On the verification of Probabilistic I/O Automata with unspecified rates*

Sergio Giro
FaMAF, Universidad Nacional de Córdoba, Argentina and CONICET
sgiro@famaf.unc.edu.ar

Pedro R. D'Argenio
FaMAF, Universidad Nacional de Córdoba, Argentina and CONICET
dargenio@famaf.unc.edu.ar

ABSTRACT

We consider the Probabilistic I/O Automata framework, for which we address the verification of reachability properties in case the rates (also called *delay parameters*) are unspecified. We show that the problem of finding (or even approximating) the supremum probability that a set of states is reached is undecidable. However, we give an algorithm to obtain a non-trivial over-estimation of this value. We explain why this over-estimation may result useful for many systems. Finally, in order to compare our approach against Markov Decision Processes, we study a simple protocol for anonymous fair service. In this case, the over-estimation computed over the PIOA gives a more realistic result than the exact computation over the MDP.

Keywords

model checking, probabilistic automata, distributed systems

1. INTRODUCTION

Probabilistic I/O Automata (PIOA) are probabilistic state machines introduced in [12] with the aim of providing a framework for modelling asynchronous probabilistic systems (for relevant results in this framework, see [10, 8, 11, 9]). In this paper, we consider the verification of *distributed probabilistic systems*, in which several entities evolve in an independent fashion and ocasionally exchange some information. For such systems, we address the nondeterminism introduced by the fact that the order in which entities execute is not specified (we refer to this concept as the *interleaving* nondeterminism). As in the PIOA setting, we do *not* consider the *internal* nondeterminism introduced by the existence of several enabled transitions in the same entity. PIOA as in [12] are useful to model systems in which a *rate* (called *delay parameter* in [12] and *rate function* in subsequent works, e.g. [9]) is known for each state in each entity. Given the actual state in each entity, the rates are used to calculate the probability that a given entity is the next one to perform an action. Here, we address the verification of systems in which the rates are not known beforehand and, moreover, rates may vary during the course of execution. Our formalism allows to specify separate entities, each one having partial knowledge of the global state of the system. This ability is essential in order to obtain accurate results during the analysis of distributed systems [5].

In Sec. 5 we show how our results complement previous literature on probabilistic distributed systems and scheduling.

Contributions. In this paper, we address the nondeterminism concerning the different interleavings in which entities perform their actions. We present a scheduling mechanism based on the *rate function* for PIOA.

Clearly, our framework allows to model some systems that cannot be modelled using the framework in [12]. We show that the model checking problem is undecidable when rates are unspecified. Moreover, the worst-case probability with which a set of states is reached cannot be even approximated. However, we present an algorithm to obtain an over-estimation of this worst-case probability. This algorithm works by translating the PIOA under consideration to a *Markov decision process* (MDP). Our algorithm has an intuitive explanation, and we explain why the over-estimation obtained with our algorithm is more realistic than the probability obtained by modelling the system directly as an MDP and applying the standard model checking algorithm in [1]. This improvement is related to the fact that, for distributed systems, the algorithm in [1] yields unrealistic probabilities, since MDPs assume total knowledge, while entities in distributed systems have access only to local information [5]. In order to illustrate the advantages of our algorithm against traditional model checking for MDPs, we present the results we obtained during the analysis of a simple protocol for anonymous fair service.

2. PROBABILISTIC I/O AUTOMATA

We present the PIOA framework in terms of reactive and generative structures [6], thus resembling the switched PIOA in [2] (see Sec. 5 for a detailed comparison). However, our framework is equivalent to the original PIOA in [12] –except for the facts that rates are not specified and the amount of states is finite– having no internal nondeterminism. For a finite set S, we denote by $\mathsf{Dist}(S)$ the set of all probability

*Supported by ANPCyT project PICT 26135 and CONICET project PIP 6391. An extended version of this paper (including proofs and pointers to PRISM code) can be found at `cs.famaf.unc.edu.ar/~sgiro/GD09-sac.pdf`

SAC'09 March 8-12, 2009, Honolulu, Hawaii, U.S.A.

A generative structure with two transitions A reactive structure with two transitions

Figure 1: Reactive and generative structures

distributions over the set S. Given a set Act of action labels and a set S of states, the set of generative transitions T_G on $(\mathsf{S}, \mathsf{Act})$ is $\mathsf{Dist}(\mathsf{S} \times \mathsf{Act})$, and the set T_R of reactive transitions is $\mathsf{Dist}(\mathsf{S})$. A generative structure on $(\mathsf{S}, \mathsf{Act})$ is a function $G : \mathsf{S} \to \mathcal{P}(T_G)$ and a reactive structure on $(\mathsf{S}, \mathsf{Act})$ is a function $R : \mathsf{S} \times \mathsf{Act} \to \mathcal{P}(T_R)$. Figure 1 depicts an example of these structures. Generative transitions model both communication and state change. The component executing a generative transition chooses both a label a to output (the ! indicates that the label is output) and a new state s according to a given distribution. Reactive transitions specify how a component reacts to a given input (the ? represents input). Since the input is not chosen, reactive transitions are simply distributions on states.

We consider systems obtained by composing several *probabilistic I/O atoms*. Each atom is a probabilistic I/O automaton having reactive and generative transitions. Since we are not interested in internal nondeterminism, we restrict to *deterministic* structures. A generative (reactive, resp.) structure is said to be *deterministic* iff, for all s and a it holds $|G(s)| \leq 1$ (or $|R(s,a)| \leq 1$, resp.) Because of this restriction, we often use $G(s)$ to denote the sole element in the set $G(s)$ (provided that there is one) and $R(s,a)$ to denote the sole element in the set $R(s,a)$.

Definition 1. A probabilistic I/O atom is a 5-tuple $(\mathsf{S}, \mathsf{Act}, G, R, \mathsf{init})$, where S is a finite set of states, Act is a finite set of actions labels, and G (R, resp.) is a determistic generative (reactive, resp.) structure on $(\mathsf{S}, \mathsf{Act})$. $\mathsf{init} \in \mathsf{S}$ is the initial state. As in [12], we require the atoms to be input-enabled, so $R(s,a) \neq \emptyset$ for every $s \in \mathsf{S}$, $a \in \mathsf{Act}$ (note that there is a set Act for each atom). We often write S_i to denote the set of states of an atom A_i and similarly for the other elements of the 5-tuple. In addition, we write T_{G_i} (T_{R_i}, resp.) for the set of generative (reactive, resp.) transitions on $(\mathsf{S}_i, \mathsf{Act}_i)$.

We define an interleaved probabilistic I/O system P as a set Atoms(P) of probabilistic I/O atoms $A_1, \cdots, A_N$. The set of states of the system is $\prod_i \mathsf{S}_i$, and the initial state of the system is $\mathsf{init} = (\mathsf{init}_1, \cdots, \mathsf{init}_N)$.

In order to define how the system evolves, we define *compound transitions*, which are the transitions performed by the system as a whole. In such compound transitions, all the atoms having the same action label in their alphabet must synchronize and *exactly one* of them must participate with an output (generative) transition (thus modelling multicasting). Formally, a compound transition is a tuple $(g_i, a, r_{j_1}, \cdots, r_{j_m})$ (we require $i \neq j_k$ and $j_k \neq j_{k'}$ for all $k \neq k'$) where g_i is a generative transition in the atom A_i (the *active* atom), $a \in \mathsf{Act}_i$ is an action label, the r_{j_k} are reactive transitions in the atoms A_{j_k} (the *reactive* atoms) and $\{A_i, A_{j_1}, \cdots, A_{j_m}\}$ is the set of all the atoms A_j such that $a \in \mathsf{Act}_j$. We say that A_i, A_{j_1}, ..., A_{j_m} are the atoms *involved* in the compound transition. A compound transition $c = (g_i, a, r_{j_1}, \cdots, r_{j_m})$ is enabled in a given state $(s_1, \cdots, s_N)$ if $g_i = G_i(s_i)$ and $r_{j_k} = R_{j_k}(s_{j_k}, a)$. The action label a in c is indicated by $label(c)$. The atom A_i that performs the output is indicated as $\mathsf{outAtom}(c)$. Note that, since we restrict to deterministic structures, for all s, i, a there is at most one c such that $\mathsf{outAtom}(c) = A_i$, and $label(c) = a$, and c is enabled in s. The (sub)probability $c(s, s')$ of reaching $s' = (s'_1, \cdots, s'_N)$ from $s = (s_1, \cdots, s_N)$ using $c = (g_i, a, r_{j_1}, \cdots, r_{j_m})$ is $g_i(s'_i, a) \cdot \prod_{k=1}^{m} r_{j_k}(s'_{j_k})$ if $s_t = s'_t$ for every atom not involved in the transition. Otherwise, $c(s, s') = 0$.

So, for all A we have $\sum_{\{c | \mathsf{outAtom}(c) = A\}} \sum_s c(s, s') = 1$.

In order to ease some definitions, we introduce a fictitious "stutter" compound transition ς. Intuitively, this transition is executed iff the system has reached a state in which no atom is able to generate a transition. The probability $\varsigma(s, s')$ of reaching s' from s using ς is 1, if $s = s'$, or 0, otherwise.

A path σ of P is a sequence $s_1.c_1.s_2.c_2 \cdots c_{n-1}.s_n$ where each s_i is a (compound) state and each c_i is a compound transition such that c_i is enabled in s_i and $c(s_i, s_{i+1}) > 0$. A path can be finite or infinite. For a finite path σ as before, the *set of extensions* (denoted by $[\sigma]$) contains all the infinite paths starting with σ. We define $\mathrm{last}(\sigma) = s_n$ and $\mathrm{len}(\sigma) = n$.

In [12], the authors propose a mechanism that resolves the interleaving choices for PIOA in a distributed fashion: a *rate* $\mathsf{rt}(s)$ is assigned to each state s in each entity. Using such rate, the choice among all the entities that are able to perform an action is transformed into a probabilistic choice as follows: when an entity arrives in state s, it draws a random delay time from an exponential distribution with parameter $\mathsf{rt}(s)$ (i.e. an exponential distribution whose mean is $1/\mathsf{rt}(s)$). This time describes the length of time the entity will remain in state s before executing an action. So, the entity having the least amount of delay time left is the next one to perform an action. In [12] it is explained that, according to this interpretation of the rate, a definite probability can be assigned to the event in which a given entity is the next one to perform an action in a given state. If each entity i is in state s_i, the probability that j is the next entity to perform an action is $\mathsf{rt}(s_j) \,/\, \sum_i \mathsf{rt}(s_i)$. The same mechanism is used also in [10, 8, 11, 9]. As a simple example, if entity E_1 (E_2, resp.) is in state s_1 (s_2, resp.) and $\mathsf{rt}(s_1) = 1$ and $\mathsf{rt}(s_2) = 2$, the probability that E_1 executes first is $1/(1+2)$ while the probability that E_2 executes first is $2/(1+2)$. Note that the rate of s_2 is twice the rate of s_1, and this is reflected in the probabilities. In general, the rate can be seen as a "likeliness factor" that indicates how likely is an entity to execute with respect to another.

This mechanism is useful in case we know the rate for each state. However, in a fully nondeterministic setting, it may be impossible to know in advance how much time an entity will delay in a certain state. Moreover, the delayed amount may vary during the execution of the system, and so the rate should be a function of the whole execution of the system upto the actual state, and not only a function of the actual state.

So, we consider *rate schedulers* for each atom. Rate schedulers choose a rate for every possible path in each atom. We want the rates for each atom to depend solely on the information available to that atom (this restriction will be shown useful later, when comparing PIOA over MDPs). Such information is modelled by the *projection* of a path over an atom.

Given a path σ, the projection $\sigma[i]$ of the path σ over an atom A_i is defined inductively as follows: **(1)** $\mathsf{init}[i] = \mathsf{init}_i$,

(2) $\sigma.c.s[i] = \sigma[i]$ if A_i is not involved in c, and **(3)** $\sigma.c.s[i] = \sigma[i].label(c).\pi_i(s)$ (where π_i denotes the i-th projection of a tuple) otherwise. The set of all the projections of paths over an atom A_i is denoted by $\mathsf{Proj}_i(P)$. We say that these projections are the *local paths* of A_i.

A *rate scheduler* for A_i is a function $\mathsf{rt}_i : \mathsf{Proj}_i(P) \to \mathbb{R}_{\geq 0}$ [1] such that $\mathsf{rt}_i(\sigma_i) = 0$ iff $G(\text{last}(\sigma_i)) = \emptyset$. A rate scheduler η for a PIOA composed of several atoms $\{A_i\}_{i=1}^N$ is a set of rate schedulers $\{\mathsf{rt}_i\}_{i=1}^N$, where each rt_i is a rate scheduler for the atom A_i. Using such schedulers, we can define probabilities for the extension sets.

Definition 2. Given a scheduler $\eta = \{\mathsf{rt}_i\}_{i=1}^N$, the probability $\Pr^\eta([\mathsf{init}])$ of the extensions of the initial state is 1. If there exists i s.t. $G_i(\text{last}(\sigma)) \neq \emptyset$, then the probability $\Pr^\eta([\sigma.c.s])$ is

$$\Pr^\eta([\sigma]) \cdot \frac{\mathsf{rt}_{\mathsf{outAtom}(c)}(\sigma[\mathsf{outAtom}(c)])}{\sum_j \mathsf{rt}_j(\sigma[j])} \cdot c(\text{last}(\sigma), s) \ .$$

If there is no such i, then the system cannot generate any transition. In this case, we let $\Pr^\eta([\sigma.c.s]) = \Pr^\eta([\sigma])$ if $c = \varsigma$ and $s = \text{last}(\sigma)$, or 0 otherwise.

Using the assumption that the structures are deterministic, we have $\sum_{c,s'} \frac{\mathsf{rt}_{\mathsf{outAtom}(c)}(\sigma[\mathsf{outAtom}(c)])}{\sum_j \mathsf{rt}_j(\sigma[j])} \cdot c(\text{last}(\sigma), s') = 1$. This probability can be extended to the least σ-field containing all the sets of extensions in the standard way. We say that the sets in such σ-field are *measurable*. Given a measurable set S, we are interested in the value $\sup_\eta \Pr^\eta(S)$. By calculating this amount it can be answered, for instance, whether or not "the probability of a package loss is at most 0.05". This property, in particular, is what we call a *reachability property*: we are interested in the set of paths in which some states are reached (namely, the states in which a package has been lost). Given a set U of states, we denote by $\Pr^\eta(\text{reach}(U))$ the probability of reaching any state in U.

3. VERIFYING PIOA

Unfortunately, the model checking problem is undecidable for PIOA with unknown rate parameters. In fact, the worst-case probability that a set of states is reached cannot be even approximated with arbitrary precision.

Theorem 1. Given a PIOA P, a set U of states, and $\epsilon > 0$, there is no algorithm to compute r such that

$$\left| \sup_\eta \Pr^\eta(\text{reach}(U)) - r \right| \leq \epsilon \ .$$

However, we found an algorithm whose result is a non-trivial over-estimation of the amount $\sup_\eta \Pr^\eta(\text{reach}(U))$ (or an under-estimation of $\inf_\eta \Pr^\eta(\text{reach}(U))$). For instance, if the algorithm over-estimates the value of $\sup_\eta \Pr^\eta(\text{reach}(U))$ as 0.2, it may be the case that the supremum is actually 0.1 (but it *cannot* be the case that it is 0.3). So, if 0.2 is an acceptable probability, we have automatically proven that the system under consideration is correct. It is important to note that, because of the way in which the algorithm works, there are many systems in which this over-estimation is actually better than the value obtained using MDPs with omniscient schedulers as in [1]. In fact, our algorithm is able to find realistic worst-case probabilities when analysing a protocol for anonymous fair service, as we explain in Sec. 4. We explain how our algorithm works using the following example.

Suppose that ABC Corp. is planning two meetings. Each member of ABC Corp. must assist to exactly one of these meetings. ABC Corp. has a business and a technical division has two divisions. The aim of ABC Corp. is to foster collaboration among people belonging to different divisions. They plan to assign a meeting to each member using a computer program. Such program randomly selects one of the meetings (with probability 1/2 each) without showing the selection. Each member of ABC Corp. is required to use the program during the course of the week, in order to know the meeting which he/she is assigned to. When a member asks the program, the program assigns to this member the meeting previously selected. Then, the program randomly selects another meeting to be assigned to the next member.

We would like the members in each meeting to be well-balanced with respect to the division they belong to. Suppose that a member of ABC Corp. decides to check the probability that all members of the technical division are assigned to the same meeting by analysing all possible orderings for the members of ABC Corp. Then, he discovers that, if the members ask the program in order $m_1, \cdots, m_N$ (where N is the total number of members) then the member m_i is assigned with probability 1/2 to each meeting, for all i. So, the probability that all members of the technical division are assigned to the same meeting is $1/2^{N_T}$ for all cases, where N_T is the number of members of the technical division. Roughly speaking, the order is selected beforehand and the system is verified by assuming that the atoms execute in the selected order. This analysis must be carried out for every possible order.

Using this idea, we explain how to translate PIOA to MDPs. An *MDP* is a tuple $M = (\mathsf{S}, \mathsf{Tran}, \mathsf{P}, \mathsf{init})$, where S is a finite set of states, Tran is a finite set of transition names, $\mathsf{P} : (\mathsf{S} \times \mathsf{Tran} \times \mathsf{S}) \to [0, 1]$ is the (three-dimensional) probability matrix, $\mathsf{init} \in \mathsf{S}$ is the initial state. $\mathsf{Tran}(s)$ denotes the set of transitions enabled in state s, i.e. the set of transitions $\alpha \in \mathsf{Tran}$ such that $P(s, \alpha, t) > 0$ for some $t \in \mathsf{S}$. For every state $s \in \mathsf{S}$, we require that $\mathsf{Tran}(s) \neq \emptyset$ and $\sum_{s' \in \mathsf{S}} \mathsf{P}(s, \alpha, s') = 1$ for every transition $\alpha \in \mathsf{Tran}(s)$. A path is a sequence $s_1.\alpha_1.\cdots.s_n$. A scheduler η for an MDP is a function mapping paths to transitions. The probability of a path $s_1.\cdots.s_n$ is $\prod_{i=1}^{n-1} \mathsf{P}(s_i, \eta(s_1.\cdots.s_i), s_{i+1})$. So, the scheduler resolves the nondeterministic choice among all the enabled transitions.

In the following, we show how to construct an MDP M from a given PIOA P. The MDP M starts with a nondeterministic choice. This choice selects a total order on the atoms having enabled generative transitions in the their initial state. (In general, for any state s, we call these atoms the *enabled* atoms –denoted by $\mathsf{enAtoms}(s)$). The interpretation of such order is that the atoms will perform outputs according to it. After this first choice has been performed, M has several available transitions. All of these transitions execute the generative structure corresponding to the atom on the top of the order, but, in addition, each transition determines how the atoms are ordered after the transition has

[1] All of our results are also valid if the range of the schedulers (and/or the range of the generative/reactive structures) is the set of rational numbers.

been executed. If the order prior to the execution is o_1, then the new order o_2 must comply $A_i <_{o_1} A_j \implies A_i <_{o_2} A_j$ for all A_i, A_j such that the label l output by the generative transition is neither in Act_i nor in $\mathsf{Act}\, j$. That is, the order of atoms that are not aware of l does not change. By repeating this mechanism, M picks the maximum atom (according to the actual order) and reorders the atoms. Note that the restriction on the reordering ensures that, if at some point of the execution it holds that $A_i <_o A_j$, then A_j will not execute after A_i unless A_i or A_j get new information.

In general, the decision of whether A_i executes before A_j is taken right after the last step in which A_i or A_j receives information. So, this choice cannot depend on subsequent probabilistic choices and, as a consequence, there are systems for which our algorithm calculates realistic probabilities. We offer an example of such systems in Sec. 4.

Let $\mathcal{O}_P$ be the set comprising all total orders on subsets of Atoms(P). Moreover, given $S \subseteq$ Atoms(P), let $\mathcal{O}(S)$ be the set comprising all total orders on S. Given an order o, a *reordering function* for o is a function $r_o : \mathsf{Act} \times \prod_i \mathsf{S}_i \to \mathcal{O}_P$ such that: **(1)** $r_o(a, s) \in \mathcal{O}(\mathsf{enAtoms}(s))$ and **(2)** $(A_i <_o A_j \wedge a \notin \mathsf{Act}_i \wedge a \notin \mathsf{Act}_j) \implies \forall s \,.\, A_i <_{r_o(a,s)} A_j$. Let $\mathcal{R}_o$ denote all the reordering functions for o.

Given an interleaved PIOA P we construct an MDP M as follows. Since MDPs have no concept of action labels, we encode the last action label as part of the state. In addition, the current order on the enabled atoms is also part of the states. The set of states of M is $\mathsf{S} = (\,(\mathsf{Act} \cup \{a_{\text{Init}}\}) \times (\prod_i \mathsf{S}_i) \times \mathcal{O}_P\,) \cup \{\mathsf{init}_M\}$, where a_{Init} is a fictitious label introduced because the initial state of P has no previous label, and init_M is the initial state of M. $\mathsf{Tran}(\mathsf{init}_M) = \{t_{\mathsf{init}_M,o} \mid o \in \mathcal{O}(\mathsf{enAtoms}(\mathsf{init}_P))\}$, with $\mathsf{P}(\mathsf{init}_M, t_{\mathsf{init}_M,o}, (a_{\text{Init}}, \mathsf{init}_P, o)) = 1$. For the remaining states, $\mathsf{Tran}((a, s, o)) = \{t_{s,o,r_o} \mid r_o \in \mathcal{R}_o\}$, where t_{s,o,r_o} is defined as follows. Given a state $s = \langle s_j \rangle_j$, an enabled atom i and a label a', let $c_{s,i,a'}$ be the compound transition $(G_i(s_i), a', \langle R_j(s_j, a') \rangle_{j \mid a' \in \mathsf{Act}_j})$ (for G_i and R_j, see Def. 1). Then, $\mathsf{P}((a, s, o), t_{s,o,r_o}, (a', s', o')) = c_{s,\max_{<_o} A_i,a'}(s, s')$ if $r_o(a', s') = o'$. Otherwise, $\mathsf{P}((a, s, o), t_{s,o',r}, (a', s', o'')) = 0$. For the states in which $\mathsf{enAtoms}(s) = \emptyset$, we define $\mathsf{Tran}(s) = \{t_\varsigma\}$ with $\mathsf{P}(s, t_\varsigma, s) = 1$.

Our algorithm simply translates the PIOA P to an MDP M as explained above and applies the algorithm in [1]. Given a set of states of the PIOA U, let $M(U)$ denote the set of states of M defined as $M(U) = \{(a, s, o) \mid s \in U\}$. The soundness of our algorithm is stated in the following theorem.

Theorem 2. Given a PIOA P, let M be the MDP constructed as explained above. Then,

$$\sup_\eta \Pr_P^\eta(\text{reach}(U)) \leq \sup_\eta \Pr_M^\eta(\text{reach}(M(U)))$$
$$\text{and} \quad \inf_\eta \Pr_M^\eta(\text{reach}(M(U))) \leq \inf_\eta \Pr_P^\eta(\text{reach}(U)) \,.$$

If we model a system using PIOA and apply our algorithm, instead of modelling the system with the obvious MDP, we may obtain a more realistic verification of the system. As an example, the system of ABC Corp. can also be modelled using MDPs. In this case, the choice of the next member to ask the program needs to be resolved by the scheduler. Since schedulers for MDPs are able to look at the whole history of the system (and not only at the projections), the scheduler may choose a member according to the hidden outcome of the probabilistic choice, and then there exists a scheduler that assigns all members of the technical division to the same party with probability 1 [2]. Note that the naive analysis for the system of ABC Corp. (in which all the orders on members are considered) is valid if all the possible cases to consider are all the orders on the set of members. Such set of cases, in turn, does not rule out realistic cases iff we assume that the order on the members is not a function of the secret random choices, which is exactly what we want.

4. ANALYSING A PROTOCOL

We used our algorithm to analyse a simple protocol for anonymous fair service. A server must serve two clients in a fair fashion regardless of the rates at which they ask for service. In addition, the clients cannot be identified, so the server cannot simply count how many times it has served each of the clients. A rough sketch of the protocol is the following: the server keeps track of the order in which requests were received. At most two requests may be pending, since we assume that clients cannot perform requests while waiting [3]. So, once two requests were received, a coin is tossed in order to decide which of the requests is replied: in case the coin lands heads, the first request is replied. Otherwise, the server replies the second request. Then, the coin is tossed again.

Different results may be obtained by changing what the server does in case a client is waiting while the other one is being served. In this case, the server may serve the client that is waiting (instead of tossing the coin again) once the service for the current one has finished. By avoiding the coin toss, the server increases its throughput, but our analysis shows that the fairness of the service may result affected. For brevity, we name the protocol that tosses a new coin as AFS_1, and the protocol that serves the waiting process as AFS_2.

We modelled the system as a PIOA. The corresponding MDP was constructed by hand. Since this construction is an error-prone task, we performed several consistence verifications on the MDP obtained. All of the verifications were carried out using the PRISM [7] model checker.

Since we focus on reachability properties (and we are still not able to verify long-run properties), we specify the system so that it stops after one of the clients is served 20 times. Then, we used PRISM to calculate the maximum probability p_m that, at any point of execution, the amount n_1 of replies to client 1 is greater than or equal to $n_2 + m$, where n_2 is the amount of replies to client 2 and m is a parameter of the property.

By ignoring the input/output restrictions, the original PIOA can be seen as an MDP. In our case, such MDP is a natural model of the system. Figure 2 compares the probabilities p_m for such MDPs against the probabilities for AFS_1 and AFS_2 over-estimated using our technique.

From the figure, it is clear that AFS_1 ensures a fair service with greater probability than that of AFS_2 (according to our analysis). In addition, the results for the MDP obtained

[2] In [5], a similar example is used to explain why it is unrealistic to assume that a single omniscient scheduler resolves the choices of all the entities. The same kind of unrealistic behaviours (in which the hidden state of an entity affects the behaviour of another entity) are also used to illustrate the lack of compositionality properties [2, Cpt. 8].

[3] Otherwise, it is impossible to guarantee fairness, since one of the entities may perform requests at an arbitrarily high rate.

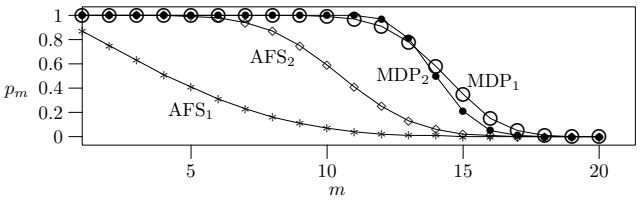

Figure 2: Results of the analysis

by ignoring the input/output restrictions of the PIOA are excessively pessimistic. Note that, for $m = 10$, the worst-case for AFS_1 using the translated PIOA yields a probability of 0.07, while using the MDP model the probability is 0.99. Although the results for AFS_2 are not as encouraging as the results for AFS_1, they are still notably better than the results for the MDPs.

5. RELATED AND FUTURE WORK

In recent literature on distributed systems [4, 5, 2], internal nondeterminism is considered. That is, an entity may have many probabilistic transitions enabled, and so the scheduler must choose also among these transitions. These works also point out that there are schedulers that do not represent realistic behaviours. As a consequence, some acceptable systems are deemed incorrect by model checking algorithms, because these algorithms find unacceptable probabilities yielded by unrealistic schedulers. When we see the execution of an MDP as a game, these unrealistic schedulers arise from the fact that the scheduler is an adversary looking at the whole state of the system, although in the real system the choices in each entity depend only on the information available to the entity. So, a more realistic verification of such distributed systems can be attempted by considering *distributed* schedulers. In this approach, local schedulers are defined for each entity. These schedulers act as adversaries that are able to see the complete execution history of their respective entity, and then choose a transition for this entity. A distributed scheduler is then obtained as a composition of the local schedulers.

Although the existing approaches related to distributed schedulers tackle the problem related to the choices of each entity, the choice of the next entity to perform an action is not resolved by schedulers. In other words, local schedulers choose what their respective entity will do, but there is no scheduler to choose the next entity. In [3], the entities are not specified explicitly (then, there are no interleaving issues) and the schedulers are restricted by imposing the condition that they must observe only a portion of each state in the history. The framework in [4, 5] allows to model completely synchronous systems: a step of the whole system is obtained by taking a step in every entity (thus, no interleaving is needed). In the framework presented in [2] the different entities have local schedulers to resolve internal nondeterminism, and a token is used in order to decide the next entity to perform an action. The interleaving among different entities is not resolved by the schedulers, since the way in which the token is passed is part of the specification. Note that, because of the internal nondeterminism, the choice of the next entity to execute is still nondeterministic, since there may be different transitions passing the token to different entities. However, since internal nondeterminism is resolved according to the local history, the choice of the next entity to execute is based on the history of the entity that passes the token. In [2] it is suggested that a fictitious arbiter entity can be added in order to specify interleaving policies. The entities pass the token to the arbiter and the arbiter selects one of the entities to which the token is passed. So, the information used to choose the next entity can be restricted simply by restricting the information available to the arbiter. Although this approach is useful in order to keep some information hidden, such approach cannot be used to simulate the rate schedulers we present. In our restriction, the lack of information depends on the set of entities considered, and there is no information completely hidden.

In the future, we plan to extend the verification to other properties, as well as to develop a program to automatically translate PIOA to MDPs.

6. REFERENCES

[1] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *FSTTCS '95*, pages 288–299. Springer, 1995.

[2] L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. PhD thesis, Radboud Universiteit Nijmegen, 2006.

[3] L. de Alfaro. The verification of probabilistic systems under memoryless partial-information policies is hard. In *Proc. of PROBMIV 99. Technical Report CSR-99-8*, pages 19–32. University of Birmingham, 1999.

[4] L. de Alfaro, T. A. Henzinger, and R. Jhala. Compositional methods for probabilistic systems. In *CONCUR 01*, pages 351–365. Springer, 2001.

[5] S. Giro and P. R. D'Argenio. Quantitative model checking revisited: neither decidable nor approximable. In *FORMATS '07*, pages 179–194. Springer, 2007.

[6] R. v. Glabbeek, S. Smolka, and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Inf. & Comp.*, 121:59–80, 1995.

[7] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. of TACAS '06*, pages 441–444. Springer, 2006.

[8] E. Stark and G. Pemmasani. Implementation of a compositional performance analysis algorithm for probabilistic I/O automata. In *Proc. of PAPM '99*, pages 3–24. Prensas Universitarias de Zaragoza, 1999.

[9] E. W. Stark. On behaviour equivalence for probabilistic I/O automata and its relationship to probabilistic bisimulation. *J. Autom. Lang. Comb.*, 8(2):361–395, 2003.

[10] E. W. Stark, R. Cleaveland, and S. A. Smolka. A process-algebraic language for probabilistic I/O automata. In R. M. Amadio and D. Lugiez, editors, *CONCUR '03*, pages 189–203. Springer-Verlag, 2003.

[11] E. W. Stark and S. Smolka. Compositional analysis of expected delays in networks of probabilistic I/O automata. In *LICS 98*, pages 466–477. IEEE CS Press, 1998.

[12] S.-H. Wu, S. A. Smolka, and E. W. Stark. Composition and behaviors of probabilistic I/O automata. *Theor. Comput. Sci.*, 176(1-2):1–38, 1997.