Weak Bisimulation for Fully Probabilistic Processes

Christel Baier[1] and Holger Hermanns[2]

[1] Fakultät für Mathematik & Informatik, Universität Mannheim, Germany
e-mail: baier@pi1.informatik.uni-mannheim.de
[2] Informatik VII, Universität Erlangen, Germany
e-mail: hrherman@informatik.uni-erlangen.de

Abstract. Bisimulations that abstract from internal computation have proven to be useful for verification of compositionally defined transition system. In the literature of probabilistic extensions of such transition systems, similar bisimulations are rare. In this paper, we introduce weak bisimulation and branching bisimulation for transition systems where nondeterministic branching is replaced by probabilistic branching. In contrast to the nondeterministic case, both relations coincide. We give an algorithm to decide weak bisimulation with a time complexity cubic in the number of states of the transition system. This meets the worst case complexity for deciding branching bisimulation in the nondeterministic case.

1 Introduction

In recent years, the need to formally reason about probabilistic phenomena in software and hardware systems has incented the study of probabilistic models of computation. A variety of models has been proposed in the literature, most of them based on transition systems. These models can be classified with respect to their treatment of nondeterminsm. Several approaches replace the concept of nondeterministic branching by probabilistic branching, e.g. [9, 19, 26, 13, 36], whereas others allow for both, nondeterministic as well as probabilistic branching, e.g. [34, 31, 17, 23, 33]. Following [13], the former model can be subdivided according to the relationship between occurences of actions and transition probabilities. In "reactive" systems, transition probability distributions are dependent on the occurrences of actions. In contrast, in "generative" (also called "fully probabilistic") systems (which can be viewed as discrete Markov chains labelled with actions), these distributions implicitly assign probabilities also to occurrences of actions. "Stratified" systems allow for levelwise probabilistic branching.

Verification techniques for such models have been inspired by succesful experiences in the nonprobabilistic case. This includes probabilistic variants of temporal logics, e.g. [2, 5, 11, 17, 19, 20, 31, 32, 34, 35]. Another research strand focusses on equivalences and preorders used to established that one system "implements" another, according to some notion of implementation, such as strong bisimulation [26], simulation [22, 33], testing preorders [7, 8, 9, 23, 37, 36], trace, failure and ready equivalence [24]. For mechanised verification purposes, the complexity of deciding such equivalences for finite state systems is a crucial aspect. In the nonprobabilistic case, for instance, (strong) bisimulation can be decided in time $\mathcal{O}(m \cdot \log n)$ [30] where n is the number of states and m the number of transitions in the underlying transition system. Most of the coarser equivalences are PSPACE-complete [25]. In the probabilistic framework, the situation is slightly different. Most of the equivalences for probabilistic processes (e.g. strong bisimulation or trace, failure, ready and testing equivalence) can be decided in time polynomial in the size of the probabilistic transition system [8, 21, 3].

Several authors mentioned that the definition of a weak bisimulation that abstracts from internal computation is desirable, but problematic in a probabilistic setting [24, 17]. In the nonprobabilistic case, weak bisimulation [29] is fundamental for compositional verification methods that exploit abstraction from internal computation (see [6] for an impressive example). The time complexity for deciding weak bisimulation is $\mathcal{O}(n^{2.3})$, using the transitive closure operation from [10]. Branching bisimulation [14] is a slightly finer relation for the same purpose, it has time complexity $\mathcal{O}(n \cdot m)$ (but a better space complexity than weak bisimulation) [15]. To the best of our knowledge, [33] is the only paper that introduces notions of weak and branching bisimulation for probabilistic transition systems. Their model can be seen as a generalization of reactive transition systems, since transition probability distributions are dependent on occurences of actions, but nondeterministic choices between different distributions are possible for the same action. The definition of weak and branching bisimulation à la [33] replaces Milner's "double arrow relation" (the transitive, reflexive closure of internal transitions) by assigning a (possibly infinite) set of distributions to each state. For a given state, this set represents the (nondeterministic) alternatives of probability distributions on those states that are reachable by sequences of internal transitions. In contrast to the nonprobabilistic case, the transitions involved form a tree rather than a linear chain. It seems to be hard to adapt this notion to other types of probabilistic transition systems, such as fully probabilistic systems.

In this paper, we propose notions of weak bisimulation and branching bisimulation for fully probabilistic transition systems that appear to be rather natural extensions of the corresponding relations in the nonprobabilistic case. We replace Milner's "double arrow relation" by the probabilities to reach states via sequences of internal transitions. In contrast to the nonprobabilistic case where branching bisimulation is strictly finer than weak bisimulation, these two relations coincide in the fully probabilistic case. We present an algorithm to compute the weak bisimulation equivalence classes in time $\mathcal{O}(n^3)$ where n is the number of states in the underlying probabilistic transition system. It is worth noting that this is the same worst case complexity as computing the branching bisimulation equivalence classes of a nonprobabilistic transition system [15].

The paper is organized as follows. In Section 2 we introduce basic notations and properties of fully probabilistic transition systems. Section 3 introduces weak and branching bisimulation and shows that both coincide. Section 4 is devoted to an algorithm to compute weak bisimulation equivalence classes. Section 5 indicates directions for further work. Due to space constraints we only provide sketches of proofs. The complete proofs are contained in [4].

2 Fully probabilistic transition systems

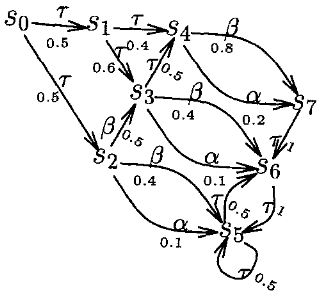
In this section we introduce fully probabilistic transition systems together with some definitions and notations that will be useful in the sequel.

A *fully probabilistic transition system* is a tuple (S, Act, P) where S is a finite set of states, Act a set of actions that contains the internal action τ (which represents any invisible computation) and $P : S \times Act \times S \to [0,1]$ a function such that $\sum_{(a,t)\in Act\times S} P(s,a,t) = 1$ for all $s \in S$. In what follows, we use arabic letters $a, b, \ldots$ to denote (internal or non-internal) actions, greek letters $\alpha, \beta, \ldots$ to denote non-internal actions. For $C \subseteq S$, we define $P(s,a,C) = \sum_{t\in C} P(s,a,t)$. An *execution fragment* is a finite "sequence" $\sigma = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots \xrightarrow{a_k} s_k$ such that $s_0, s_1, \ldots, s_k \in S$, $a_1, \ldots, a_k \in Act$ and $P(s_{i-1}, a_i, s_i) > 0$, $i = 1, \ldots, k$. We define $last(\sigma) = s_k$, $first(\sigma) = s_0$, $length(\sigma) = k$, $trace(\sigma) = a_1a_2\ldots a_k$ and

$$Prob(\sigma) = P(s_0,a_1,s_1)\cdot P(s_1,a_2,s_2)\cdot\ldots\cdot P(s_{k-1},a_k,s_k).$$

An *execution* in (S, Act, P) is an infinite "sequence" $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots$ where $s_0, s_1, \ldots, \in S$, $a_1, a_2, \ldots \in Act$ and $P(s_{i-1}, a_i, s_i) > 0$, $i = 1, 2, \ldots$. We define $first(\pi) = s_0$, and $\pi(k) = s_k$. $\pi^{(k)} = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_k} s_k$ is called the k-th prefix of π. For σ to be an execution fragment with $length(\sigma) = k$, let $\sigma\uparrow$ be the set of executions π with $\pi^{(k)} = \sigma$.

Example 1. A fully probabilistic transition system with 8 states and $Act = \{\alpha, \beta, \tau\}$. If $P(s,a,t)$ is different from zero, its value is annotated to an a-transition joining s and t. To illustrate the above definitions, we calculate $P(s_2, \beta, \{s_3, s_4, s_5\}) = 0.9$. Concerning the execution $\pi = s_0 \xrightarrow{\tau} s_1 \xrightarrow{\tau} s_3 \xrightarrow{\alpha} s_6 \xrightarrow{\tau} s_5 \xrightarrow{\tau} s_5 \xrightarrow{\tau} \ldots$, we have $Prob(\pi^{(3)}) = 0.5 \cdot 0.6 \cdot 0.1 = 0.003$ and $trace(\pi^{(3)}) = \tau\tau\alpha$.



We suppose the reader to be familiar with basic notions of probability theory (see e. g. [16]). For fixed $s \in S$, we define a probability space on the executions starting in s: Let $Exec(s)$ be the set of executions starting in s (i.e. the set of executions π with $first(\pi) = s$), $ExecFrag(s)$ the set of execution fragments σ with $first(\sigma) = s$. Let $\Sigma(s)$ be the smallest sigma field on $Exec(s)$ which contains the basic cylinders $\sigma\uparrow$, $\sigma \in ExecFrag(s)$, and let $\mathcal{P}$ be the unique probability measure on $\Sigma(s)$ with $\mathcal{P}(\sigma\uparrow) = Prob(\sigma)$. For $\Lambda \subseteq Act^*$, $C \subseteq S$, we define $Exec(\Lambda, C)$ to be the set of executions π that lead from $first(\pi)$ to a state in C via a sequence of actions belonging to Λ. Formally, if $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots$ is an execution then $\pi \in Exec(\Lambda, C)$ iff there is some $k \geq 0$ with $trace(\pi^{(k)}) \in \Lambda$ and $s_k \in C$. Let $Exec(s, \Lambda, C) = Exec(\Lambda, C) \cap Exec(s)$. Clearly, $Exec(s, \Lambda, C)$ is measurable in $\Sigma(s)$ as $Exec(s, \Lambda, C) = \bigcup_\sigma \sigma \uparrow$ where σ ranges over all execution fragments starting in s such that $trace(\sigma) \in \Lambda$ and $last(\sigma) \in C$. The probabilities $\mathcal{P}(s, \Lambda, C) = \mathcal{P}(Exec(s, \Lambda, C))$ solve the equation system:

$$\mathcal{P}(s, \Lambda, C) = 1 \quad \text{if } s \in C \text{ and } \varepsilon \in \Lambda$$

$$\mathcal{P}(s, \Lambda, C) = \sum_{(a,t) \in Act \times S} P(s, a, t) \cdot \mathcal{P}(t, \Lambda/a, C) \quad \text{otherwise}$$

where $\Lambda/a = \{\lambda : a\lambda \in \Lambda\}$. Here, ε denotes the empty word in Act^*. If $t \in S$ then we write $\mathcal{P}(s, \Lambda, t)$ rather than $\mathcal{P}(s, \Lambda, \{t\})$. In what follows, we identify a regular expression (e.g. τ^*, $\tau^*\alpha$ or $\tau^*\alpha\tau^*$) with the corresponding set of traces. For instance, $\mathcal{P}(s, \tau^*, C)$ denotes the probability to reach C from s via internal actions.

Example 2. For the fully probabilistic transition system of Example 1, we calculate $\mathcal{P}(s_1, \tau^*\beta\tau^*, \{s_5, s_6, s_7\}) = 0.4 \cdot 0.8 \cdot 1 + 0.6 \cdot (0.4 \cdot 1 + 0.5 \cdot 0.8 \cdot 1) = 0.8$.

3 Weak and branching bisimulation

In this section we define weak and branching bisimulation for fully probabilistic transition systems. While in the nonprobabilistic case branching bisimulation is strictly finer than weak bisimulation, these two relations coincide in the fully probabilistic case.

For the definition of weak bisimulation, we replace Milner's "double arrow" relation $\Rightarrow$ (the transitive, reflexive closure of $\xrightarrow{\tau}$) by the function $\mathcal{P}(s, \tau^*, t)$, which assigns to each pair (s, t) of states the probability to reach state t from s via internal actions. Similarly, for $\alpha \in Act \setminus \{\tau\}$, we deal with the probabilities $\mathcal{P}(s, \tau^*\alpha\tau^*, t)$ rather than Milners weak transition relations $\Rightarrow\xrightarrow{\alpha}\Rightarrow$. In what follows, we fix a fully probabilistic transition system (S, Act, P).

Definition 1. A *weak bisimulation* on (S, Act, P) is an equivalence relation R on S such that for all $(s, s') \in R$, $\lambda \in (Act \setminus \{\tau\}) \cup \{\varepsilon\}$ and all equivalence classes $C \in S/R$:

$$\mathcal{P}(s, \tau^*\lambda\tau^*, C) = \mathcal{P}(s', \tau^*\lambda\tau^*, C).$$

(Note that ε denotes the empty trace and that $\tau^*\varepsilon\tau^* = \tau^*$.) Two states s, s' are called *weakly bisimulation equivalent* (denoted by $s \approx s'$) iff $(s, s') \in R$ for some weak bisimulation R.

Example 3. For the system of Example 1, the smallest equivalence relation R which identifies the states s_5, s_6, s_7 and s_1, s_3, s_4 is a weak bisimulation. To illustrate this, we compute, for instance, $\mathcal{P}(s_4, \tau^*\alpha\tau^*, C_{567}) = 0.2$, as well as $\mathcal{P}(s_3, \tau^*\alpha\tau^*, C_{567}) = 0.5 \cdot \mathcal{P}(s_4, \tau^*\alpha\tau^*, C_{567}) + 0.1 = 0.2$ and

$\mathcal{P}(s_1, \tau^*\alpha\tau^*, C_{567}) = 0.4 \cdot \mathcal{P}(s_4, \tau^*\alpha\tau^*, C_{567}) + 0.6 \cdot \mathcal{P}(s_3, \tau^*\alpha\tau^*, C_{567}) = 0.2$ where $C_{567} = \{s_5, s_6, s_7\}$. As a whole, we obtain the following values of $\mathcal{P}$ (where $C_{134} = \{s_1, s_3, s_4\}$) indicating that the states s_5, s_6, s_7 and s_1, s_3, s_4 are weakly bisimulation equivalent.

$\mathcal{P}$	$\{s_0\}$	$\{s_2\}$	C_{134}	C_{567}	$\{s_0\}$	$\{s_2\}$	C_{134}	C_{567}	$\{s_0\}$	$\{s_2\}$	C_{134}	C_{567}
	τ^*				$\tau^*\alpha\tau^*$				$\tau^*\beta\tau^*$			
s_5 (also s_6, s_7)	0	0	0	1	0	0	0	0	0	0	0	0
s_1 (also s_3, s_4)	0	0	1	0	0	0	0	0.2	0	0	0	0.8
s_2	0	1	0	0	0	0	0	0.1	0	0	0.5	0.4
s_0	1	0.5	0.5	0	0	0	0	0.15	0	0	0.25	0.6

It can be shown that $\approx$ is a weak bisimulation. In the nonprobabilistic case, it holds for weakly bisimulation equivalent states s, s' that if $s \overset{\alpha_1\ldots\alpha_k}{\Longrightarrow} t$ then $s' \overset{\alpha_1\ldots\alpha_k}{\Longrightarrow} t'$ such that t and t' are weakly bisimulation equivalent. Here, $\overset{\alpha_1\ldots\alpha_k}{\Longrightarrow}$ denotes $\Rightarrow\xrightarrow{\alpha_1}\Rightarrow \ldots \Rightarrow\xrightarrow{\alpha_k}\Rightarrow$. This result carries over to the probabilistic case.

Theorem 2. *Let A be a regular expression of the form $\tau^*\alpha_1\tau^*\alpha_2\tau^* \ldots \tau^*\alpha_k$ or $\tau^*\alpha_1\tau^*\alpha_2\tau^* \ldots \tau^*\alpha_k\tau^*$. Then:*

$$\textit{If } s \approx s' \textit{ then } \mathcal{P}(s, A, C) = \mathcal{P}(s', A, C) \textit{ for all } C \in S/\approx.$$

Proof. by induction on k. The basis of induction ($k = 1$) follows by the fact that, for each state s, the vector $(\mathcal{P}(s, \tau^*\alpha, C))_{C \in S/\approx}$ is the unique solution of the linear equation system $\mathbf{x} \cdot \mathbf{A} = \mathbf{a}$ where $\mathbf{A} = (\mathcal{P}(C, \tau^*, C'))_{C,C' \in S/\approx}$ and $\mathbf{a} = (\mathcal{P}([s], \tau^*\alpha\tau^*, C')_{C' \in S/\approx}$. Here, $[s]$ denotes the weak bisimulation equivalence class of s and $\mathcal{P}(C, \tau^*\lambda\tau^*, C') = \mathcal{P}(t, \tau^*\lambda\tau^*, C')$ for some (all) $t \in C$. The induction step follows by the induction hypothesis, the basis of induction and the fact that

$$\mathcal{P}(s, \tau^*\alpha_1\tau^* \ldots \tau^*\alpha_k, C) = \sum_{A \in S/\approx} \mathcal{P}(s, \tau^*\alpha_1, A) \cdot \mathcal{P}(A, \tau^*\alpha_2\tau^* \ldots \tau^*\alpha_k, C)$$

and $\mathcal{P}(s, \tau^*\alpha_1\tau^* \ldots \tau^*\alpha_k\tau^*, C) = \sum_{A \in S/\approx} \mathcal{P}(s, \tau^*\alpha_1\tau^*\alpha_2\tau^* \ldots \tau^*\alpha_k, A) \cdot \mathcal{P}(A, \tau^*, C)$.

□

Van Glabbeek & Weijland [14] introduces branching bisimulation which is strictly finer than weak bisimulation. The basic idea of branching bisimulation is that in order to simulate a step $s \xrightarrow{\alpha} t$ by an equivalent state s', s' is allowed to perform arbitrary many internal actions leading to a state which is still equivalent to s (i.e. the intermediate states before s' also fall in the equivalence class of s and s') and then to perform α reaching a state t' which is equivalent to t. In the probabilistic case, we require that for equivalent states s, s', the probabilities for s and s' to perform internal actions inside the equivalence class of s and s' and then to perform a visible action α leading to state of a certain equivalence class C are the same.

Definition 3. A *branching bisimulation* on (S, Act, P) is an equivalence relation R on S such that

$$\mathcal{P}_R(s, \tau^*\lambda, C) = \mathcal{P}_R(s', \tau^*\lambda, C)$$

for all $(s, s') \in R$, $C \in S/R$ and $\lambda \in (Act \setminus \{\tau\}) \cup \{\varepsilon\}$. Here, $\mathcal{P}_R(s, \tau^*\lambda, C) = \mathcal{P}(Exec_R(s, \tau^*\lambda, C))$ and $Exec_R(s, \tau^*\lambda, C)$ is the set of executions $\pi \in Exec(s)$ such that there is some $k \geq 0$ with $(s, \pi(i)) \in R$, $i = 1, \ldots, k-1$, $trace(\pi^{(k)}) \in \tau^*\lambda$ and $\pi(k) \in C$.

Two states s, s' are called *branching bisimulation equivalent* (denoted $s \approx_{br} s'$) iff $(s, s') \in R$ for some branching bisimulation R.

It can be shown that $\approx_{br}$ is a branching bisimulation. In contrast to the nonprobabilistic case, branching and weak bisimulation coincide:

Theorem 4. *$s \approx s'$ iff $s \approx_{br} s'$.*

Proof. It is easy to see that $\approx_{br}$ is a weak bisimulation. Hence, $\approx_{br} \subseteq \approx$. For the converse, we show that $\approx$ is a branching bisimulation where we use the characterization of branching bisimulations that we give in the next section (Lemma 5). Condition (2) is an easy verification. For condition (1), one first shows that, for all $C \in S/\approx$ and $s \in S \setminus C$,

$$\mathcal{P}(s, \tau^*, C) = \sum_{A \in S/\approx} P'(s, \tau, A) \cdot \mathcal{P}(A, \tau^*, C)$$

where $\mathcal{P}(A, \tau^*\lambda\tau^*, C) = \mathcal{P}(t, \tau^*\lambda\tau^*, C)$ for some (all) $t \in A$, $P'(s, a, A) = P(s, a, A)/(1 - P(s, \tau, [s]))$ if $s \notin A$ or $a \neq \tau$ and $P'(s, \tau, [s]) = 0$ (Again, $[s]$ denotes the weak bisimulation equivalence class of s.). Thus, the vector $(P'(s, \tau, A))_{A \in S/\approx}$ is a solution of the linear equation system $x_{[s]} = 0$, $\sum_{A \in S/\approx} x_A \cdot \mathcal{P}(A, \tau^*, C) = \mathcal{P}([s], \tau, C)$. The matrix $(\mathcal{P}(A, \tau^*, C))_{A, C \in S/\approx}$ can be shown to be regular. Hence, the above equation system has a unique solution. This yields $P'(s, \tau, A) = P'(s', \tau, A)$ for all s, $s' \in S$ with $s \approx s'$. For all $\alpha \in Act$, $s \in S$ and $C \in S/\approx$ we have:

$$\mathcal{P}(s, \tau^*\alpha\tau^*, C) = \sum_{A \in S/\approx} P'([s], \tau, A) \cdot \mathcal{P}(A, \tau^*\alpha\tau^*, C) + P'(s, \alpha, C)$$

where $P'([s], \tau, A) = P'(s, \tau, A)$. This yields $P'(s, \alpha, C) = P'(s', \alpha, C)$ for all s, $s' \in S$ with $s \approx s'$.

□

4 Computing weak bisimulation equivalence classes

In this section we develop an algorithm to compute weak (and branching) bisimulation equivalence classes. The general idea is to use a partitioning/splitter-technique similar to the ones proposed by Kanellakis & Smolka [25] resp. Paige & Tarjan [30] for deciding strong bisimulation in the nonprobabilistic case. The algorithm starts with the trivial partition $X = \{S\}$ and then successively refines the given partition X (with the help of a "splitter" of X), eventually resulting in the set of weak bisimulation equivalence classes.

A *partition* of S is a set X containing pairwise disjoint subsets of S such that each element $s \in S$ is contained in some $C \in X$. Let $[s]_X$ refer to the (unique) element of X with $s \in [s]_X$. For a partition X, let $T_X = \{s \in S : P(s, \tau, [s]_X) < 1\}$. T_X contains all states that with nonzero probability can perform something visible or silently step into a different class. If $s \in T_X$ then we define

$$P_X(s, a, C) = \frac{P(s, a, C)}{1 - P(s, \tau, [s]_X)}.$$

A partition X of S is called a branching bisimulation iff the induced equivalence relation $R_X := \bigcup_{C \in X} C \times C$ is a branching bisimulation. A possible candidate for a "splitter" of a partition X is a pair (α, C) (or a pair (τ, C)) that violates the condition for X to be a branching bisimulation, i.e. $\mathcal{P}_{R_X}(s, \tau^*\alpha, C) \neq \mathcal{P}_{R_X}(s', \tau^*\alpha, C)$ ($\mathcal{P}_{R_X}(s, \tau^*, C) \neq \mathcal{P}_{R_X}(s', \tau^*, C)$, respectively) for some $B \in X$ and $s, s' \in B$. The following characterization of branching bisimulations yields a simpler condition for splitters as it does not require the computation of the probabilities $\mathcal{P}_{R_X}$.

Lemma 5. *A partition X is a branching bisimulation iff the following conditions (1) are (2) are satisfied:*

(1) For all $A \in X$, $s, s' \in A \cap T_X$: $P_X(s, \tau, C) = P_X(s', \tau, C)$ for all $C \in X \setminus \{A\}$, and $P_X(s, \alpha, C) = P_X(s', \alpha, C)$ for all $C \in X$, $\alpha \in Act \setminus \{\tau\}$.

(2) For all $A \in X$ either $A \cap T_X = \emptyset$ or for each $s_0 \in A \setminus T_X$ there is an execution fragment $s_0 \xrightarrow{\tau} \ldots \xrightarrow{\tau} s_k$ with $s_0, \ldots, s_{k-1} \in A \setminus T_X$, $s_k \in A \cap T_X$.

Moreover, if X is a branching bisimulation then $\mathcal{P}_{R_X}(s, \tau^*\lambda, C) = P_X(A, \lambda, C)$ for all $A, C \in X$, $s \in A$. Here, $P_X(A, \lambda, C)$ denotes $P_X(t, \lambda, C)$ for arbitrary $t \in A \cap T_X$ unless $A \cap T_X = \emptyset$. If $A \cap T_X = \emptyset$ then $P_X(A, \tau, A) = 1$ and $P_X(A, a, C) = 0$ if $a \neq \tau$ or $A \neq C$.

Definition 6. A *splitter* of a partition X is a tuple (a, C) consisting of an action $a \in Act$ and some $C \in X$ such that there exists some $B \in X$ (with $B \neq C$ if $a = \tau$) and $P_X(s, a, C) \neq P_X(s', a, C)$ for some states s, $s' \in B \cap T_X$.

The main idea for refining a given partition X via a splitter (a, C) is to isolate in each $B \in X$ (with $B \neq C$ if $a = \tau$) those states s, $s' \in B \cap T_X$ where $P_X(s, a, C) = P_X(s', a, C)$. By condition (2), each such equivalence class A of $B \cap T_X$ has to be enriched with exactly those states $s \in B \setminus T_X$ that can reach A via internal actions and that cannot reach any other equivalence class A' of $B \cap T_X$ without passing A.

Definition 7. For (a, C) to be a splitter of a partition X and $B \in X$ (with $B \neq C$ if $a = \tau$), we define $Split(B, a, C) = (B \cap T_X)/\equiv$ where $s \equiv s'$ iff $P_X(s, a, C) = P_X(s', a, C)$. If $A \in Split(B, a, C)$ then we define the closure $\overline{A}$

of A in X with respect to (a,C) to be the largest set $V \subseteq B$ which contains A and such that for all $s \in V \setminus A$: $P(s,\tau,V) = 1$ and there exists an execution fragment $s = s_0 \xrightarrow{\tau} \dots \xrightarrow{\tau} s_k$ with $s_0,\dots,s_{k-1} \in V$ and $s_k \in A$. We define $Refine(B,\tau,B) = \{B\}$ and, if $a \neq \tau$ or $B \neq C$,

$$Refine(B,a,C) = \{\overline{A} : A \in Split(B,a,C)\} \cup Res(B,a,C),$$
$$Refine(X,a,C) = \bigcup_{B\in X} Refine(B,a,C),$$

where $Res(B,a,C) = \{B \setminus \bigcup_{A\in Split(B,a,C)} \overline{A}\} \setminus \{\emptyset\}$.

It is easy to see that for each partition X which is coarser than $S/\approx_{br}$ and each splitter (a,C) of X, the partition $Refine(X,a,C)$ is coarser than $S/\approx_{br}$ and strictly finer than X. If there is no splitter for X and X is coarser than $S/\approx_{br}$ then $X = S/\approx_{br} = S/\approx$.

Algorithm for computing the weak bisimulation equivalence classes

```
Input:   fully probabilistic transition system $(S, Act, P)$
Output:  $S/\approx$
Method:  $X$ := $\{S\}$;
         While $X$ contains a splitter $(a,C)$ do $X := Refine(X,a,C)$;
         Return $X$.
```

Example 4. Partitioning the transition system from Example 1 proceeds as follows. For the initial partition $\{S\}$, we consider the set $T_{\{S\}} = \{s_2,s_3,s_4\}$. (α,S) and (β,S) are splitters, since, for example, $P_{\{S\}}(s_2,\alpha,S) = 0.1 \neq 0.2 = P_{\{S\}}(s_3,\alpha,S)$. $Split(S,\alpha,S)$ refines $S \cap T_{\{S\}}$ into $\{s_2\}$ and $\{s_3,s_4\}$. The closure in $\{S\}$ yields $\overline{\{s_2\}} = \{s_2\}$ and $\overline{\{s_3,s_4\}} = \{s_1,s_3,s_4\}$, which leads to $Res(S,\alpha,S) = \{\{s_0,s_5,s_6,s_7\}\}$. We have $Refine(\{S\},\alpha,S) = \{\{s_0,s_5,s_6,s_7\},\{s_1,s_3,s_4\},\{s_2\}\}$. This new partition X contains a splitter $(\tau,\{s_2\})$, because $P_X(s_0,\tau,\{s_2\}) = 0.5 \neq 0 = P_X(s_5,\tau,\{s_2\})$. The subsequent refinement step merely seperates s_0 from its former partition, i.e. $Refine(X,\alpha,\{s_2\}) = \{\{s_0\},\{s_5,s_6,s_7\},\{s_1,s_3,s_4\},\{s_2\}\}$. This partition does not contain further splitters, it thus represents the weak bisimulation equivalence classes.

In what follows, $n = |S|$. We suppose that the alphabet *Act* is fixed.

Theorem 8. *The algorithm above can be implemented in time $O(n^3)$ and space $O(n^2)$.*

Proof. In order to avoid multiple computations of the values $P(s,a,C)$ where C is a block in X that has not been changed in the last refinement step we replace the assignment $X := Refine(X,a,C)$ by $Y := Refine(X,a,C)$; $X_{new} := Y \setminus X$;

$X := Y$. (I.e. X_{new} contains the set of blocks that have been modified in the last iteration step. Initially, $X_{new} = \{S\}$.) Initially, $X_{new} = \{S\}$.
Initialization of the refine step: Let X be the current partition. We compute the values $P(s,a,C)$ and $P_X(s,a,C)$ for each $s \in S$, $a \in Act$, $C \in X_{new}$. The set T_X can be derived from the probabilities $P(s,\tau,C)$, $s \in C$. For each pair (a,C) (where $a \in Act$, $C \in X_{new}$) and $A \in X$ we compute $min(A,a,C) = \min_{s \in A} P_X(s,a,C)$ and $max(A,a,C) = \max_{s \in A} P_X(s,a,C)$. Then, (a,C) is a splitter of X iff $min(A,a,C) < max(A,a,C)$ for some A with $a \neq \tau$ if $A = C$. If there is no splitter of X then $X = S/\approx$. Otherwise we choose some splitter (a,C) of X.
Refinement step: For all $B \in X$ with $B \neq C$ if $a = \tau$ we compute the set $Refine(B,a,C)$ as follows. We construct an ordered binary tree $Tree(B)$ by successively inserting the values $P_X(s,a,C)$, $s \in B \cap T_X$. Each node v of $Tree(B)$ is represented as a record with components $v.key$ and $v.states$. $v.key$ is the key value of v (i. e. one of the values $P_X(s,a,C)$, $s \in B \cap T_X$) such that $v.key < w.key$ ($v.key > w.key$) for all nodes w in the right (left) subtree of v. For each state $s \in B \cap T_X$ we traverse the tree $Tree(B)$ starting in the root and search for the value $P_X(s,a,C)$. If we reach a node v with $v.key = P_X(s,a,C)$ then we insert s into $v.states$. Otherwise, $P_X(s,a,C)$ is not yet represented in $Tree(B)$ and we insert a node v with $v.key = P_X(s,a,C)$ and $v.states = \{s\}$. In the final tree, $v.states$ is the set of states $s \in B \cap T_X$ with $P_X(s,a,C) = v.key$. Thus, the nodes of the final tree $Tree(B)$ represent the sets $A \in Split(B,a,C)$. More precisely, $Split(B,a,C)$ consists of the sets $v.states$ where v ranges over all nodes of $Tree(B)$. We derive $Refine(B,a,C)$ as follows. Let G_B be the directed graph (B, E_B) where $(s,t) \in E_B$ iff $P(t,\tau,s) > 0$ and $t \in B \setminus T_X$. We compute the sets $\overline{A}$, $A \in Split(B,a,C)$, by a breadth first search like method: We define $label(s) = A$ for all $s \in A$ and $A \in Split(B,a,C)$ and $label(s) = \perp$ ("undefined" or "not yet visited") for all $s \in B \setminus T_X$. In what follows, we use label $*$ for states that are reachable in G_B from two or more sets $A \in Split(B,a,C)$. Thus, all successors of a $*$-labelled state in G_B are also labelled by $*$. We use a queue Q which initially contains the states $s \in A$, $A \in Split(B,a,C)$. While Q is not empty we take the first element s of Q, remove s from Q and, if $label(s) \neq *$ then for all $t \in B \setminus T_X$ with $(s,t) \in E_B$ we do:

(1) If $label(t) = \perp$ then we add t to Q and set $label(t) = label(s)$.
(2) If $label(t) \in Split(B,a,C)$, $label(t) \neq label(s)$, then we set $label(u) = *$ for $u = t$ and all successors u of t in G_B.

(In step (2), we use a depth first search starting in t to find all successors of t. States that are already labelled by $*$ are ignored.) Then, $\overline{A} = \{s \in B : label(s) = A\}$ and $Res(B,a,C) = \{\{s \in B : label(s) \in \{\perp, *\}\}\} \setminus \{\emptyset\}$.
Complexity: It is clear that the method described above can be implemented in space $\mathcal{O}(n^2)$. We show that the time complexity of our method is $\mathcal{O}(n^3)$. First, we observe that there are at most n iterations of the refinement step. Thus it suffices to show that each refinement step takes time $\mathcal{O}(n^2)$: It is clear that for each refinement step, the initialization requires $\mathcal{O}(n^2)$ time. (For each tuple (s,a,C), one has to calculate the sum $\sum_{t \in C} P(s,a,t)$. Hence, for fixed a and

ranging over all $s \in S$ and $C \in X$ we get the time complexity $\mathcal{O}(n^2)$. Since we suppose *Act* to be fixed, the values $P(s,a,C)$ can be computed in time $\mathcal{O}(n^2)$.) Ranging over all B, the construction of the trees $Tree(B)$ (thus, the computation of the sets $Split(B,A,C)$) takes $\mathcal{O}(n \cdot \log n)$ time if one uses some kind of balanced trees, e.g. AVL-trees [1]. We show that, ranging over all $B \in X$, the sets $\overline{A}$ and $Res(B,a,C)$ can be derived in time $\mathcal{O}(n^2)$: For fixed $B \in X$, the directed graph G_B can be constructed in time $\mathcal{O}(|B|^2)$. Each state $s \in B$ is added to Q at most once. (Note that only states with label $\perp$ can be added to Q.) Each state t which is visited during a depth first search in step (2) is labelled by $*$. Thus, it can never be visited in step (2) once again. As a consequence, each state causes time costs (at most) of order $2n$ in the computation of $Refine(B,a,C)$: as an element of Q and as a state with label $\neq *$ that is visited in step (2). Either case involves $\mathcal{O}(n)$ computations. Summing up over all $s \in B$, the computation of $Refine(B,a,C)$ has time complexity $\mathcal{O}(|B| \cdot n)$. So, we obtain $Refine(X,a,C)$ in time $\mathcal{O}(n^2)$. Thus, we get the overall time complexity $\mathcal{O}(n^3)$.

□

5 Further directions

In this paper we have extended the notions of weak and branching bisimulation equivalence to fully probabilistic transition systems. In contrast to the non-probabilistic case, both relations coincide. We have described an algorithm that computes weak (and branching) bisimulation equivalence classes in time $\mathcal{O}(n^3)$ and space $\mathcal{O}(n^2)$.

Obviously, our notion of equivalence is coarser than strong bisimulation equivalence [26]. In addition, it can be shown that weak bisimulation equivalence is finer than the testing equivalences of [7, 8]. It is also finer than the testing equivalence of [9, 36] that considers τ-free tests only but incomparable with their test equivalence that allows for general tests.

The definition of composition operators for fully probabilistic transition systems is an important subject for further work. In the presence of composition operators, a proper notion of equality should be preserved; that is, it is required that weak bisimulation equivalence is a congruence with respect to the operators. Indeed, prefixing, hiding, restriction and (guarded) probabilistic choice can be easily adopted from the nonprobabilistic to the fully probabilistic setting such that weak bisimulation is a congruence for them, see [4]. Unfortunately it is not straightforward to adapt parallel composition to this framework. Other fully probabilistic calculi like PCCS [12] and similar calculi [18, 27], are based on synchronous CCS [28]. In particular, their parallel composition is synchronous. In the essence, activities (of different components) that may happen with nonzero probability occur synchronously, with a probability given by the product of the individual probabilities. Such synchrony includes internal activities, because they do not play a distinguished role in PCCS. This reflects the lack of a notion of equivalence that abstracts from internal computation. In our framework, it seems promising to allow internal computation to occur asynchronously, similar to the asynchronous product in synchronous CCS. However, the shape of this operator still has to be settled.

We restricted ourselves to fully probabilistic transition systems that are generative in nature. This model is not adequate to represent the truly asynchronous behaviour of concurrent probabilistic processes [34]. For this purpose, some kind of probabilistic transition system is required that allows for nondeterministic branching. As far as the authors know, the question of decidable notions of weak bisimulation is still open in this setting. The weak bisimulation of [33] for such a model is based on a "double arrow relation" that assigns a set of transition probability distributions to each state. In general, this set, representing nondeterministic alternatives, is infinite. This substantially differs from the nonprobabilistic and the fully probabilistic case where weak bisimulation equivalence can be decided using a finitely branching transition system.

References

1. G. Adel'son-Velshii, Y. Landis: An Algorithm for the Organization of Information. Soviet. Math. Dokl. 3, pp 1259-1262, 1962.
2. A. Aziz, V. Singhal, F. Balarin, R. Brayton, A. Sangiovanni-Vincentelli: It usually works: The Temporal Logic of Stochastic Systems. Proc. CAV'95, LNCS 939, pp 155-165, 1995.
3. C. Baier: Polynomial Time Algorithms for Testing probabilistic Bisimulation and Simulation. Proc. CAV'96, LNCS 1102, pp 38-49, 1996.
4. C. Baier, H. Hermanns: Weak Bisimulation for Fully Probabilistic Processes. Techn. Bericht IMMD-VII/1-97, Universität Erlangen.
5. A. Bianco, L. de Alfaro: Model Checking of Probabilistic and Nondeterministic Systems. Proc. Foundations of Software Technology and Theoretical Computer Science, LNCS 1026, pp 499-513, 1995.
6. G. Chehaivbar, H. Garavel, N. Tawbi, F. Zulian: Specification and Verification of the Powerscale Bus Arbitration Protocol: An Industrial Experiment with LOTOS. Formal Description Techniques IX, Chapmann Hall, 1996.
7. I. Christoff: Testing Equivalences and Fully Abstract Models for Probabilistic Processes, Proc. CONCUR'90, LNCS 458, pp 126-140, 1990.
8. L. Christoff, I. Christoff: Efficient Algorithms for Verification of Equivalences for Probabilistic Processes. Proc. CAV'91, LNCS 575, pp 310-321, 1991.
9. R. Cleaveland, S. Smolka, A. Zwarico: Testing Preorders for Probabilistic Processes. Proc. ICALP'92, LNCS 623, pp 708-719, 1992.
10. D. Coppersmith, S. Winograd: Matrix Multiplication via Arithmetic Progressions. Proc. 19th ACM Symposium on Theory of Computing, pp 1-6, 1987.
11. C. Courcoubetis, M. Yannakakis: Verifying Temporal Properties of Finite-State Probabilistic Programs. Proc. FOCS'88, pp 338-345, 1988.
12. A. Giacalone, C. Jou, S. Smolka: Algebraic Reasoning for Probabilistic Concurrent Systems. Proc. IFIP TC2 Working Conf. on Programming Concepts and Methods, 1990.
13. R. van Glabbeek, S.A. Smolka, and B. Steffen: Reactive, generative, and stratified models of probabilistic processes. Information and Computation, Vol 121, pp 59-80, 1995.

14. R. van Glabbeek, W. Weijland: Branching Time and Abstraction in Bisimulation Semantics. Journal of the ACM 43(3), pp. 555-600,1996.
15. J. Groote, F. Vaandrager: An Efficient Algorithm for Branching Bisimulation and Stuttering Equivalence. Proc. ICALP'90, LNCS 443, 1990.
16. P. Halmos: Measure Theory. Springer-Verlag, 1950.
17. H. Hansson: Time and Probability in Formal Design of Distributed Systems. Ph.D.Thesis, Uppsala University, 1994.
18. H. Hansson, B. Jonsson: A Calculus for Communicating Systems with Time and Probabilities. Proc. IEEE Real-Time Systems Symposium, 1990.
19. H. Hansson, B. Jonsson: A Logic for Reasoning about Time and Probability. Formal Aspects of Computing, Vol. 6, pp 512-535, 1994.
20. S. Hart, M. Sharir: Probabilistic Temporal Logic for Finite and Bounded Models. Proc. 16th ACM Symposium on Theory of Computing, 1984.
21. T. Huynh, L. Tian: On some Equivalence Relations for Probabilistic Processes. Fundamenta Informaticae, Vol. 17, pp 211-234, 1992.
22. B. Jonsson, K.G. Larsen: Specification and Refinement of Probabilistic Processes. Proc. LICS'91, pp 266-277,1991.
23. B. Jonsson, W. Yi: Compositional Testing Preorders for Probabilistic Processes. Proc. LICS'95, pp 431-443, 1995.
24. C.C. Jou, S. Smolka: Equivalences, Congruences and Complete Axiomatizations for Probabilistic Processes. Proc. CONCUR'90, LNCS 458, pp 367-383, 1990.
25. P. Kanellakis, S. Smolka: CCS Expressions, Finite State Processes, and Three Problems of Equivalence. Information and Computation, Vol. 86, pp 43-68, 1990.
26. K. Larsen, A. Skou: Bisimulation through Probabilistic Testing. Information and Computation, Vol. 94, pp 1-28, 1991.
27. K. Larsen, A. Skou: Compositional Verification of Probabilistic Processes. Proc. CONCUR'92, LNCS 630, pp 456-471, 1992.
28. R. Milner: Calculi for Synchrony and Asynchrony. Theoretical Computer Science, Vol. 25, pp 269-310, 1983.
29. R. Milner: Communication and Concurrency. Prentice Hall, 1989.
30. R. Paige, R. Tarjan: Three Partition Refinement Algorithms. SIAM Journal of Computing, Vol. 16, No. 6, pp 973-989, 1987.
31. A. Pnueli, L. Zuck: Verification of Multiprocess Probabilistic Protocols. Distributed Computing, Vol. 1, No. 1, pp 53-72, 1986.
32. A. Pnueli, L. Zuck: Probabilistic Verification. Information and Computation, Vol. 103, pp 1-29, 1993.
33. R. Segala, N. Lynch: Probabilistic Simulations for Probabilistic Processes. Proc. CONCUR 94, LNCS 836, pp 481-496, 1994.
34. M. Vardi: Automatic Verification of Probabilistic Concurrent Finite-State Programs. Proc. FOCS'85, pp 327-338, 1985.
35. M. Vardi, P. Wolper: An Automata-Theoretic Approach to Automatic Program Verification. Proc. LICS'86, pp 332-344, 1986.
36. S. Yuen, R. Cleaveland, Z. Dayar, S. Smolka: Fully Abstract Characterizations of Testing Preorders for Probabilistic Processes. Proc. CONCUR'94, LNCS 836, pp 497-512, 1994.
37. W. Yi, K. Larsen: Testing Probabilistic and Nondeterminsitic Processes. Protocol Specification, Testing and Verification XII, Elsevier Science Publishers, IFIP, pp 47-61, 1992.