathrm{PreOrd}(\mathrm{Distr})$. Then, $\langle R, \mathcal{R} \rangle$ is $\langle \mathrm{prob}, \mathrm{pre} \rangle$-complete if and only if the following two conditions hold:* (i) *if $t \in R(s)$ and $s \xrightarrow{a} d$ then there exists $e$ such that $t \xrightarrow{a} e$ and $e \in \mathcal{R}(d)$;* (ii) *if $e \in \mathcal{R}(d)$ then $d \leq_R e$.*

Thus, a preorder $R \in \mathrm{PreOrd}(\Sigma)$ is a simulation on $\mathcal{S}$ if and only if $\langle R, \leq_R \rangle$ is $\langle \mathrm{prob}, \mathrm{pre} \rangle$-complete. In turn, the greatest simulation preorder $R_{\mathrm{sim}}$ can be obtained as a preorder shell.

**Theorem 5.2.** $\langle R_{\mathrm{sim}}, \equiv_{R_{\mathrm{sim}}} \rangle = \mathrm{Shell}_{\langle \mathrm{prob}, \mathrm{pre} \rangle}(\top_{\mathrm{PreOrd}(\Sigma)}, \top_{\mathrm{PreOrd}(\mathrm{Distr})})$.

## 6 A New Efficient Probabilistic Simulation Algorithm

A new efficient algorithm for computing simulations in PLTSs, called $\mathrm{PSim}$, is designed in this section by instantiating the basic shell algorithm to $\mathcal{F} = \{\mathrm{prob}_p\}_{p \in [0,1]}$ and $\mathcal{G} = \{\mathrm{pre}_a\}_{a \in Act}$, and by viewing preorders in $\mathrm{PreOrd}(\Sigma) \times \mathrm{PreOrd}(\mathrm{Distr})$ as abstract domains.

The high-level design of $\mathrm{PSim}$ is that of ShellAlgo in Figure 1, the only difference being that the input is a PLTS $\mathcal{S}$. $\mathrm{PSim}$ maintains a pair of state and distribution preorders $\langle R, \mathcal{R} \rangle \in \mathrm{PreOrd}(\Sigma) \times \mathrm{PreOrd}(\mathrm{Distr})$, whose initialization and stabilization functions are given in Figure 3 and 4.

The function preStabilize() makes the pair $\langle \mathcal{R}, R \rangle$ pre-complete by refining the state preorder $R$ until there exists a transition $s \xrightarrow{a} d$ such that and $R(s) \not\subseteq \mathrm{pre}_a(\mathcal{R}(d))$. Such a refinement can be efficiently done by following the incremental approach of Henzinger et al. [6] for nonprobabilistic LTSs. On the other hand, the function probStabilize() makes the pair $\langle \mathcal{R}, R \rangle$ prob-complete by refining the distribution preorder $\mathcal{R}$ by iteratively refining it until there exist $e, d$ such that $e \in \mathcal{R}(d)$ and $d \not\leq_R e$. Here, in order to get an efficient incremental computation, we resort to the approach of Zhang et

```
1  Initialize() {
       // Initialize R and ℛ
2      forall s ∈ Σ do R(s) := {t ∈ Σ | out(s) ⊆ out(t)};
3      forall d ∈ Distr do ℛ(d) := {e ∈ Distr | Init_SMF(d, e, R) = true};
       // Initialize in
4      forall d ∈ Distr do in(d) := {a ∈ Act | pre_a(d) ≠ ∅};
       // Initialize Count
5      forall e ∈ Distr do
6          forall a ∈ in(e) do
7              forall x ∈ pre_a(Distr) do
8                  Count(x, a, e) := |{d ∈ Distr | x -a-> d, d ∈ ℛ(e), a ∈ in(e)}|;
       // Initialize Remove
9      forall d ∈ Distr do
10         forall a ∈ in(d) do
11             Remove_a(d) := {s ∈ Σ | a ∈ out(s), s -a-/-> ℛ(d)};
       // Initialize Stability Flags
12     probStable := true;
13     if ∃e ∈ Distr, a ∈ in(e) such that Remove_a(e) ≠ ∅ then preStable := false;
14     else preStable := true;
       // Initialize Listener
15     forall x, y ∈ Σ do Listener(x, y) := {(d, e) | x ∈ supp(d), e ∈ supp(e)};
       // Initialize Deleted Arcs
16     Deleted := ∅;
17 }
```

**Fig. 3.** Initialization function.

al.'s simulation algorithm [13], and we stabilize the distribution preorder $\mathcal{R}$ by computing sequences of maximum flow problems. More precisely, given a pair of distributions $(d, e)$, successive calls to probStabilize() might repeatedly check whether $d \leq_R e$ where $R$ is the current (new) state preorder. Let us recall [1] that the test $d \leq_R e$ can be implemented by checking whether the maximum flow over a network built out of $(d, e)$ and $R$, here denoted by $\mathcal{N}(d, e, R)$, is 1. Zhang et al. [13] observe that the networks for a given pair $(d, e)$ across successive iterations are very similar, since they differ only by deletion of some edges due to the refinement of $R$. Therefore, in order to incrementally deal with this sequence of tests, Zhang et al.'s algorithm saves after each iteration the current network $\mathcal{N}(d, e, R)$ together with its maximum flow information, and this allows us to use at the next iteration a so-called preflow algorithm which is initialized with the previous maximum flow function. Due to lack of space, we do not discuss the details of the preflow algorithm in [13], that is used here as a black box that efficiently solves the sequence of maximum flow problems that arise for a same network.

PSim is designed around the following data structures. First, the two preorders $R \subseteq \Sigma \times \Sigma$ and $\mathcal{R} \subseteq \text{Distr} \times \text{Distr}$, which are stored as boolean matrices and are initialized so that they are coarser than $R_{\text{sim}}$ and $\leq_{R_{\text{sim}}}$. In particular, the initial pre-

```
1  bool preStabilize(⟨ℛ, R⟩) {
2      Deleted := ∅;
3      while ∃ Remove_a(e) ≠ ∅ do
4          Remove := Remove_a(e); Remove_a(e) := ∅;
5          forall t -a-> e do
6              forall w ∈ Remove do
7                  if w ∈ R(t) then Deleted := Deleted ∪ {(t, w)}; R(t) := R(t) \ {w};
8      if (Deleted ≠ ∅) then probStable := false;
9      return probStable;
10 }

1  bool probStabilize(⟨R, ℛ⟩) {
2      forall (t, w) ∈ Deleted do
3          forall (d, e) ∈ Listener(t, w) such that e ∈ ℛ(d) do
4              if SMF(d, e, (t, w)) = false then
5                  ℛ(d) := ℛ(d) ∖ {e};
6                  forall b ∈ in(e) ∩ in(d) do
7                      forall s -b-> e do
8                          Count(s, b, d)- -;
9                          if Count(s, b, d) = 0 then
10                             Remove_b(d) := Remove_b(d) ∪ {s}; preStable := false;
11     return preStable;
12 }
```

**Fig. 4.** Stabilization functions.

order $R$ is coarser than $R_{\mathrm{sim}}$ since if $s \xrightarrow{a} d$ and $t \not\xrightarrow{a}$ then $t \notin R_{\mathrm{sim}}(s)$. Moreover, line 3 initializes $\mathcal{R}$ so that $\mathcal{R} = \leq_R$: this is done by calling the function Init_SMF$(d, e, R)$ which in turn calls the preflow algorithm to check whether $d \leq_R e$, and in case this is true, stores the network $\mathcal{N}(d, e, R)$ to reuse it in later calls to probStabilize(). The additional data structures used by PSim come from the efficient refinement approaches used in [6] and [13]. Indeed, as in [6], for any distribution $e$ and for any incoming action $a \in \mathrm{in}(e)$, we store and maintain a set $\mathrm{Remove}_a(e) \triangleq \{s \in \Sigma \mid s \xrightarrow{a}, \ s \not\xrightarrow{a} \mathcal{R}(e)\}$ that is used to refine the relation $R$ such that if $t \xrightarrow{a} e$ then $R(t)$ is pruned to $R(t) \setminus \mathrm{Remove}_a(e)$ (lines 5-7 of preStabilize()). The Count table is used to efficiently refill the remove sets (line 10 of probStabilize()), since it allows to test whether $s \not\xrightarrow{a} \mathcal{R}(e)$ in $O(1)$. On the other hand, in order to get an efficient refinement also for the distribution preorder $\mathcal{R}$, as in Zhang et al. [13], for any pair of states $(x, y)$ we compute and store a set $\mathrm{Listener}(x, y) \triangleq \{(d, e) \in \mathrm{Distr} \times \mathrm{Distr} \mid x \in \mathrm{supp}(d), \ y \in \mathrm{supp}(e)\}$ that contains all the pairs of distributions $(d, e)$ such that the network $\mathcal{N}(d, e, R)$ could contain the edge $(x, y)$, i.e., the networks that are affected when the pair $(x, y)$ is removed from $R$. Indeed, these sets are used in probStabilize() to recognize the pairs $(d, e)$ that have been affected by the refinement of $R$ due to the previous call of preStabilize() (lines 2-3 of

probStabilize()).

As a result, at the end of the initialization, the probStable flag is true (due to the initialization of $\mathcal{R}$ as $\leq_R$), whereas the preStable flag is false if there is at least a nonempty remove set. The main loop of $\mathrm{PSim}$ then proceeds by repeatedly calling the stabilization functions until $\langle R, \mathcal{R}\rangle$ becomes $\langle \mathrm{prob}, \mathrm{pre}\rangle$-complete. More precisely, a call to preStabilize(): (i) empties all the $\mathrm{Remove}$ sets, (ii) collects all the pairs removed from $R$ into the set $\mathrm{Deleted}$, and (iii) sets the probStable flag to false when $R$ has changed. On the other hand, a call to probStabilize() relies on the sets $\mathrm{Deleted}$ and $\mathrm{Listener}$ to identify all the networks $\mathcal{N}(d, e, R)$ that have been affected by the refinement of $R$ due to preStabilize(). For any pair $(t, w)$ that has been removed from $R$, the call SMF$(d, e, (t, w))$ at line 4 removes the edge $(t, w)$ from the network for $(d, e)$ and checks whether it still has a maximum flow equals to 1. Hence, if this is not the case, $e$ is removed from $\mathcal{R}(d)$. Notice that such a pruning may induce an update of some $\mathrm{Remove}_b(d)$ set, which in turn triggers a further call of preStabilize() by setting the preStable flag to false.

The correctness of $\mathrm{PSim}$ is a direct consequence of Theorems 3.1 and 5.2 and the fact that the procedures in Figure 4 correctly stabilize the preorders $R$ and $\mathcal{R}$. The complexity bounds of $\mathrm{PSim}$ are given in terms of the following sizes. Let $\mathcal{S} = \langle \Sigma, Act, \rightarrow\rangle$ be the input PLTS. Then, $|\mathrm{Distr}| = |\bigcup_{a\in Act} \mathrm{post}_a(\Sigma)|$ is the number of distributions appearing as target of some transition. Also, the number of edges in $\mathcal{S}$ is $|\rightarrow| \leq |\Sigma||\mathrm{Distr}||Act|$. Moreover, we consider the following sizes: $p \triangleq \sum_{d\in\mathrm{Distr}} |\mathrm{supp}(d)|$ and $m \triangleq \sum_{a\in Act}\sum_{s\in\Sigma}\sum_{d\in\mathrm{post}_a(s)}(|\mathrm{supp}(d)| + 1)$. Thus, $p$ represents the full size of $\mathrm{post}(\Sigma)$, being the number of states that appear in the support of some distribution in $\mathrm{post}(\Sigma)$, while $m$ represents the number of transitions from states to states in $\mathcal{S}$, where a "state transition" $(s, t)$ is taken into account when $s \xrightarrow{a} d$ and $t \in \mathrm{supp}(d)$. Clearly, $|\mathrm{Distr}| \leq p \leq |\mathrm{Distr}||\Sigma|$ and $|\Sigma| \leq |\rightarrow| \leq m \leq |\mathrm{Distr}||\Sigma|$. The key point to remark is that $p \leq m$, since the "states" of $\mathcal{S}$ are always less than the "state transitions" in $\mathcal{S}$.

**Theorem 6.1 (Correctness and Complexity).** *Let $\mathcal{S}$ be a finite PLTS. $\mathrm{PSim}(\mathcal{S})$ terminates with output $\langle R_{\mathrm{sim}}, \leq_{R_{\mathrm{sim}}}\rangle$ and runs in $O(|\Sigma|(p^2 + |\rightarrow|))$-time and $O(p^2 + (|\Sigma| + |\mathrm{Distr}|)|\rightarrow|)$-space.*

It is easy to observe that $(|\Sigma| + |\mathrm{Distr}|)|\rightarrow| \leq m^2$, so that $\mathrm{PSim}$ results to be more efficient than the most efficient probabilistic simulation algorithm in literature, that is Zhang et al.'s algorithm [13], that runs in $O(|\Sigma|m^2)$-time and $O(m^2)$-space. Our scaling down from the factor $m$ to $p$, that is from the size of the "state transitions" to the size of the "state" space, basically depends on the fact that in Zhang et al.'s algorithm the same test $d \not\leq_R e$ is repeated for every pair of states $(s_i, t_i)$ such that $s_i \in \mathrm{pre}_a(d), t_i \in \mathrm{pre}_a(e)$, whereas in $\mathrm{PSim}$ once the test $d \not\leq_R e$ has been performed, every state $t_i$ is removed from $R(s_i)$. Such a difference becomes evident when the input PLTS $\mathcal{S}$ degenerates to a LTS. In this case a call to the function SMF() can be executed in $O(1)$, so that the time complexity of [13] boils down to $O(|\rightarrow|^2)$, whereas in this case $\mathrm{PSim}$ runs in $O(|\Sigma||\rightarrow|)$-time, essentially reducing to Henzinger et al. [6]'s nonprobabilistic simulation algorithm. As a further difference, it is worth observing that Zhang et al.'s algorithm relies on a positive logic that at each iteration $i$ computes the pairs $(s_i, t_i)$ such that $t_i \in R_i(s_i)$, whereas $\mathrm{PSim}$ follows a dual, negative, strategy that removes from $R_i$ the pairs $(s_i, t_i)$ such that $t_i \notin R_i(s_i)$.

## 7 Future Work

We have shown how abstract interpretation can be applied in the context of behavioral relations between probabilistic processes. We focused here on bisimulation/simulation relations on PLTSs and we showed how efficient algorithms that compute these behavioral relations can be derived. As future work, we plan to investigate how this abstract interpretation approach can be adapted to characterize the weak variants of bisimulation/simulation and the so-called probabilistic bisimulations/simulations on PLTSs [12]. We also intend to apply a coarsest partition refinement approach to design a "symbolic" version of our $\mathrm{PSim}$ simulation algorithm. Analogously to the symbolic algorithm by Ranzato and Tapparo [10] for nonprobabilistic simulation, the basic idea is to symbolically represent the relations $R$ on states and $\mathcal{R}$ on distributions through partitions (of states and distributions) and corresponding relations between their blocks. It is worth noting that this partition refinement approach has been already applied by Zhang [14] to design a space-efficient simulation algorithm for PLTSs. Finally, we envisage to study how the abstract interpretation approach can be related to the logical characterizations of behavioral relations of probabilistic processes studied e.g. in [9].

## References


1. C. Baier, B. Engelen and M. Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *J. Comp. Syst. Sci.*, 60:187-231, 2000.
2. 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.
3. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. 6th ACM POPL*, pp. 269–282, 1979.
4. R. Giacobazzi and E. Quintarelli. Incompleteness, counterexamples and refinements in abstract model checking. In *Proc. 8th SAS*, LNCS 2126, pp. 356-373, 2001.
5. R.J. van Glabbeek, S. Smolka, B. Steffen and C. Tofts. Reactive, generative and stratified models for probabilistic processes. In *Proc. IEEE LICS'90*, pp. 130-141, 1990.
6. M.R. Henzinger, T.A. Henzinger and P.W. Kopke. Computing simulations on finite and infinite graphs. In *Proc. 36th FOCS*, pp. 453–462, 1995.
7. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1-28, 1991.
8. R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973-989, 1987
9. A. Parma and R. Segala. Logical characterizations of bisimulations for discrete probabilistic systems. In *Proc. FOSSACS'07*, LNCS 4423, p. 287-301, 2007.
10. F. Ranzato and F. Tapparo. A new efficient simulation equivalence algorithm. In *Proc. IEEE LICS'07*, pp. 171-180, 2007.
11. F. Ranzato and F. Tapparo. Generalized strong preservation by abstract interpretation. *J. Logic and Computation*, 17(1):157-197, 2007.
12. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic J. of Computing*, 2(2):250-273, 1995.
13. L. Zhang, H. Hermanns, F. Eisenbrand and D.N. Jansen. Flow faster: efficient decision algorithms for probabilistic simulations. *Logical Methods in Computer Science*, 4(4), 2008.
14. L. Zhang. A space-efficient probabilistic simulation algorithm. In *Proc. CONCUR'08*, LNCS 5201, pp. 248-263, 2008.