My Model Checker Died! How Well Did It Do?

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. However, model checking techniques suffer from the traditionally called *state explosion* problem, that is, models which are useful for analysis grow exponentially in size when verifying their concurrent behaviour. This state explosion problem is a serious limitation of model checking techniques, often making the application of tools that apply them infeasible, and limiting techniques to only a partial exploration of the complete state space. In this work we propose a novel approach that could help gather useful, quantified domain-related information from such incomplete explorations, leveraging on the concept of probabilistic behaviour models of the environment, for supporting dependability cases.

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

Automated techniques that explore automata-based models in order to gain increased assurance regarding the absence of errors have been investigated for some time, with promising accounts of success. A notable example is model checking [2], 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 or not.

Model checking, however, suffers from the state explosion problem: as the system-under-analysis descriptions grow more complex, the state space to be explored is usually too large and the model checker exhausts its resources (physical memory and/or alloted running time) before a positive or negative result can be provided. In these cases model checkers provide no feedback at all, which is not only frustrating and a waste of resources but also a lost opportunity, as the user does not get any information regarding the portion of the state space the model checker was able to explore.

In this position paper we put forward an idea that aims to provide useful feedback on an incomplete model exploration. In particular, we aim to quantify in a meaningful way the degree to which the exploration of the model has covered its entire state space. Such a quantification would serve as relevant information when constructing a dependability case [8] despite the fact that a conclusive model exploration is not feasible.

What would a meaningful measure of a partial exploration be? Reliability claims are probabilistic in nature and usually are relative to operational profiles of software [10]. In this position paper, we propose measuring a partial model exploration as a probability that captures the likelihood of a system's execution evolving out into states that have not been previously checked in the partial model exploration. This probability, which would be computed from a probabilistic model of the environment that interacts with the system under analysis, represents an upper bound of the probability of a specific property being violated in the system under analysis, assuming such a violation was not found in the initial, incomplete, exploration performed during the model verification. Such a measure is akin to the reliability measures studied in the software reliability community where there is interest in what the likelihood of a failure being encountered is. One central idea we propose in this position paper is that understanding the extent to which a partial exploration is relevant for a dependability case can be achieved by setting and varying the operational profile against which the measure of the partial exploration is computed. However, in contrast to software reliability approaches based on testing, we aim to exploit the rigourous and extensive explorations that model checking tools and techniques are capable of, thanks to their fine-grained control of the model exploration strategy and efficient techniques for identifying already visited states.

QUOVADIS '10, May 3 2010, Cape Town, South Africa

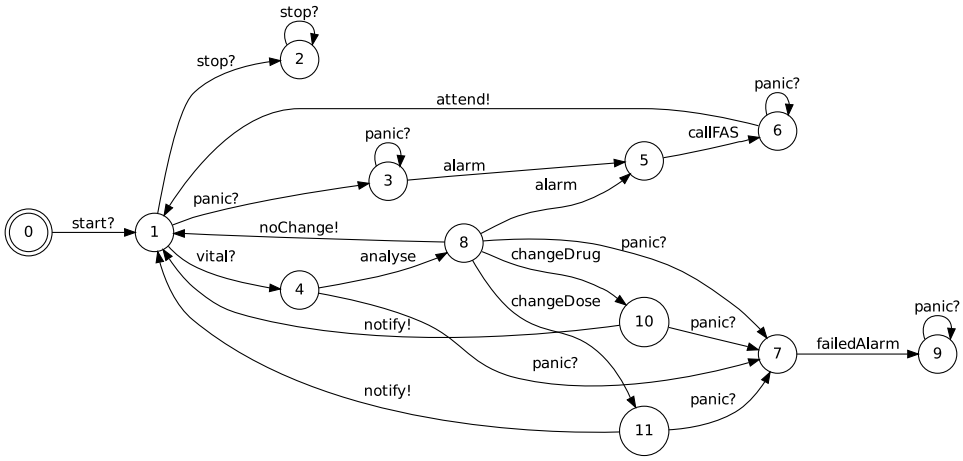

Figure 1: The TeleAssistance Software model (SUA).

Summarising, in this paper we put forward two novel ideas: extracting useful information from inconclusive exhaustive model checking analyses, and quantifying partial state explorations using the system environment's probabilistic behaviour profile. We show how our ideas can be built on top of our previous results on composition of probabilistic environment behaviour models with non-probabilistic system models [15] and we illustrate our ideas with a running example.

2. OVERVIEW

In this section we give an overview of the approach we propose. We introduce a running example (an extension of the case study presented in [4]) and show some of the results that could be obtained using the approach.

Consider a web-based system designed to remotely analyse vital information about a patient, provide feedback related to the patient's medication, and also provide assistance to the patient in case of emergencies or if explicitly required by the patient. Figure 1 depicts the behaviour of the system, which we'll refer to as the TeleAssistance (TA) system. In its most basic interaction, the patient commences operation via a `start` command. This puts the TeleAssistance system in an infinite loop accepting any of the following requests:

- `stop`: the patient may cancel TeleAssistance service for now.
- `vital`: the patient 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, or perform no adjustments, informing the patient through the `noChange` message. If a successful adjustment is made, the patient is notified via the `notify` message (with no details regarding the kind of adjustment made). If any anomalies are detected during the analysis, an alarm is raised, which results in a First-Aid Squad (FAS) being requested and sent. In the case of a FAS being sent, the patient is informed via the `attend` message.
- `panic`: if feeling sick, the patient may activate a panic button, triggering an alarm in the TeleAssistance service. A successful processing of the alarm results in a FAS being sent to the patient's home, and the `attend` message being issued as well.

A critical property the system is expected to satisfy is that every time an alarm is raised, the First-Aid Squad must eventually be dispatched to the patient's home, and must be dispatched before the patient is able to send new vital information messages. Since the model is small enough, it is easy to see that this scenario might be violated by the TA software if it evolves to state 7. Such an execution is possible, for example, if the patient chooses to press the panic button during the processing of the vital parameters sent for analysis. In other words, in order to avoid the critical error, once `vital` has occurred, `panic` must not occur until either `noChange`, `notify`, or `callFAS` happen. Note that the actual occurrence of such a critical failure depends not only on the software's behaviour but also on the patient's behaviour.

The behaviour model of the patient, given in the nondeterministic model of Figure 2 (ignore the numbers in brackets for now) shows how the patient interacts with the TA system and how it reacts to the events it receives from the system as a response to inputs. In the case that the patient sends vital statistics, he or she waits for a response from the TA system (although the patient may become anxious at any given time and press the `panic` button while waiting for this response).

It is easy to see, since the model is small enough, that the critical property described above can be violated as a result of the combined behaviour of the software and patient: the failure scenario described falls within possible behaviours of the patient. Let us assume however, in the interest of the argument, that we cannot explore the model manually and that an (extremely limited) model checker attempts to check the composition of the software and patient models against the property, but fails due to lack of resources.

Given the above assumption, we would be left with no feedback whatsoever on the relation between the models and the property. We would only know that a property violation has not been found up to the moment resources ran out. But how much of the composite system was found to be free of violations? And how confident can we be on the nonexistence of a property violation? We now show how these questions could be answered to some extent by quantifying

the partially explored state space using a probabilistic profile of the system's environment.

Let us assume that the state space explored by the model checker before running out of memory is that shown in Figure 3(a). The states in the squared section have been fully explored and found to be exent of error. Note that the user of the model checker would not be allowed to see such partial state space as it is typically too big; and also note that error states are not present in the figure, thus these error states are not only outside of the explored space, but from the model checker and the user's perspective it is unknown if a property violation exists.

Let us now assume that a probabilistic model of the patient's behaviour is available (see Figure 2 and now considering bracketed values too). This model describes, for instance, the likelihood of the patient choosing between options `stop`, `vital` and `panic` (in state 1). It also shows that if the patient sends his or her vital statistics there is a 0.3 chance that the patient may become anxious and press the `panic` button (transition from 5 to 11) and that if the TeleAssistance system informs the patient of an imminent change in the prescribed drug or dosage (`notify`), then the patient is much more likely to press the `panic` button (transitions from state 9 to 10 and 10 to 3) in the next interaction. Note that the probabilistic model in Figure 2 only extends the previous non-deterministic behaviour model of the patient with probabilistic transitions and that it does not change its non-probabilistic behaviour (*i.e.* if the probabilities of Figure 2 are ignored we obtain a model equivalent to the one that was used in the initial verification effort).

If a partial exploration similar to that of Figure 3(a) were generated using the probabilistic model of the environment of Figure 2 rather than the non-probabilistic one, the probabilistically annotated partial exploration of Figure 3(b) is obtained. From this model, a measure of the partial exploration which is relevant in terms of the expected use of the software can be calculated.

A first, naïve, question that could be asked is what is the probability of the system exhibiting behaviour that goes beyond the partially explored state space. The answer to this question is 1: the only chance of not leaving the explored state space is for the system to take the loop covering states 1, 2, 3, and 4 an infinite number of times. However, the probability of this occurring, as more interactions take place, tends to 0. For this reason, we must ask questions that are related to some notion of time passage, bounding this question within a certain timeframe. This is akin to reliability approaches that measure mean time to failure (MTTF) or probability of failure upon demand (PFD).

Consequently, in this setting, given that time is not explicitly modelled, sensible questions could be posed in terms of, for instance, the number of patient-controlled actions performed: `vital`, `panic`, and `stop`. We can compute the probability of the system executing beyond the explored space within less than 1 patient-controlled operation, which turns out to be 0.4 (the sum of 0.05 and 0.35 on transitions from state 0 that leave the squared section in Figure 3(b)).

A more interesting case is computing the probability of escaping the state space within less than 2 patient-controlled operations. Given that in state 4 a non-deterministic choice occurs, this question cannot be answered with one singular value as we do not know the likelihood of `changeDose` occurring over `changeDrug` and `noChange`. For such a query, a maximum and minimum probability must be given. The *maximum* probability of leaving the partial state space is, assuming the worst case, that in state 4 the `changeDose` transition is always taken, thus it is 1 (0.35 plus 0.05 from transitions on state 1, plus 0.6×0.3 from the path 0, 1, 2 and then exiting; plus $0.6 \times 0.7 \times 1$ from the path 0, 1, 2, 3, 4 and exiting on `changeDose`). Conversely, the *minimum* probability of escaping the state space in less than 2 patient-controlled operations is 0.58: 0.4 which corresponds to exiting the state space on state 1 plus 0.18 (0.6×0.3) from exiting on state 2. We do not add any other exiting case as we assume the most optimistic scenario in which `noChange` is taken, leading to a scenario in which exiting is not possible before another patient-controlled action. All other assumptions on how the non-deterministic choice on state 4 is resolved yields a probability of exiting between the computed minimum and maximum: 0.58 and 1.

It is straightforward to compute automatically the minimum and maximum probabilities of escaping the explored state space for an arbitrary number of patient-controlled actions. For instance the likelihood of escaping in less than 20 patient-controlled operations is between 0.999 and 1.

What does the range $[0.999 - 1]$ tell us? It tells us that for a very reasonable execution of the TeleAssistance software in which a patient, interacting with the software as modelled in Figure 2, performs up to 20 operations, the likelihood of a reaching a state that has not been verified by the model checker is extremely high. It also tells us that the fact that the model checker did not find a property violation when we checked the composition of the software model of Figure 1 and the non-deterministic patient model of Figure 2 is quite irrelevant irrespective of the number of states the model checker did manage to explore before running out of memory.

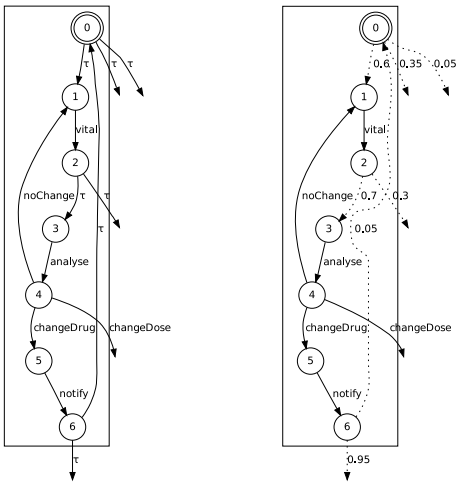

Figure 3: Partial state space exploration (a), probabilistically annotated (b)

Of course, in this trivial example we could have guessed, by simply observing how small the explored state space was against what we imagine is the complete state space, that the fact that no violations were found did not make a se-

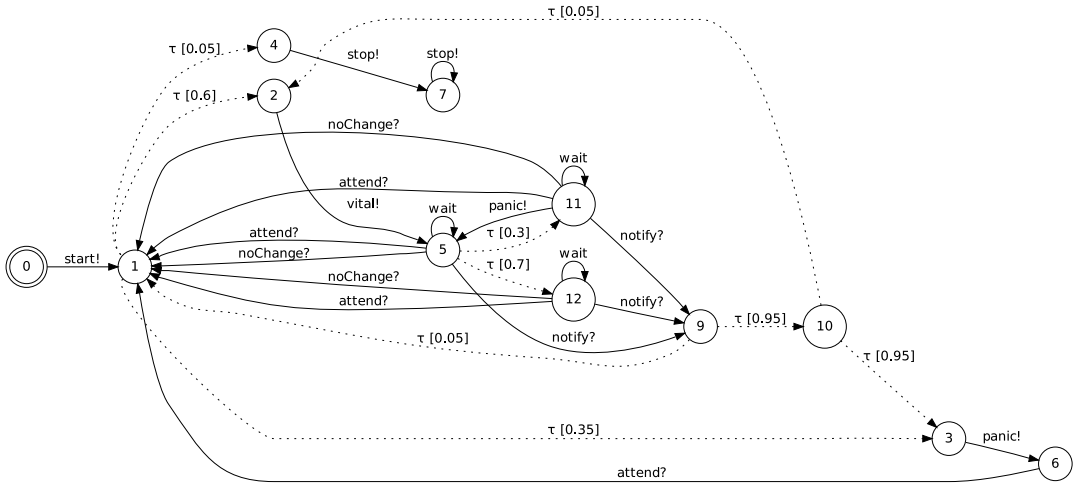

Figure 2: The patient's behavioural model. The probabilistic behavioural model can be obtained by annotating the transitions with the probabilities between brackets (ENV, [PENV]).

rious case for the dependability of the system. However, we hypothesise that in the normal case, where inspecting the partial exploration manually is impossible, this kind of analysis can provide feedback on the extent to which not having found errors in a partial exploration is relevant.

In the next sections, we outline our approach in a more formal context, explaining how the analysis could be formalised and some the theoretical results needed to argue its soundness. We then show some preliminary validation that we have done in larger (and proportionally more significant) partial explorations than the one discussed above.

3. BACKGROUND

In this section we present the formalisms and results we leverage on to perform our approach on the subject. In the first place, our probabilistic quantification is inspired on the concept of environmental and operational profiles as introduced by Musa [13]. In particular, we model these operational profiles via an automata based formalism, Probabilistic Interface Automata (PIA), that we have introduced in [15] and which is defined as follows.

Definition 3.1 (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* $M_{MDP} =< S_M, s^0_M, A_M, R_M >$.

Probabilistic Interface Automata share many of the characteristics of reactive probabilistic systems [16], such as Markov Decision Processes, and of Interface Automata [3]. In particular, the notion of *composability* is lifted from the latter. Also, note that a classic interface automaton is a particular case of a PIA, and that a PIA A has an *underlying interface automaton*, noted $A\downarrow$, defined as follows:

Definition 3.2 (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. Probabilistic Interface Automata also extend Interface Automata notion of illegal states with the following restrictions, which make them suitable for the modelling of system environments. The reader is referred to [15] for further details, complete definitions and motivations.

Definition 3.3 (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) $\exists a \in Shared(P, Q)$ *such that* $a \in A^O_P(s) \wedge a \notin A^I_Q(q)$*, or conversely* $\exists a \in Shared(P, Q)$ *such that* $a \in A^I_P(s) \wedge a \notin A^O_Q(q)$.

ii) s (resp. q) belongs to a transient loop α *in* P *(resp.* Q*); that is, such that any action* $a \in \alpha$ *verifies* $a \in A_P \setminus Shared(P, Q)$ *(resp.* $a \in A_Q \setminus Shared(P, Q)$*).*

Probabilistic interface automata product is carried out much in the spirit of a combination of Interface Automata product and reactive probabilistic automata product, as the following definition illustrates.

Definition 3.4 (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 [3], 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 reactive probabilistic product.*

In this way, PIA product is governed by both the probabilistic product of its probabilistic transitions, and synchronises its input, output and internal (hidden) actions in the same way as Interface Automata synchronises them. The operand $\otimes$ stands for both IA and PIA product, and its meaning in each case will be clear from the context. Definitions 3.3 and 3.4 allow for the definition of a *valid environment*.

DEFINITION 3.5 (VALID ENVIRONMENT). *Given an interface automaton P, another nonempty probabilistic 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 $P \otimes Q$.*

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* [1] is a viable logic, since we can leverage on their underlying reactive probabilistic structure and the resolution of non-determinism via the use of schedulers or adversaries. Probabilistic Interface Automata 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 state the basis of this idea.

THEOREM 3.1 (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 a finite execution fragment 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.*

COROLLARY 3.1 (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 $E \models \phi \Rightarrow E \otimes M \models \phi$.*

The reader is referred to [15] for insight into the theorem. These definitions and theorem provide us with an adequate framework for expressing properties based on environmental properties, and ensure that the approach that we introduce in the following section is sound with respect to these properties and the model under analysis.

4. APPROACH

In this section we outline the formal aspects that will be needed to underpin the approach presented in this position paper.

Our analysis assumes the existence of a system-under-analysis model (SUA) and a model of its environment (ENV). This environment ENV should be a valid environment (see Definition 3.5) with respect to SUA. Performing an (incomplete) model checking analysis of these models, we obtain a partial exploration of their composition (EXP). EXP is a subgraph of $SUA \otimes ENV$ that contains the initial state and such that all states in EXP are reachable from this initial state. A *sink* state is then added to EXP, and for every transition t in $SUA \otimes ENV$ such that $source(t)$ is in EXP and $target(t)$ is *not* in EXP, an internal transition from $source(t)$ to sink is also added. Aditionally, this sink state stutters all input labels.

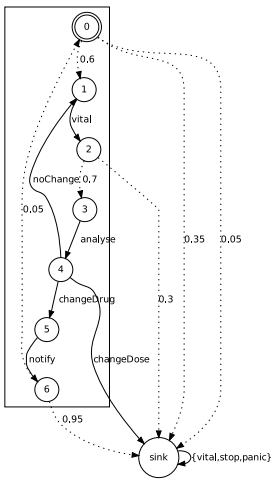

Figure 4: Probabilistically annotated partial exploration $PEXP$

For the second part of the analysis, we require also the modelling of a Probabilistic Environment model ($PENV$) that captures the probabilistic behaviour model of the environment. This $PENV$ model must be such that it is equivalent to the ENV model if its probabilities are dropped (*i.e.*, $PENV \downarrow = ENV$, see Definition 3.2). Currently, this notion of equivalence is defined to be isomorphism, although we plan to weaken that requirement (probably to weak bisimulation) in order to attain greater generality. The other required input to the method is a *tick granularity* definition against which the size of EXP will be evaluated: we currently define the number of relevant environmentally-controlled performed actions as this tick measure. Then, once we have defined a maximum number (MAX) of ticks, we can perform our quantitative analysis. The formal question we look to answer is what is the measure of the set of all cones induced by a finite execution fragment α_{PENV} in $PENV$ that are such that there exists a (finite) run α in EXP which satisfies both (i) α_{PENV} is the restriction of α to the language of $PENV$; and (ii) α reaches the sink state in less than or exactly MAX steps.

To achieve the aforementioned goal we process the artefacts in the following way: first, we annotate EXP transitions with probabilities extracted from the probabilistic environment. The result is called the partial probabilistic exploration ($PEXP$) and it is characterised by the fact that $PEXP$ is equivalent to composing $PENV$ with SUA and cutting out the same partial exploration (in terms of its traversed states and transitions) as when EXP was produced. For example, Figure 4 depicts the $PEXP$ model that would be obtained from the EXP partial exploration of Figure 3(a). Note that $PENV$ is a valid environment for SUA in the context of Probabilistic Interface Automata. Thanks to being compatible interfaces, cone measures in $PEXP$ are known to be the ones dictated by the cone counterpart in $PENV$ (see Theorem 3.1). This is what we actually need since we want to measure the probability of the environment

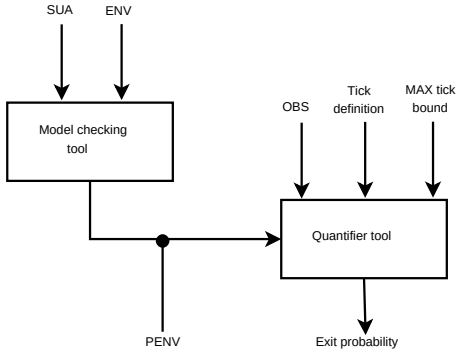

Figure 5: Workflow of the quantification approach

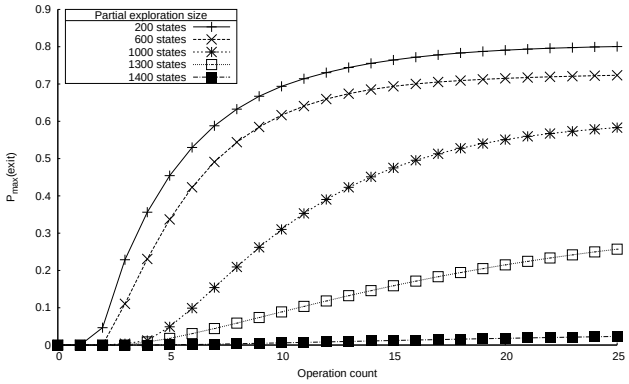

Figure 6: Obtained results for partial explorations, varying sizes, counting environmentally-controlled operations.

driving the system into non-visited states.

We are almost at the point where reliability questions relative to the explored state space can be posed. In order to be able to pose and answer these questions, we build an observer OBS that recognizes the relevant ticks (that is, the alphabet of OBS is comprised of these relevant action labels) and increments a counter when synchronising, with no further interaction with $PEXP$. This being the only restrictions in the forming of OBS, it turns out that OBS is a non-blocking automaton that can be composed with $PEXP$ with no behavioural side effects. Finally, we perform checks querying for the values of $P_{min}[\diamond(sink \wedge ticks \leq MAX)]$ and $P_{max}[\diamond(sink \wedge ticks \leq MAX)]$, for the required values of MAX, which allow us to quantify the desired cones. Figure 5 depicts the complete process of the approach.

5. PRELIMINARY VALIDATION

In this section, we describe some experiments that were conducted in order to explore some preliminary hypotheses regarding the technique and more importantly, to observe, aiming to gain additional insights, the behaviour of the technique when applied to a larger exemplar than the ones presented in Section 2.

The experiments were conducted using two tools. In the first place, the Labelled Transition System Analyser (LTSA) [11], which suits our approach since it is an explicit-state model checker and can be set up so as not to use any additional optimizations. We coupled LTSA along with the probabilistic model checker PRISM [7]. The former is a non-probabilistic model checker which we modified *ad-hoc* in order to instrument different resource depletion scenarios and various search heuristics. The latter was used to quantify the explored state spaces produced by LTSA.

The exemplar used for our experiments was a slightly more complex version of the TeleAssistance system presented in Section 2 which, among other variations, includes a more complicated path to the error state, which will only be reached if the panic button is pressed more than three times by the patient between a request operation and its response, and *without* the TeleAssistance system having responded by dispatching a First Aid Squad in response to those panic button presses. Being more complex, checking the property against the composition of the software and patient model results in a bigger state space—the complete model in this version spans 2112 states.

Our first two experiments were designed as a sanity check: there are two, intuitive, basic hypotheses that the approach should confirm. First, that given a fixed partial exploration, the probability of observing an execution that escapes the explored state space should increase when the maximum number of ticks is increased. Second, that given a fixed maximum number of ticks, the probability of escaping the explored state space decreases as the explored state space is extended.

The execution of the experiments involved model checking the TeleAssistance software model composed with the patient model in LTSA and outputting the explored state spaces at increments of one hundred states. The search order used was breadth-first based (the state-space exploration heuristic used by LTSA). Hence, a series of partial explorations are generated, each of which subsumes the previous ones, on which our second hypothesis could be studied. For each of these partial explorations, we utilized the PRISM batch model checking facilities to check for the values of pCTL* formulae $P_{min}[\diamond(sink \wedge ticks \leq MAX)]$ and $P_{max}[\diamond(sink \wedge ticks \leq MAX)]$, in both cases for $1 \leq MAX \leq 25$. This formulae capture the probabilities of reaching the $sink$ state, restricted to a MAX amount of $ticks$. Recall that the ticks are observed via the OBS automaton, which increments the tick count for every environmentally-controlled operation performed.

The data collected in these experiments is depicted in Figure 6. The vertical axis shows the maximum probability of evolving out of the explored state space. It is worth noting that the minimum probability turned out to be always zero for this system: because of its highly non-deterministic behaviour, it is always possible in the given explorations, for at least one scheduler, to loop indefinitely within the explored portion of the state space.

The horizontal axis shows the increasing values of the timing measure, the number of environmentally-controlled operations that have occurred. The different lines correspond to the various partial explorations, where every exploration subsumes the states of all smaller ones. Note that we only show explorations up to size 1400 as the model checker finds the error when exploring at least 1500 states with this breadth-first strategy.

Figure 6 shows that the probability of escaping the explored state space behaves as expected: it increases when

time (the maximum number of ticks) increases and decreases as the state space increases. More interesting is the shape of the curves produced: they tend to stabilise on a specific probability as the number of environmentally-controlled operations increases. This is convenient as the analyses we foresee will be on "large" time-scales and such behaviour could mean that the fine-grained choice of the maximum number of ticks (e.g. 1000 vs. 1001 ticks) will not have a large impact on the upper bound to the probability of escaping the explored state space.

Another observation that can be made from Figure 6 is the qualitative leap obtained when analysing 1400 states. Although the model checker finds the error exploring less than an extra 100 more states, the probability of exiting the explored state space within these 1400 explored states tends to stabilise in the vicinity of ≈ 0.025. Furthermore, the probability of violating the critical system property, if computed on the full state space, is stabilised around 0.0000215. It is noteworthy that the curves are tending to the actual probability of error as the number of states explored increases. This gives some experimental assurance that the proposed method for measuring the partial state space is sound.

Another interesting point is that relatively large explorations (with respect to the total state space) such as the 1000 state exploration yield quite high probabilities of exiting the explored state space. We understand this as being an indication that, when reasoning about partial explorations, a careful distinction should be made between the *quantity* of states explored and the *quality* of those states: LTSA performs a breadth-first search of the composition of the software, patient and property models. This heuristic does not take into account the paths that are more likely to be taken as a result of the probabilistic behaviour of the patient. We hypothesise that different heuristics will achieve different probabilities of escaping the partially explored state space and that some of them may produce high reliability measures (*i.e.* low probabilities of escaping) and better approximate the actual probability of error in the complete state space. If this hypothesis proves correct, then it may be possible to design search heuristics that attempt to maximize the relevance of the state space explored based on the probabilistic information on the environment behaviour that is available; an operational profile guided exploration is one of our longer term research goals.

6. DISCUSSION AND RELATED WORK

The area of software reliability engineering has, from its inception, been concerned with producing quantitative (rather than qualitative) results in the form of metrics such as mean time to failure (MTTF), mean time between failures (MTBF) or probability of failure upon demand (PFD). Frequentistic statistical approaches to testing and reliability metrics have their roots in work such as [14, 12, 9, 13] where the aim is to provide confidence bounds for the truth of dependability claims when no failure is found by testing.

We believe that results that exploit inconclusive model-checking explorations should be integrated into reasoning frameworks for dependability cases [8]. We foresee the development of theories and tools to obtain useful claims from potentially incomplete verification phases that should be analysable in the same realm as claims inferred by operational testing and other methods in order to gain confidence on system reliability.

To the best of our knowledge there is no previous work on using incomplete model checking explorations to support dependability claims. This work tries to take a first step towards that direction.

Unlike reliability testing, claims obtained by our method are quantifications for a series of reliability queries (a set of curves) which are likely to be more conservative but are yielded with absolute confidence. That is, in reliability testing, reliability claims are accompanied by a confidence level based on an appropriate statistical analysis. These approach can trade off higher reliability claims for lower confidence levels and vice versa. In our approach, the confidence level is absolute and as a consequence the reported reliability may be more conservative than in reliability testing approaches.

The use of operational profiles to inform reliability claims has been done by others. For instance, Whittaker et.al. [17] have presented a statistical testing approach based on Markov chains. The use of these probabilistic chains allows for test input sequences to be generated from multiple probability distributions. The method proposed by the authors also creates a second Markov chain to encapsulate the history of the test, including any observed failure information. The influence of the failures is assessed through analytical computations on that chain. An important difference with our proposal is that the Markovian models do not distinguish between environmental behaviour and that of the system under test, and do not support purely non-deterministic transitions.

A recent approach that bridges the gap between (random) testing and model cheking is that of *approximate* model checking. For example, Smolka and Grosu [5] take a Monte Carlo approach to model checking, as they randomly generate paths in an attempt to find violations, and in so doing they arrive at a measure of certainty about the property being violated or not. However, as in the case of testing, the probability associated to the potential violation of the property is subject to a confidence value, which in order to be sufficiently high, requires large amounts of paths to be generated. Furthermore, other key differences with respect to our approach are that neither the probability measure is relative to an operational profile nor the problem is the one of measuring an amount of verification achieved by an arbitrary incomplete yet systematic exploration.

[6, 1] and others have proposed the use of probabilistically enriched automata models to model and analyze designs and protocols of reactive and distributed systems. Probabilities may be used to describe system properties such as failure rates, activation rates and similar actions that may not always be modelled or analysed via classical Labelled Transition Systems. Although we do leverage our approach on such theories and tools, it is worth pointing out the unfeasibility, in our application context, of performing such a sort of check to obtain the probability to get into a failure state. In fact, unfortunately, if a problem is intractable to a conventional model checking, then it is so for probabilistic model checking which is at least as expensive as a traditional one. That is why our approach submits to the probabilistic model checker a fragment of the state space (the part succesfully explored by the conventional model checker) for querying in tools like PRISM.

7. CONCLUSIONS AND FURTHER WORK

Model checking is a promising technique for automatic

analysis of system behaviours. However, due to the state explosion problem, it is not uncommon for model checkers to run out of resources before any meaningful feedback is returned. The result is that a subset of the model's state space has been explored and no errors found, but the engineer is left with no information about the relevance of the error-free portion of the model that has been checked.

In this paper we have presented an outline of an approach for quantifying, based on the probabilistic behaviour of the environment, a partial exploration resulting from an aborted model check. We have also outlined the underlying formal framework needed to underpin the approach and also some initial experiments performed to gain insights on the implications of using the approach.

To become a sound and practical technique, the ideas presented in this position paper require a number of future developments. Firstly and foremost, the formal underpinnings of the approach must be finalised. Although the basis for this underpinning, Probabilistic Interface Automata [15], has been studied, there are a number of theoretical results that need to be proved in order to complete the argument as to why the various compositions, partial explorations and measurements are sound.

Secondly, significant work will be required to make the approach scale. As mentioned previously, it is unlikely that the partial exploration produced by a conventional model checker that has run out of resources will be analysable in a probabilistic model checker, that must add to the partial exploration not only probabilistic information, but also the observer model for tracking time ticks and the observer required for answering reliability queries. In addition to the development of specific model checking algorithms, the study of behavioural property preserving abstractions and careful treatment of the observers required to pose the reliability questions may be beneficial.

Related to the above point is the development of appropriate tool support. One of the challenges here is to find efficient ways of combining conventional and probabilistic model checking technology to support the algorithmic aspects of the approach. Our approach currently assumes that the model checker that produces the partial exploration uses an un-optimised explicit state representation. Another line of future work is to study the applicability of our approach to partial explorations generated by model checkers that use optimization strategies and different representations of the state space.

A more longer term line of research is the study of the impact that different search heuristics applied by model checkers have on the quantification of partial explorations and in particular the development of new heuristics, that will most likely exploit the known probabilistic behaviour of the environment, to minimize the probability of the system escaping the explored state space and thus provide a tighter bound on the probability of failure being encountered.

Finally, from a methodological standpoint there are a number of areas that need to be developed. In this paper, we used environmentally-controlled actions as the basis for a notion of time to produce reliability measures. It stands to reason that there may be other, equally suitable or better, measure notions. What are appropriate criteria for selecting these measures? In addition, given these different measures, we should study how the different reliability curves produced by the technique should be interpreted.

8. ACKNOWLEDGEMENTS

This work was partially supported by CONICET and grants UBACyT X021, ANPCyT 2005 32440, ANCPCyT PICT PAE 37279 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] E. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, 1999.

[3] L. de Alfaro and T. A. Henzinger. Interface automata. In *ESEC/FSE'01 Proceedings*, pages 109–120. ACM, 2001.

[4] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time adaptation. In *ICSE '09 Proceedings*. IEEE Computer Society, 2009.

[5] R. Grosu and S. Smolka. Monte carlo model checking. *TACAS 2005 Proceedings*, 3440(631):271–286, 2005.

[6] H. Hansson and B. Jonsson. A framework for reasoning about time and reliability. *Real-Time Systems Symposium 1989 Proceedings*, 1989.

[7] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *TACAS'06 Proceedings*, volume 3920, pages 441–444. Springer, 2006.

[8] B. Littlewood. Limits to Dependability Assurance–A Controversy Revisited. *ICSE'07 Proceedings Companion*, 2007.

[9] B. Littlewood and L. Strigini. Validation of ultrahigh dependability for software-based systems. *Communications of the ACM*, 36(11), 1993.

[10] B. Littlewood and L. Strigini. Software reliability and dependability: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering-ICSE*. ACM, 2000.

[11] J. Magee. Behavioral analysis of software architectures using LTSA. In *ICSE'99 Proceedings*. ACM, 1999.

[12] K. Miller, L. Morell, R. Noonan, S. Park, D. Nicol, B. Murrill, and M. Voas. Estimating the probability of failure when testing reveals no failures. *IEEE Transactions on Software Engineering*, 18(1), 1992.

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

[14] D. Parnas. Assessment of safety-critical software in nuclear power plants. *Nuclear Safety*, 32(2), 1991.

[15] E. Pavese, V. Braberman, and S. Uchitel. Probabilistic environments in the quantitative analysis of (non-probabilistic) behaviour models. In *ESEC/FSE '09 Proceedings*, pages 335–344. ACM, 2009.

[16] R. van Glabbeek, S. Smolka, B. Steffen, and C. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *LICS'90 Proceedings.*, 1990.

[17] J. Whittaker and M. Thomason. A Markov chain model for statistical software testing. *IEEE Transactions on Software Engineering*, 20(10), 1994.