Probabilistic Environments in the Quantitative Analysis of (Non-Probabilistic) Behaviour Models

Esteban Pavese, Víctor Braberman
Universidad de Buenos Aires
{epavese,vbraber}@dc.uba.ar

Sebastian Uchitel
Universidad de Buenos Aires and
Imperial College London
suchitel@dc.uba.ar

ABSTRACT

System specifications have long been expressed through automata-based languages, enabling verification techniques such as model checking. These verification techniques can assess whether a property holds or not, given a system specification. *Quantitative* model checking can provide additional information on the probability of these properties holding. We are interested in quantitatively analysing the probability of errors in non-probabilistic system models by composing them with probabilistic models of the environment. Although many probabilistic automata-based formalisms and composition operators exist, these are not adequate for such a setting. In this work we present a formalism inspired on interface automata and a suitable composition operator for these automata that enables validation of environment models in isolation and sound analysis of its composition with the non-probabilistic model of the system-under-analysis.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements / Specifications

General Terms

Design, Verification

Keywords

Behaviour models, probability, interface automata, model checking

1. INTRODUCTION

Modelling languages are envisioned with the objective of capturing and conveying certain aspects of a system's design, many times resorting to diverse languages to elucidate separate aspects of said design. In the realm of software engineering in particular, many such languages have been introduced into general use including automata-based languages which have the advantage of being simple enough to be used as a means to exhibit design and documentation, but also formal enough to be used as an artefact amenable to automated techniques for supporting validation and verification.

Automated techniques that explore automata-based models in order to gain increased assurance regarding the absence of errors have been investigated for some time. A notable example is model checking [4] where an exhaustive search of the model yields, in its most basic and widespread form, a yes/no response to the question of whether the model satisfies a specific property.

Although obtaining binary results from model checkers has been shown to be useful for validation and verification, when the model checker returns a negative answer this can represent insufficient information. This is acknowledged, for instance, in the software reliability community where the interest is not in whether certain properties hold but measuring the likelihood of them being violated. In particular, it is useful and interesting to be able to analyse properties that fail to be universally true in the model of the system under assumptions on how the sofware shall be used by its environment [14].

Musa [14] and others [15, 8] attempt to analyse erroneous system behaviour quantitatively by modelling the probabilistic behaviour of the system's environment, so as to understand the influence that the environment's probabilistic behaviour exerts over the system and quantify this influence.

How to accurately measure and model the probabilistic behaviour of the environment in isolation has been studied for some time. For instance Musa [14] has proposed guidance for producing *user operational profiles*, a specific type of system environment where each operation offered by the system is associated a probability of being called by the user (i.e. the environment). The operational profile can be used to annotate an automata model that captures the composite (non-probabilistic) behaviour of the system and environment. To be able to account for more complex profiles in which the history of execution affects the probabilistic behaviour of the environment, others [15, 8] have proposed using a sample space of existing or similar system runs as the source of probabilistic behaviour and using an algorithm that summarises and annotates an (non-probabilistic) automata composite model of the system and environment with probabilities.

Although operationally intuitive, annotation approaches lack a declarative characterisation of the resulting annotated composite model and its relation to the source of probabilities. In addition, and more importantly they lack a notion

ESEC-FSE'09, August 24–28, 2009, Amsterdam, The Netherlands.

of property preservation. This is crucial as, whatever the source of probabilities for the annotation is, this source must have been validated according to some criteria. If the annotation algorithm does not preserve such criteria, then little can be said about the validity of analysis performed on the probabilistically annotated composite model.

Producing an automata-based probabilistic model of the environment alone before reasoning about its influence on the system has two important advantages. First, that automated techniques can be used to validate the model. This is a crucial step as an invalid model of the environment invalidates any quantitative reasoning about the system. Second, that the construction of the composite model system/environment can then be constructed compositionally.

Although many variations of probabilistic automata (e.g. [20, 3, 19, 5]) and composition operators (see [17] for a survey) have been proposed, none of these have been originally envisioned for reasoning quantitatively about system behaviour through composition of a probabilistic model of the environment and a non-probabilistic model of the software. Hence, existing formalisms are inadequate for said task. In particular, they lack an appropriate treatment of the notion of action controllability which leads to a number of problems including unclear semantics of the probabilities of the environment model, unintuitive probabilities in the composite model, and lack of preservation of environment probabilistic properties.

In this paper we propose a novel formalism for reasoning quantitatively about the behaviour of systems which combines and adds to notions taken from Input-Output Probabilistic Automata [20] and Interface Automata [7]. By doing so we address the limitations existing formalisms have when used to compose probabilistic behaviour of the environment with the non-deterministic behaviour of the software model. More generally, the formalism supports constructing models compositionally through automata composition, it supports validating the probabilistic behaviour of component models and provides guarantees on the preservation of this behaviour in the composition.

The remainder of the paper is organised as follows: in section 2 we use an example to motivate our approach, comparing to existing ones. In section 3 we present some building blocks for our work, while in section 4 we present our novel approach to the problem. In section 4.2 we expand on our approach and show that composition over this new formalism is correct in terms of the semantics intended. Along the paper, we illustrate our ideas with a motivating example and expand the ideas on a case study in section 5. Finally, we discuss the relation of our work with previous efforts in section 6 and offer our conclusions and prospects of future work on the subject.

2. MOTIVATION

In this section we present a simple example to motivate the problem of quantitative analysis of non-probabilistic models using probabilistic models of the system environment. We also briefly highlight the main issues related to modelling the probabilistic environments and of reasoning about the composite probabilistic system.

Let's assume we set out to model and analyse properties over a coffee machine system. To this end, we develop the model for the machine presented in Figure 1. This coffee machine prepares, at the user's request, either an espresso

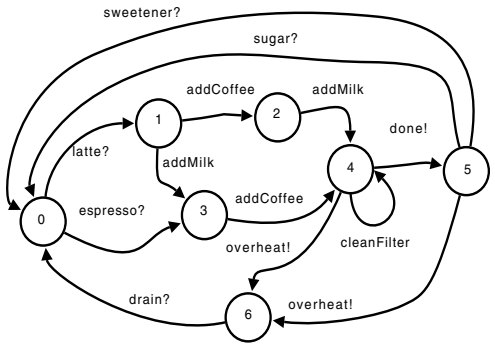

Figure 1: A simple coffee machine.

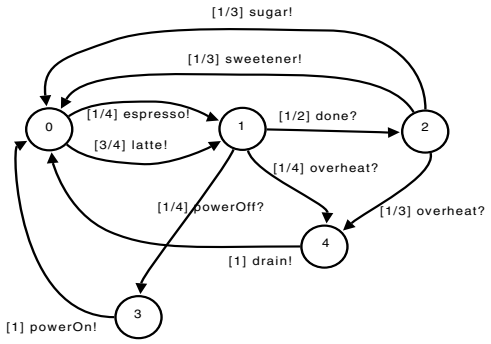

Figure 2: A (potential) generative environment.

(pure coffee) or a latte (coffee and milk). Once the user chooses her selection, the machine, in a way unbeknown to the user, prepares the beverage. Then, the machine informs the user it has finished the preparation, prompting for the addition of sugar or sweetener, finally delivering the drink. However, this coffee machine is known to sometimes overheat, requring manual drainage.

We would like to know, for example, whether the machine can overheat *after* it has filled the coffee cup (that is, after it reports the `done` message), as at this point, if the machine is overheated, the coffee may boil and spill violently towards the user, posing a safety hazard. The size of this model makes this question trivial, and by observing the trace that traverses states 0, 3, 4, 5, 6, we see that such an error is clearly possible. A larger model would require the use of a model-checker. Rather than attempting to fix the problem (this in general may not be possible or desirable), quantifying the ocurrence of this error based on the expected behaviour of the environment interacting with the coffee machine is of interest.

In order to do this, we first model a probabilistic environment via a *generative* model [3] (where the sum of probabilities of outgoing transitions on a state is 1), as seen in Figure 2 and try to study its relationship with the system model. We then can compose this environment model with the system model to reason about their correlation. Though at first this may seem plausible, in fact it exhibits some interesting problems, which make a sensible composition difficult to define:

- The *generative* paradigm forces the modeller to place probabilities on all transitions. This means that, for example, in state 1 we must assign probabilities to

events which are not controlled by the environment: `done?`, `overheat?`, and `powerOff?`. What do these probabilities mean if the environment has no say on the probabilistic occurrence of these? Even if the failure ratio of `overheat` was known, it would pertain to the system's specification rather than the environment's.

- State 1 also provisions for an autonomous coffee machine shutdown. However, analysing the composition of the environment with the system, it is simple to observe that the environment cannot leave state 1 until the system has reached state 4. However, state 4 cannot perform a `powerOff!` action. Hence, what does the probability of the `powerOff?` transition in the environment model mean? Clearly it cannot mean that in 25% of the cases, the environment will observe a power off event, as this will not be true in the composite model. In addition, now that `powerOff` cannot occur, how should the probabilities of `done` and `overheat` be computed in the composite model? Should, for example, the 1/4 be distributed equally leading to 5/8 and 3/8 for `done` and `overheat` respectively? Or should it be distributed based on the relative weights of the other actions: as `done` is twice as probable of occurring than `overheat`, `done` gets another 2/12 while `overheat` only gets 1/12? Clearly any decision here is somewhat arbitrary and difficult to justify in terms of the real world, hence questioning the validity of any posterior analysis on the composite model.
- State 2 also assigns probabilities to a mix of events, one controlled by the coffee machine (`overheat?`) while others controlled by the environment (`sugar!` and `sweetener!`) within the same probabilistic distribution. This distribution is not only encoding the probability of the machine overheating once the drink is done and the relative weights of the user choosing sugar over sweetener, but it is also compounding implicitly the race between the environment and the user.

The problems described above can be explained technically in terms of a lack that generative models have in modelling non-determinism and with composition. Not allowing non-determinism means that these models are at a loss when it comes to model external actions the environment must act in response to. The problems with composition in generative models have been studied previously [5].

Alternatively, the environment can be modelled under the *reactive* paradigm [19] (each action on each state has a probabilistic distribution that defines the next state). Figure 3 shows such an attempt. Reactive models, contrary to their generative counterpart, do allow for non-determinism, but do not allow probabilistic choice between different actions. There is a workaround for this, however, using internal actions. For instance, in our model the `choice` transition from state 4 is being used to introduce the probability of the environment choosing between sugar, sweetener or nothing.

The use of a reactive probabilistic model solves many of the issues of the generative paradigm. However, in general, it is too unconstrained and can lead to unsatisfactory results when composing a probabilistic environment with a non-probabilistic system model. Recall the property that the machine does not overheat once it has reported the coffee is `done`. This property does not hold for the system, and now the composite model system/environment can be constructed and the likelihood of such a violation can be

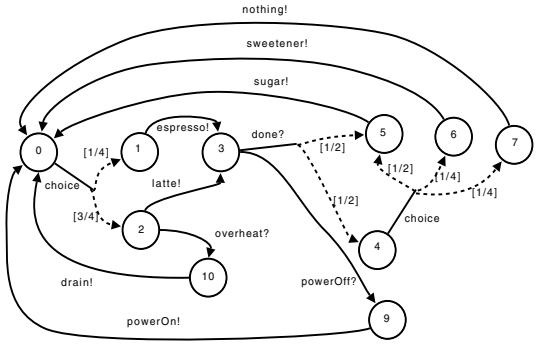

Figure 3: A (potential) reactive environment.

computed. Oddly, the result obtained using standard composition [19] and analysis [10] is zero which means the error cannot happen under this environment. However, reviewing the coffee machine model in isolation it is clear that there are no choices which the environment can make to avoid reaching the error. The reason for such an unintuitive result is that the environment constrains the occurrence of the system transition from state 2 to 4 that represents the error. When the system is in state 4, the environment must be in states 4, 5, 6, or 7, none of which admit `overheat`.

The result in the previous analysis is quite unintuitive (a property that does not hold in the system, that does not depend on the environment to hold and that when composed with an environment it does hold) and indicates that something is wrong with the way we have modelled the environment, composed it with the system or computed the probability.

Another example of unintuitive analysis results is the following: analysing the environment in isolation, we see that once the coffee is reported done, the probability of the environment not adding sugar nor sweetener is 1/8 (1/2 for the transition from 3 to 4, times 1/4 for the transition from 4 to 7). However, when we compute the composition and query the same property, the probability is zero. This time, the problem is the inverse of the above. The system can constrain the occurrence of `nothing!` (state 7 to 0 in the environment) because in state 5 it does not offer such a transition. Hence, a probabilistic property of the environment is not being preserved in the composition. This can be a serious problem: if such a property was used as part of the justification of the validity of the model of the environment (for instance, the property was validated against stakeholders or a sample space of measurements), and the composition does not preserve it, then the validity of the composite model is suspect.

A closer look at the reactive environment reveals other problems:

- State 2 and 4 block the system, as at these points the coffee machine may overheat. This introduces an artifical wait time on the system. Moreover, once the environment does advance, no further state allows for the processing of the overheating message, bringing the composite system collaboration to a deadlock.
- Analogously, state 7 is blocked by the system, as it can't bypass either the sugar or sweetener input.
- State 3 also exhibits a problem, although a more subtle one. In this state, the environment only listens to sys-

tems' outputs (namely `done` and `powerOff`). However, it may be the case that none of those actions are ever generated by the system at this spot; `powerOff` because it does not exist in the system model, and `done` because the system may choose to `cleanFilter` indefinitely instead. In this case, the environment-system cooperation could be left in a livelock state (and if the coffee machine arrives at a state it can't advance itself, a deadlock).

Although these three problems do not relate strictly to the probabilistic annotations, rather, they relate to the input / output synchronization of the system and the environment, they do have an impact in terms of preservation of properties of the system (as in our first property) and preservation of the probabilities of properties of the environment (as in our second property).

Summarising, in order to model the probabilistic behaviour of the environment and compose it with a non-probabilistic behaviour model of the system to obtain meaningful quantitative results a formalism is needed that can *i*) treat both non-deterministic behaviour and probabilistic behaviour, *ii*) address notions of controllability and monitorability of action by the environment and system, *iii*) preserve probabilistic properties of the environment after composition and *iv*) preserve violations of system properties that cannot be averted by the environment.

In the following section we propose a formalism inspired on interface automata which distinguishes controlled and monitored actions and supports probabilistic and non-deterministic behaviour. We present a suitable composition operator for these automata and in Theorem 4.2 and its corollary we demonstrate the required results of property preservation.

3. BACKGROUND

In this section we present some building blocks for our work. We first recall the definition of interface automata:

Definition 3.1 (Interface Automata [7]). *An* interface automaton *is a tuple* $P =< S_P, s_P^0, A_P^I, A_P^O, A_P^H, R_P >$ *consisting of* S_P*, a set of states;* $s_P^0 \in S_P$ *a distinct initial state;* A_P^I, A_P^O, A_P^H*, mutually disjoint sets of* input, output *and* hidden *actions respectively; and* $R_P \subseteq S_P \times A_P \times S_P$*, the transition relation.*

We denote the set of all actions $A_P = A_P^I \cup A_P^O \cup A_P^H$. We also note $A_P^I(s)$, $A_P^O(s)$ and $A_P^H(s)$ for a state $s \in S_P$ to denote the subset of actions in each A_P^I, A_P^O and A_P^H, respectively, that are *enabled* at said state s. An action $a \in A_P$ is said to be enabled at state $s \in S_P$ if there exists $t \in S_P$ such that $(s, a, t) \in R_P$. Analogously, we denote $A_P(s)$ the subset of actions enabled at state s, regardless their status as input, output or hidden.

In essence, an interface automaton is an LTS [12] where its action set has been further subdivided to distinguish the input, output and hidden actions[1]. As we will see, this does not make a syntactic difference, but it does semantically.

This action segregation allows for the notion of *composability* of interface automata:

[1]We reduce the original set of initial states to a single one without loss of generality

Definition 3.2 (Composability [7]). *Let* P *and* Q *be two interface automata; we say* P *and* Q *are* composable *if it holds simultaneously that* $A_P^H \cap A_Q = \emptyset$*,* $A_P \cap A_Q^H = \emptyset$*,* $A_P^I \cap A_Q^I = \emptyset$*, and* $A_P^O \cap A_Q^O = \emptyset$*.*

Furthermore, when referring to the interaction of two interface automata P and Q, it is usual to allude to its shared set of actions, $Shared(P, Q) = A_P \cap A_Q$. Note that if P and Q are composable, then $Shared(P, Q) = (A_P^I \cap A_Q^O) \cup (A_P^O \cap A_Q^I)$. We recall the definition of interface automata product and illegal states.

Definition 3.3 (Product [7]). *Let* P *and* Q *be two composable interface automata. Their product* $P \otimes Q$ *is defined by states* $S_{P\otimes Q} = S_P \times S_Q$*; initial state* $s_{P\otimes Q}^0 = (s_P^0, s_Q^0)$*; action sets* $A_{P\otimes Q}^I = (A_P^I \cup A_Q^I) \setminus Shared(P, Q)$*;* $A_{P\otimes Q}^O = (A_P^O \cup A_Q^O) \setminus Shared(P, Q)$ *and* $A_{P\otimes Q}^H = A_P^H \cup A_Q^H \cup Shared(P, Q)$*. Its transition relation* $R_{P\otimes Q}$ *is defined by*

$$R_{P\otimes Q} = \left\{ \begin{array}{l} \{((s,t), a, (s',t)) \,|\, (s, a, s') \in R_P \wedge \\ \quad t \in S_Q \wedge a \notin Shared(P, Q)\} \cup \\ \{((s,t), a, (s,t')) \,|\, (t, a, t') \in R_Q \wedge \\ \quad s \in S_P \wedge a \notin Shared(P, Q)\} \cup \\ \{((s,t), a, (s',t')) \,|\, a \in Shared(P, Q) \wedge \\ \quad (s, a, s') \in R_P \wedge (t, a, t') \in R_Q\} \end{array} \right\}$$

Definition 3.4 (Illegal states [7]). *Given two composable interface automata* P *and* Q*, their product's illegal states are defined by the set* $Illegal(P, Q) \subseteq S_P \times S_Q$*. For any* $s \in S_P$*,* $q \in S_Q$*,* $(s, q) \in Illegal(P, Q)$ *if* $\exists a \in Shared(P, Q)$ *such that* $a \in A_P^O(s) \wedge a \notin A_Q^I(q)$*, or conversely* $\exists a \in Shared(P, Q)$ *such that* $a \in A_P^I(s) \wedge a \notin A_Q^O(q)$*.*

Informally, the idea behind illegal states is that, for a composition to be legal, component systems should not block each other's enabled *output* actions.

The notions of composability and illegal states make it possible to define what a *valid environment* for a given interface automaton is.

Definition 3.5 (Valid environment [7]). *Given interface automaton* P*, another nonempty interface automaton* Q *is a* valid environment *for* P *if all the following hold: i)* P *and* Q *are composable.; ii)* $A_Q^I = A_P^O$*; and iii) no state in* $Illegal(P, Q)$ *is reachable in* $P \otimes Q$*.*

The previously presented definitions push us halfway through our goal of providing a suitable language for the specification of *probabilistic* user environments. The probabilistic semantics are introduced via a well-known reactive probabilistic formalism, that of (non-deterministic) Markov Decision Processes (MDP) [18, 6]. As we will see, this model extends classic LTSs by modifying the transitions so that they no longer reach a single state, but a probabilistic distribution over a set of destination states instead.

Definition 3.6 (Markov Decision Process (MDP)). *A* Markov Decision Process *is defined by a tuple* $M =< S_M, s_M^0, A_M, R_M >$ *where* S_M *is a set of states;* $s_M^0 \in S_M$ *is a distinct initial state;* A_M *is a set of actions; and* $R_M \subseteq S_M \times A_M \times D(S_M)$ *is the transition relation, where* $D(S_M)$ *denotes the set of* probabilistic distributions *over the set of states* S_M*.*

For convenience, we assume that for all states $s \in S_M$, the transition relation is such that $R_M(s) \neq \emptyset$ [6]. As additional notation, for any $s, s' \in S_M$, $a \in A_M$, $0 \leq p \leq 1$, we may note $s \xrightarrow{a,p}_M s'$ as a shorthand for $\exists \mu \in R_M(s,a)$ such that $\mu(s') = p > 0$. Also, if $p = 1$ we may note $s \xrightarrow{a}_M s'$.

In order to express and analyse properties over these probabilistic models, these automata are coupled with modal logics whose formulae express said properties. For the specific case of probabilistic models, the temporal logic pCTL* [1] has been introduced as an extension of the well known temporal logic CTL*. Essentially, pCTL* replaces path quantifiers present in CTL* for *probabilistic quantification bounds* on the related path formulae.

Informally, given a path formula ϕ, a typical pCTL* state formula takes the form of a classic CTL* state formula, but where path quantifiers have been replaced by the probabilistic operator $P_{\sim a}$. Thus, a state formula $P_{\leq a}\phi$ (resp. $P_{\geq a}\phi$), is true at a given state of the system if its possible evolutions from that state satisfy the formula ϕ with probability at most (resp. at least) a.

The definition of pCTL* satisfiability is very similar to CTL* satisfiability. Of course, the main differences reside in how to deal with the probabilistic formulae $P_{\sim k}\psi$. For in-depth discussion, the reader is referred to [1, 9], but the gist revolves on *measuring* the set of the *paths* satisfying ψ. The formula is then satisfied if and only if this measure is $\sim k$. Naturally, the measure in question is closely related to the probabilities associated to each of the transitions in the paths. For the scope of this paper, we'll give an informal notion of pCTL* satisfiability to which we'll refer repeteadly for the remainder of the paper. It is worth noting that MDPs are intrinsically non-deterministic. Therefore, it is in general impossible to assign a single probability value for a given event (a formula). In this sense, it is most usual to calculate upper and lower *bounds* for the probability with which a formula is satisfied by the MDP. We say an MDP satisfies a given formula within a *lower bound* (conversely, *upper bound*) probability, if when employing *any scheduler* to resolve non-determinism, the measure of the paths satisfying the formula is *higher* (conversely, *lower*) than the bound. We define a *scheduler* as follows:

Definition 3.7 (Scheduler). *A* scheduler *for an MDP M is a total function $A : frag^*(M) \rightarrow A_M \times D(S_M)$ such that if $A(\alpha) = (a, \mu)$ then it must be that $(a, \mu) \in R_M(last(\alpha))$.*

In the previous definition, $frag^*(M)$ denotes the set of *finite execution fragments* of M, that is, those sequences of the form $s_0 a_0 p_0 s_1 a_1 p_1 \ldots s_k$ of alternating states, actions and probabilities, ending with a state, such that if the subsequence $s_i a_i p_i s_{i+1}$ appears in the sequence, then there exists a distribution $\mu \in R_M(s_i, a_i)$ such that $\mu(s_{i+1}) = p_i$. $last(\alpha)$ denotes the state with which the sequence terminates. Note that resolving non-determinism over an MDP M via a scheduler A yields a fully probabilistic process, where probability for an event can be determined exactly.

4. PROBABILISTIC MODELLING

In this section we present our new modelling formalism designed to overcome the difficulties presented. For the remainder of this paper, we'll make use of various concepts related to measure and probability theory when referring to the probabilistic characteristics of the language. Although various concepts will be summarily introduced, the interested reader is referred to [9] for in-depth discussion. We will also introduce some other concepts relating to probabilistic automata theory as the need arises.

4.1 Probabilistic interface automata

Leveraging on the previous definitions, we can attain our aim of merging the notion of MDPs with that of interface automata. As a way to attain this objective, we define *probabilistic interface automata* based on MDPs.

Definition 4.1 (Probabilistic interface automata). *A* probabilistic interface automaton *is a tuple of the form $M =< S_M, s^0_M, A^I_M, A^O_M, A^H_M, R_M >$ where the sets A^I_M, A^O_M and A^H_M are mutually disjoint, and such that defining $A_M = A^I_M \cup A^O_M \cup A^H_M$ yields a Markov Decision Process $M_{MDP} =< S_M, s^0_M, A_M, R_M >$.*

Therefore, a probabilistic interface automaton is an MDP that shares the input, output and hidden action semantics from interface automata. Note that since a probabilistic interface automaton must induce an MDP, then $R_M \subseteq S_M \times A_M \times D(S_M)$. Note also that a probabilistic interface automaton A has an *underlying interface automata*, noted $A\downarrow$ and defined as follows:

Definition 4.2 (Underlying IA). *Given a probabilistic interface automaton E, we define its* underlying interface automaton *as the classic interface automaton $E\downarrow =< S_{E\downarrow}, s^0_{E\downarrow}, A_{E\downarrow}, R_{E\downarrow} >$ such that $S_{E\downarrow} = S_E$, $s^0_{E\downarrow} = s^0_E$, $A_{E\downarrow} = A_E$ and for all $s, s' \in S_{E\downarrow}$, $a \in A_{E\downarrow}$, $(s, a, s') \in R_{E\downarrow}$ if and only if there exists a distribution $\mu \in R_E(s, a)$ such that $\mu(s') > 0$.*

Simply put, the underlying interface automaton of a probabilistic interface automaton is a non-deterministic automaton with the same state and edge structure, where all probabilities have been *forgotten* and replaced by non-deterministic transitions, leaving all other information unchanged. Conversely, it is also worth noting that a classic interface automata can be embedded in a probabilistic interface automata by restricting R_M to Dirac distributions.

Every notion already defined for interface automata can thus be extended for probabilistic interface automata. Composability is defined in exactly the same way, as probabilities have no bearing in the definition. Probabilistic interface automata product, however, does express some differences regarding the composition of the transition relation:

Definition 4.3 (Product). *Given P and Q two composable probabilistic interface automata, their product $P \otimes Q$ is defined by the probabilistic interface automata:*

$$P \otimes Q =< S_{P\otimes Q}, s^0_{P\otimes Q}, A^I_{P\otimes Q}, A^O_{P\otimes Q}, A^H_{P\otimes Q}, R_{P\otimes Q} >$$

where $S_{P\otimes Q}$,$s^0_{P\otimes Q}$, $A^I_{P\otimes Q}$, $A^O_{P\otimes Q}$ and $A^H_{P\otimes Q}$ are defined in the same way as interface automata composition, and its transition relation $R_{P\otimes Q} \subseteq S_{P\otimes Q} \times A_{P\otimes Q} \times D(S_{P\otimes Q})$ is constructed in such a way that distributions governed by internal actions are translated directly and synchronizing actions are translated by way of pairwise distribution product as defined by MDP product [18].

Refer to Figure 4.1 for an example of two-state composition, where $a?$ denotes a is an input action for the automaton, and

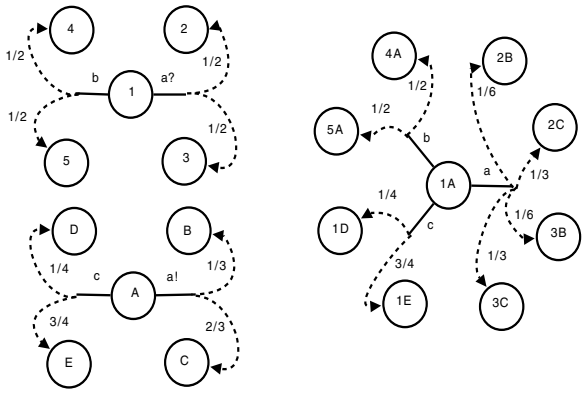

Figure 4: Probabilistic interface automata product (states 1 and A)

$a!$ denotes it is an output. Unannotated actions are internal. Recall that we would like the notion of probabilistic interface automata to exceed a syntactic notion and actually have an interesting semantic bearing, as otherwise its usefulness would be drastically reduced. We will see to this objective in theorems 4.1 and 4.2.

Interestingly enough, probabilities do not interfere with the interfacing notions of classic interface automata. This fact is reflected on the following theorem.

THEOREM 4.1 (PRODUCT INVARIANCE). *Given two probabilistic interface automata P and Q, their product is invariant on their probabilities, that is, $(P \otimes Q) \downarrow = (P \downarrow) \otimes (Q \downarrow)$*

The complete proof, which we omit for space reasons, is based on construction and definition of both the product operator and the downgrade operator $\downarrow$.

The notions of illegal states and valid environments can also be extended for probabilistic interface automata. In essence, they share the same definition, except for an important difference in the illegal states concept.

DEFINITION 4.4 (ILLEGAL STATES). *Given two composable probabilistic interface automata P and Q, their product's illegal states are defined by the set $Illegal_{ProbIA}(P,Q) \subseteq S_P \times S_Q$. For any $s \in S_P$, $q \in S_Q$, $(s,q) \in Illegal_{ProbIA}(P,Q)$ if any of the following is true: i) $(s,q) \in Illegal(P \downarrow, Q \downarrow)$; or ii) s (resp. q) belongs to a loop α in P (resp. Q) such that any action $a \in \alpha$ verifies $a \in A_P \setminus Shared(P,Q)$ (resp. $a \in A_Q \setminus Shared(P,Q)$).*

Condition *ii*) makes explicit a liveness condition underlying the semantics of MDPs: schedulers as total functions mean that at least one transition is expected to be taken to leave any state. A purely internal loop in one process of the product would imply a product-level scheduler in which the other processes involved would be stuck in one state forever. As an example, recall the coffee machine of Figure 1, and the reactive environment of Figure 3. We have previously stated that state 3 was modelled in such a way that could induce a livelock if the coffee machine cleaned it filters indefinitely while the environment was at that state. We will see these scheduler issues and condition *ii*) are central to the validity of Theorem 4.2 and its corollary.

4.2 Property preservation

Armed with the aforementioned concepts, a probabilistic environment's behaviour, when said environment is modelled via a probabilistic automata such as probabilistic interface automata, can be observed by expressing interesting properties in appropriate logics. In the case of probabilistic interface automata, pCTL* is a viable logic, since we can leverage on their underlying MDP structure and the scheduler definition (recall Definition 3.7). To this end, we need to convey the notion that the product of the environment and the system model is not merely a syntactic convenience, but that it does maintain a semantic relationship between the models and its properties. The following theorem and its corollary see to this objective.

THEOREM 4.2 (PRESERVATION OF CONES). *Let M be an interface automaton representing a system under analysis and a probabilistic interface automaton E modelling a valid environment for system M. Let c be the cone induced by finite fragment execution α in $E \otimes M$. Then, noting the cone c_E as the cone induced by the restriction of α to the language of E, it holds that both c and c_E have the same measure.*

In the above theorem, *cones* are the infinite sets of execution fragments obtained by the extension of a finite execution fragment. It is a well known result that cones correlate to measure theory, in the sense that cones have measure. Furthermore, the measure of a given pCTL* formula can be expressed as a summation of cones. Informally, this theorem states that cones, and therefore pCTL* formulae measure are governed by the environment alone. This theorem and these notions, along with the composability conditions and valid environmental notions for probabilistic interface automata lead to the following corollary:

COROLLARY 4.1 (PROPERTY PRESERVATION). *Let M be an interface automaton representing a system model, and the probabilistic interface automaton E modelling a valid environment for system M. Then, for any formula $\phi \in pCTL^*_{A_E}$ closed under stuttering, it holds that $E \models \phi \Rightarrow E \otimes M \models \phi$.*

We restrict the logic to the set of actions A_E, which makes sense since interesting properties are those observed over the shared set of actions of the composite system under analysis (and in some cases internal environmental actions). Also, we restrict formulae to those closed under stuttering.

Informally, this theorem provides a validation for the compositional view of the environment-system model relation, as properties formulated early in the validation process do not lose their meaning once the environment is composed with the system model. Intuitively, this is true, since the composition does not add new behaviour and neither does it prohibit allowed behaviour by the environment. Noting the validity of the theorem in the case of the probabilistic formulae is a little more troublesome, but the main point relates to the fact that, although the composition does not forbid the environment to act as it pleases, it may restrict it somewhat. The formal proof of the theorems involve manipulation of schedulers in both $E \otimes M$ and E, suggested by the previously stated intuitive idea.

We omit the complete proof for space reasons. However, the insight of the proof lies in noticing that computing the

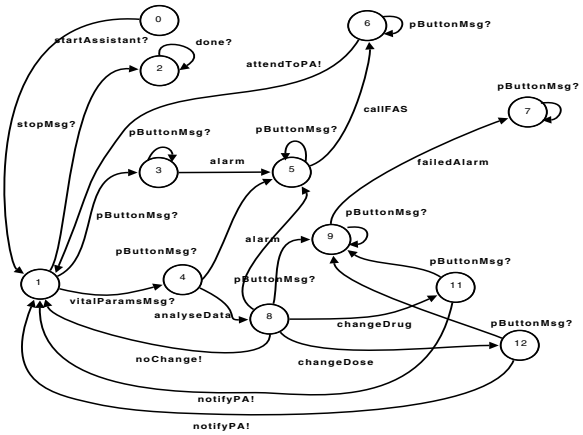

Figure 5: The TeleAssistance Software.

product of the environment with the system model essentially restricts the set of feasible environmental schedulers by pruning some behaviours. For example, refer to to the coffee machine example and its proposed reactive environment in Figures 1 and 3. Looking at the environment, a scheduler A for the reactive model, having witnessed the partial trace (0) `choice` (2) and now deciding to take the `overheat` transition to state 10 is a valid one for the environment. However, once this reactive model is composed with the coffee machine, any scheduler for the composite system that witnesses a partial trace that, when projected on the reactive model, equals the aforementioned trace, *cannot* take the `overheat` transition, as the system forbids this transition at those states. In this sense, scheduler A is deemed invalid by the product. However, at least one scheduler must remain valid, in order to determine the lower and upper bounds for the property probability.

In a most pathological case, one could argue that there could exist valid schedulers for the composition such that, once projected over the environment, become invalid for the environment itself. However, it can easily be shown that Definition 4.4 prevents this situation from happening: such a situation would render said environment illegal for the system under analysis. As a result of this observation, the correlation between the valid schedulers for the composition and environmental schedulers is guaranteed and thus the property preservation results presented in Theorem 4.2 and its corollary also also hold.

As an additional note, it is worth noting that composition, while preserving pCTL* properties, does not actually preserve the *exact* event probabilities for a given property. For example, assume environment E satisfies the property $P_{\leq 0.75}\psi$. Recalling the satisfiability definition, this means that E satisfies ψ with probability at most 0.75 under the control of *any* scheduler. There may, or may be not, an actual scheduler such that, when putting E uner its control, actually witness probability 0.75 for formula ψ. The interesting issue is that even if there is such a scheduler, the existence of a scheduler for $E \otimes S$ witnessing probability 0.75 for ψ is not guaranteed; in fact every scheduler for $E \otimes S$ may witness an *inferior* probability, without ever reaching the maximum witnessed in the environment by itself.

5. CASE STUDY

In order to validate our approach, we analyse quantitatively the behaviour of an existing software system by varying the expected probabilistic behaviour of the environment, described using a probabilistic interface automata, composing it with a non-deterministic model of the software, and producing bounds on the probability of system-specific properties holding.

5.1 The TeleAssistance Software

The software system we analyse is an extension of the case study presented in [8]: The *TeleAssistance* (TA) software, a web-based application providing remote assistance to patients. In its most basic interaction, the patient commences operation via a `startAssistance` command. This puts TA in an infinite loop accepting any of the following requests:

- `stopMsg`: the user may cancel TA service for now.
- `vitalParamsMsg`: the user may send varied body readings via a supplied device. The stats are analysed by the application server, which may then suggest a course of action: change the patient's medication via the `changeDrug` and `changeDose` commands. If a succesful adjustment is made, the patient is notified via the `notifyPA` message (with no details regarding the kind of adjustment made). If any anomalies are detected during the analysis, a First-Aid Squad (FAS) is requested and sent. In the case of a FAS being sent, the patient is informed via the `attendToPA` message.
- `pButtonMsg`: if feeling sick, the user may activate a panic button, triggering an alarm in the TA service. A succesful processing of the alarm results in a FAS being sent to the patient's home.

We have augmented the simplified model presented in [8] in two ways in order to introduce richer software-environment interactions: first by specifying that, for emergency reasons, the panic button may be pushed at any operational state of the software, even if waiting for other results; and second, by refining the feedback provided by the software so that the patient is told if no medication adjustment if needed. We depict an abstract model of the TA software in Figure 5, note that the model is as an interface automaton, which is a special case of the probabilistic interface automata introduced in this paper. As is customary, output actions are appended with '!', and input actions are appended with '?', while internal actions are left with no annotations.

The TA software exhibits a critical failure, reached by the triggering of the `failedAlarm` event, in which an alarm has been raised but the alarm failed to be acknowledged or properly handled, thus not calling and sending the FAS. In this implementation, such an error (state 7) is reached if the user presses the panic button once the software has started analysing vital parameters' data, an event sequence not properly foreseen by the implementation team. Relying on the software's model only, we can easily see that such a state is reachable. However, actual probability of reaching said state is highly dependent on the environment's behaviour. We now show how to model the probabilistic behaviour of the environment using probabilistic interface automata, and how such model and the theory presented in previous sections allows quantifying in a meaningful way the probability of critical failures based on the modelled probabilistic assumptions of the environment.

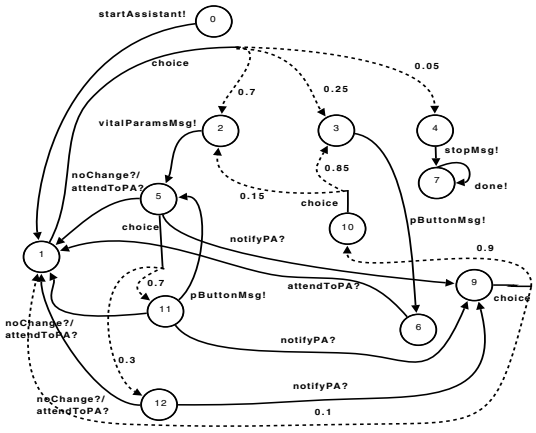

Figure 6: An initial environment for TA system

5.2 Modelling the Environment

In Figure 6 we depict a first attempt at modelling the probabilistic behaviour of the environment of the TA software. This environment, when waiting for a vital parameters analysis response, probabilistically chooses to wait patiently or press the panic button. Also, it reflects a certain degree of anxiety in the patient's behaviour, since it behaves quite differently depending on whether the software determines to adjust her medication or not. If the medication is not adjusted, the environment reverts to its usual behaviour, however, if the medication is indeed adjusted, the environment becomes more prone to pressing the panic button.

Although seemingly a reasonable model of this environment, this is not the case. It is straightforward to see that Figure 6 is a probabilistic interface automaton (see Def. 4.1) and that it is composable (see Definition 3.2) with TA system model (Figure 5). However, Figure 6 is not a valid environment (see Definition 3.5) for the TA system model as state 7 of the system and state 12 of the environment are an illegal pair (see Def. 4.4) that is reachable in the product (see Def. 4.3) of both models via the trace: $(1,1)$ `choice` $(1,2)$ `vitalParamsMsg` $(4,5)$ `analyseData` $(8,5)$ `choice` $(8,11)$ `pButtonMsg` $(9,5)$ `choice` $(9,12)$ `failedAlarm` $(7,12)$ where the first element of each pair is a state of the system and the second is a state of the environment.

The fact that $(7,12)$ is an illegal state highlights that the environment is making incorrect assumptions on the behaviour of the system and renders the probabilistic evironment behaviour modelled meaningless: for instance, analysing the behaviour of the probabilistic environment it is easy to conclude that the probability of sending a `vitalParamsMsg` to the system as the next message if being at state 12 is at most 0.7, and at least 0.205 (the upper bound is obtained if the `noChange/attendToPA` transition is followed, while the lower bound is the result of the sum of the possible outcomes of taking the `notifyPA` transition). However, the same analysis on the product results in a probability inconsistent with the analysis on the environment alone. The inconsistency is that while on the environment the lower bound for the property was 0.205, the lower bound was decreased to zero (rather than increased) when composed with the system. The increase of the lower bound is due to the fact that the environment's behaviour specified in state 12 is restricted when the system is in state 7, hence the contribution that outgoing transitions from 12 made to the lower bound of the property are no longer possible. However, this particular environment fails to makes a provision in modelling the possibility of such a restriction.

In summary, if the analyses performed to validate the probabilistic behaviour of the environment are not valid once the environment is composed with the software, then the model of the environment has a limited, if any, potential for sound analysis. The definition of legal environment, which the model in Figure 6 does not satisfy, is aimed to guarantee sound analysis.

Thus, a legal environment for the TA system needs to add to the previous version of the environment `timeout` transitions from states 6 and 12, modelling the environment can give up waiting for the software response concluding that it has probably crashed in some way. The property discussed with the initial probabilistic environment now evaluates to the interval $[0, 0.7]$ in this legal environment, and this is consistent with the evaluation of the property when composing the legal environment with the software. In fact, due to Theorem 4.2 and its corollary we know that any property that has been used to validate the probabilistic behaviour in this legal environment will be preserved in its composition with the software.

5.3 Quantitative Analysis of the TA Software System

Since we have a legal environment for the TA software, we can now analyse quantitatively the behaviour of the TA software system by checking the probability of system properties holding when the TA software is composed with the legal probabilistic environment. The properties we considered for our quantitative analysis were taken from [8]. To perform the analysis we use the model checker PRISM [10], a well-known probabilistic verification tool.

Consider the property that states that the FAS is always sent to the patient location when the alarm has been raised. Clearly, the TA software does not satisfy this property (see transition from 9 to 7). However, it is interesting to quantify the probability of such error under the assumption of a particular probabilistic behaviour of the environment. To do this we first computed the product of the interface automaton for the TA software and the probabilistic interface automaton modelling the environment and then using PRISM we quantify the occurrence of the error which can be characterised by the CTL formula $error = (\text{true}\,\mathcal{U}\,\text{TA.state} = 7)$. Recall that, because of non-determinism in the TA software, we will not obtain a single probability as the event measure, but rather an interval of where the probability lies.

In the product of the TA software and its legal environment, the lower and upper bounds for reaching $error = (\text{true}\,\mathcal{U}\,\text{TA.state} = 7)$ can be characterised as $P^{error_1}_{min} = \max_x\{P_{\geq x}(\text{true}\,\mathcal{U}\,\text{TA.state} = 7)\}$ and respectively $P^{error_1}_{max} = \min_x\{P_{\leq x}(\text{true}\,\mathcal{U}\,\text{TA.state} = 7)\}$. Computing these values in PRISM yields $P^{error_1}_{min} = 0$ and $P^{error_1}_{max} \sim 0.9108$. Note that the lower bound does not convey much information because of the highly non-deterministic nature of the TA software model: there is always a non-deterministic possibility that the error is *never* reached, in other words, there is a scheduler (a way of making the various non-deterministic choices in the model) that always avoids the error. If information were available on the probabilistic behaviour of the internal choices of the TA software (e.g. reliability in-

formation) then the lower bound for this property could be non-zero.

The (rather high) value of $P_{max}^{error_1}$ is sensitive to the probabilistic behaviour of the environment, namely the four probabilistic transitions labelled with `choice` (states 1, 5, 9 and 10). Varying the probabilities on these transitions helps understand the impact of the probabilistic environment on the system behaviour. Table 1 summarises a few analyses where these probabilities have varied and how these variations affect the final value of $P_{max}^{error_1}$. Note that from theorem 4.2, only these variations make an impact on the final value of the property.

The table shows that the value of $P_{max}^{error_1}$ decreases noticeably when the probability of exiting the assistance system (that is, transitioning from state 1 to 4 on `choice`) is increased (see row 5). This is sensible as the property being checked has an unbounded until operator, meaning that we are interested in the occurrence of errors no matter how long the system runs. Hence, the longer the system runs, the more likely it is to fail; and so the more likely the environment chooses to stop the software (transition 1 to 4) the less likely the probability of error, bringing $P_{max}^{error_1}$ down.

Another interesting analysis, taken from [8] is to understand the probability of the following scenario occurring: "If a `changeDrug` or `changeDose` occurs, the next message received by the TA generates an alarm which fails". Rather than using a pCTL* formula, we modelled this property by means of an observer interface automaton that flags this scenario as an error. Table 1 also summarises the results obtained for this error measure, $P_{max}^{error_2}$ for the same distributions as before, showing the relationship between distributions, and the differences with $P_{max}^{error_1}$.

Summarising, in this Section we have shown how probabilistic interface automata supports quantitative analysis of non-deterministic models. Crucial is the notion of legal environment (and related theorems) which constrains the acceptable models of the probabilistic behaviour of the environment to those that ensure that analysis performed to validate the environment's probabilistic behaviour is sound and preserved when analysing the composite system.

6. DISCUSSION AND RELATED WORK

In the last few decades, researchers have paid attention to the concept and consequences of operational profile in system reliability specification and analysis [2, 14].

Regarding enriching models with probabilistic information, we can mention the work in [15, 8] that, unlike our compositional work, yields a verification artefact that is a single model containing all the relevant probabilistic transition information, both pertaining to the environment and to the system. Also, Markov models such as these are purely probabilistic, which may not allow us to fully model concurrent systems' non-deterministic behaviour.

The non-deterministic-probabilistic dichotomy is not a new issue. For example, although *generative* models [3] do not directly allow non-determinism themselves, an asynchronous parallel composition (à la CSP [11]) induces such non-determinism and must be dealt with. Works such as [5, 16] advance in this direction resorting to redistributing probabilities when finding synchronising actions with no matching counterpart. It is unclear if this approach is suitable when the probabilities reflect system-environment interaction—the environment (in the most usual case, a user) may not actually redistribute probabilities on allowed actions when the desired one is not allowed. Regarding *reactive* models [19] the reader can refer to Sect.2 to understand limitations regarding our goals.

It must be noted that an important precedent to this work is that of probabilistic I/O automata [20]. This model enriches classic I/O automata [13] with probabilities, establishing a hybrid between the generative and reactive models, since output actions are modelled in a generative way while input actions are modelled reactively. The approach in itself is interesting, but the probabilistic I/O automata model has some characteristics we consider problematic. In first place, it inherits from I/O automata the notion of input enabledness, that is, every component automaton, at every state, must allow every possible input as a transition. As we previously argued, this is not a realistic restriction in most cases, since systems are usually designed with some concept of the environment in mind, and thus restricting some inputs at certain points of execution. Another characteristic aspect of probabilistic I/O automata is that they introduce a real-valued parameter to each state in each component automaton. This parameter, an additional random variable as it happens, models a *delay* on each automaton state. The rationale for this delay is the need to somehow resolve conflicting races, since at some points when composing it would be feasible for more than one component to synchronise its actions. This delay then establishes an order in which the automata advance, that is, the automata in which the state delay is the least will advance first. This notion of resolving races between competing transitions is also present in our model, as in other proposed models [16]; however it is represented by an external entity (the scheduler), in such a way that the scheduler has no dependence on the model itself, and the model behaves independently of the scheduler. We argue that the idea of building-in the scheduler as a composite aspect of the system model is undesirable, as we aim to a separation of concerns. Finally, a behaviour composability result is presented for probabilistic I/O automata, though it is different to the one we present in this paper. Probabilistic I/O automata behaviour preservation stems from that of the original non-probabilistic I/O automata. This results states that every execution trace in the composite automata, when restricted to the actions of each component automaton, is an execution trace of said component automaton. However, this result leverages heavily on the embedded scheduler concept depicted above. Our result does not establish such a stringent relation, since we establish that system-environment composition does refine the specified behaviour, but observed probabilistic behaviour in the environment is still preserved, thus allowing for early elicitation of interesting properties.

7. CONCLUSIONS

Quantitative model checking and analysis are promising techniques to complement Yes/No automatic analysis of behaviour. This work deals with some of the software engineering challenges that need to be solved to enable such a technology, namely, the incorporation of probabilities into system models lacking probabilistic information. We believe that the methodological approach to add and reason about validity of behavior measures should be based on the use of some external component modelling the system environment. This naturally raises several formal and practical

Run	R(1,choice)	R(5,choice)	R(9,choice)	R(10,choice)	$P_{max}^{error_1}$	$P_{max}^{error_2}$
1	$(2 \mapsto 0.7),(3 \mapsto 0.25),(4 \mapsto 0.05)$	$(12 \mapsto 0.3),(11 \mapsto 0.7)$	$(1 \mapsto 0.1),(10 \mapsto 0.9)$	$(2 \mapsto 0.15),(3 \mapsto 0.85)$	0.9108	0.1435
2	$(2 \mapsto 0.7),(3 \mapsto 0.25),(4 \mapsto 0.05)$	$(12 \mapsto 0.3),(11 \mapsto 0.7)$	$(1 \mapsto 0.1),(10 \mapsto 0.9)$	$(2 \mapsto 0.99),(3 \mapsto 0.01)$	0.9304	0.6727
3	$(2 \mapsto 0.7),(3 \mapsto 0.25),(4 \mapsto 0.05)$	$(12 \mapsto 0.3),(11 \mapsto 0.7)$	$(1 \mapsto 0.95),(10 \mapsto 0.05)$	$(2 \mapsto 0.15),(3 \mapsto 0.85)$	0.9076	0.4707
4	$(2 \mapsto 0.7),(3 \mapsto 0.25),(4 \mapsto 0.05)$	$(12 \mapsto 0.8),(11 \mapsto 0.2)$	$(1 \mapsto 0.1),(10 \mapsto 0.9)$	$(2 \mapsto 0.15),(3 \mapsto 0.85)$	0.7584	0.41
5	$(2 \mapsto 0.15),(3 \mapsto 0.2),(4 \mapsto 0.65)$	$(12 \mapsto 0.3),(11 \mapsto 0.7)$	$(1 \mapsto 0.1),(10 \mapsto 0.9)$	$(2 \mapsto 0.15),(3 \mapsto 0.85)$	0.1441	0.105
6	$(2 \mapsto 0.15),(3 \mapsto 0.2),(4 \mapsto 0.65)$	$(12 \mapsto 0.3),(11 \mapsto 0.7)$	$(1 \mapsto 0.95),(10 \mapsto 0.05)$	$(2 \mapsto 0.15),(3 \mapsto 0.85)$	0.1393	0.105
7	$(2 \mapsto 0.15),(3 \mapsto 0.2),(4 \mapsto 0.65)$	$(12 \mapsto 0.8),(11 \mapsto 0.2)$	$(1 \mapsto 0.95),(10 \mapsto 0.05)$	$(2 \mapsto 0.99),(3 \mapsto 0.01)$	0.0458	0.0384

Table 1: Varying $P_{max}^{error_1}$ and $P_{max}^{error_2}$ for different distributions

challenges regarding preservation of meaning of annotations in both of the existing and composed artifacts. The key to these challenges is a careful treatment of controllability of actions and non-determinism. We present probabilistic interface automata as a suitable formalism satisfying these requirements and show that the language is compositional, that is, there is a notion of property preservation between the components and the composite system. As a way to validate our claims, we present a case study along with our results obtained by the use of the technique. Although the case study presented leverages on manually generated environments, research on the generation of useful and sound environments is the focus of future and ongoing work.

It is worth noting that our current approach precludes from specifying probabilities over actions that are not controlled by the environment by considering their occurrence and ordering a matter of non-determinism (e.g. system-controlled actions, race conditions between system and environment-controlled actions, etc.). This is key to formal characterisation of the composite system in this paper and is reasonable in many settings including the motivation for this work. However, part of our research agenda includes sound ways to extend the probabilistic aspects supported by our approach without altering its theoretical and methodological basis. In particular, the current results of property preservation require one of the models to be non-deterministic. Although composition is well-defined between fully probabilistic Interface Automata, results analogous to theorems 4.1 and 4.2 may require stronger conditions for fully probabilistic IA composability. Further work is also expected on supporting fairness conditions, in order to weaken conditions for legality of product states.

8. ACKNOWLEDGEMENTS

This work was partially supported by CONICET and grants UBACyT X021, ANPCyT 2005 32440 and CONICET PIP112-200801-00955KA4P.

9. REFERENCES

[1] A. Aziz, V. Singhal, F. Balarin, R. Brayton, and A. Sangiovanni-Vincentelli. It Usually Works: The Temporal Logic of Stochastic Systems. *Lecture Notes in Computer Science*, pages 155–155, 1995.

[2] R. C. Cheung. A user-oriented software reliability model. *IEEE Trans. Softw. Eng.*, 6(2):118–125, 1980.

[3] I. Christoff. Testing equivalences and fully abstract models for probabilistic processes. In *Proceedings of CONCUR*, volume 90, pages 126–140. Springer, 1990.

[4] E. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, 1999.

[5] P. D'Argenio, H. Hermanns, and J. Katoen. On Generative Parallel Composition. *Electronic Notes in Theoretical Computer Science*, 22:30–54, 1999.

[6] L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.

[7] L. de Alfaro and T. A. Henzinger. Interface automata. In *ESEC/FSE-9: Proceedings of the 8th European software engineering conference and 9th international symposium on Foundations of software engineering*, pages 109–120. ACM, 2001.

[8] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time adaptation. In *ICSE '09: Proceedings of the International Conference on Software Engineering*. IEEE Computer Society, 2009.

[9] P. R. Halmos. *Measure Theory*. Springer-Verlag, 1950.

[10] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920, pages 441–444. Springer, 2006.

[11] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, August 1978.

[12] R. M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, 1976.

[13] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 137–151, New York, NY, USA, 1987. ACM.

[14] J. D. Musa. Operational profiles in software-reliability engineering. *IEEE Softw.*, 10(2):14–32, 1993.

[15] R. Roshandel and N. Medvidovic. Toward Architecture Based Reliability Estimation. In *Proc. Twin Workshops on Architecting Dependable Systems*, 2004.

[16] R. Segala and N. Lynch. Probabilistic Simulations for Probabilistic Processes. In *CONCUR'94, Concurrency Theory: 5th International Conference*. Springer, 1994.

[17] A. Sokolova and E. P. de Vink. Probabilistic automata: System types, parallel composition and comparison. In *LNCS Validation of Stochastic Systems*, volume 2925, pages 1–43, August 2004.

[18] M. Stoelinga. *Alea jacta est: verification of probabilistic, real-time and parametric systems*. PhD thesis, University of Nijmegen, The Netherlands, 2002.

[19] R. van Glabbeek, S. Smolka, B. Steffen, C. Tofts, and A. CWI. Reactive, generative, and stratified models of probabilistic processes. In *Logic in Computer Science, 1990. LICS'90 Proceedings.*, pages 130–141, 1990.

[20] S. Wu, S. Smolka, and E. Stark. Composition and behaviors of probabilistic I/O automata. *Theoretical Computer Science*, 176(1-2):1–38, 1997.