Stuttering in Abstract Probabilistic Automata

Benoît Delahaye,
Aalborg University, Denmark
benoit@cs.aau.dk

Kim G. Larsen,
Aalborg University, Denmark
kgl@cs.aau.dk

Axel Legay,
INRIA/IRISA, France
axel.legay@irisa.fr

Mikkel L. Pedersen,
Aalborg University, Denmark
mikkelp@cs.aau.dk

Andrzej Wąsowski,
ITU University, Copenhagen, Denmark
wasowskip@itu.dk

1 Context

Nowadays, systems are tremendously big and complex and mostly result from the assembling of several components. These components are usually designed by teams working *independently* but with a common agreement on what the interface of each component should be. These interfaces precise the behaviors expected from each component as well as the environment in which they can be used, but do not impose any constraint on how the components are implemented.

Instead of relying on Word/Excel text documents or modeling languages such as UML/XML, as is usually done in practice, we recommend relying most possibly on mathematically sound formalisms. Mathematical foundations that allow to reason at the abstract level of interfaces, in order to infer properties of the global implementation, and to design or to advisedly (re)use components is a very active research area, known as *compositional reasoning* [3]. Aiming at practical applications *in fine*, the software engineering point of view naturally leads to the following requirements for a good theory of interfaces.

- **Satisfaction.** It should be decidable whether an interface admits an implementation (a model). One should also be capable of synthesizing an implementation for such an interface. In this paper, implementation shall not be viewed as a programming language but rather as a mathematical object that represents a set of programming languages sharing common properties.

- **Refinement.** It is important to be able to replace a component by another one without modifying the behaviors of the whole design. At the level of interfaces, this corresponds to the concept of *Refinement*. Refinement allows replacing, in any context, an interface by a more detailed version of it. Refinement should entail substitutability of interface implementations, meaning that every implementation satisfying a refinement also satisfies the larger interface. Refinement thus extends the classical simulation relation between systems to specifications.

- **Conjunction.** Large systems are concurrently developed for their different *aspects* or *viewpoints* by different teams using different frameworks and tools. The issue of dealing with multiple aspects or multiple viewpoints is thus essential. This implies that several interfaces are associated with a given component, namely (at least) one per viewpoint. These interfaces are to be interpreted in a conjunctive way.

- **Composition.** The interface theory should also provide a combination operation, which reflects the standard interaction/composition between systems.

In addition, any good interface theory should guarantee classical properties such as independent implementability that allows to combine components in various orders.

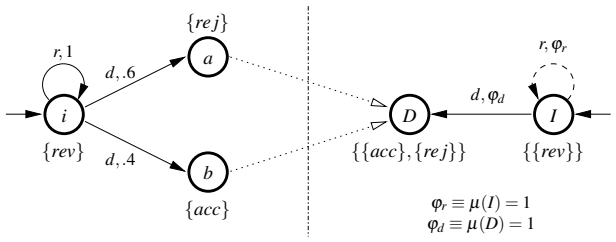

Figure 1: Implementation PA (left) and specification APA (right) of a reviewer

2 The Challenge

Building good interface theories has been the subject of intensive studies among which one finds classical logical specifications, various process algebrae such as CSP, or Input/Output automata/interfaces (see [4, 1, 6]). Recently, a new series of works has concentrated on *modal specification* [5], a language theoretic account of a fragment of the modal mu-calculus logic which admits a richer composition algebra with composition, conjunction and even residuation operators. The interest of modal automata lies in the fact that this is the only model where both composition and conjunction can be expressed and computed in a very simple and elegant manner [7].

As soon as systems include randomized algorithms, probabilistic protocols, or interact with physical environment, probabilistic models are required to reason about them. This is exacerbated by requirements for fault tolerance, when systems need to be analyzed quantitatively for the amount of failure they can tolerate, or for the delays that may appear. As Henzinger and Sifakis [3] point out, introducing probabilities into design theories allows assessing dependability of IT systems in the same manner as commonly practiced in other engineering disciplines.

Our response to this problem was to propose Constraint Markov Chains (CMC) that is a complete specification theory for pure stochastic systems, namely Markov Chains (MC). Roughly speaking, a CMC is a MC equipped with a constraint on the next-state probabilities from any state. An implementation for a CMC is thus a MC, whose next-state probability distribution satisfies the constraint associated with each state. Contrary to Interval Markov Chains where sets of distributions are represented by intervals, CMCs are closed under both composition and conjunction.

However CMCs do not permit to reason on non-deterministic behaviors, hence on Probabilistic Automata (PA). A solution to this problem was provided in [2], where we have presented Abstract Probabilistic Automata (APA), the first complete specification theory for probabilistic automata. APAs are specifications that represents a possibly infinite set of PAs. APAs combine Modal Automata and CMCs – the abstractions for labelled transition systems and Markov Chains, respectively. The theory has been implemented and tested on several case studies.

Example 1. *Consider the implementation (left) and specification (right) of a reviewer given in Figure 1. The specification specifies that there are two possible transitions from initial state I: a may transition with action r (read) and a must transition with action d (decide). May transitions are represented with dashed arrow, while must transitions are represented with plain arrow. The probability distributions associated with these actions are specified by the constraints φ_r and φ_d, respectively. One can see that the implementation gives a more precise behavior of the reviewer: action r loops back to initial state i with probability* 1*, while the decision leads to state a (reject) with probability* .6 *and to state b (accept) with probability* .4*. Satisfaction between implementation and specification lifts the classical notion of simulation for PA to APA as follows: (1) all must transitions of the specification must be matched with transitions in the implementations, and (2) all transitions in the implementation must be matched with may transitions in the specification. Additionally, we have to check that the probability distributions in*

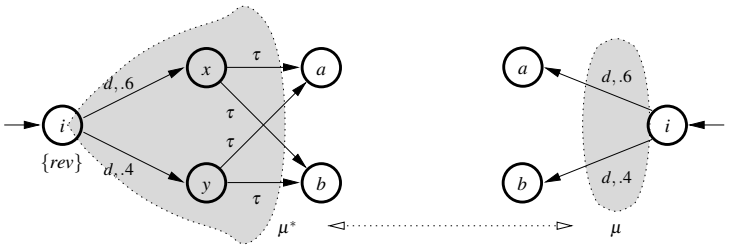

Figure 2: Illustration of weak bisimulation between PAs

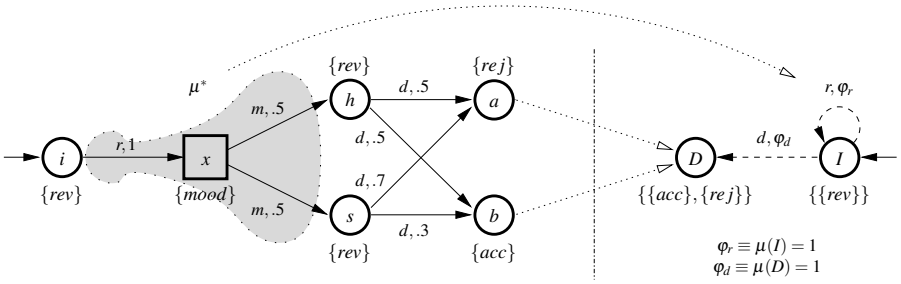

Figure 3: Illustration of stutter satisfaction for APAs

the implementation are matched with probability distributions in the specification that satisfy the given constraints.

However, while APA is a complete specification theory in itself, it still lacks some important design features that are needed to make this theory attractive from an engineering point of view. As an example, in the process of incremental design, it is sometimes necessary to incrementally widen the scope of implementations. Usually, for PA, the latter is done by permitting the addition of hidden actions also called stutter steps. Such actions clearly complicate the definition and the computation of operations such as bisimulation/simulation and composition. As an example, simulation between PAs has to be lifted to weak simulation as illustrated with the following example.

Example 2. *In Figure 2, we illustrate the notion of weak bisimulations between two given PAs. In the left PA, internal action τ has to be taken into account. In this case, one has to compute the overall probability μ^* of reaching states a and b from state i and compare this probability to the transition probability μ in the right PA. In essence, weak bisimulation holds if these probabilities are equal.*

The objectives of this presentation are to survey the theory of APAs as well as to present a solution to the treatment of hidden actions. Our solution takes the form of an extension of the one proposed to handle internal actions of PAs: we say that a distribution μ^* is reached via a stutter transition a if there exists a scheduler for the internal transitions that can follow the action a, such that the overal distribution reached after executing only internal actions is μ^*. The idea is illustrated with the following example.

Example 3. *In our formalism, hidden actions and local states are made explicit. As illustrated in Figure 3 the state space of PAs is split into visible (circles) and local (boxes) states. Invisible states can only perform hidden actions. Stuttering satisfaction then computes all internal overall probabilities of reaching visible states, and verifies that these overall probabilities match distributions in the specification satisfying the constraints.*

During the presentation, we shall show how extending APA with internal actions impacts both the definition and the algorithmic treatment of their operations and the properties they guarantee. Finally, we will also present a new logical characterization of APAs that takes internal actions into account.

References

[1] L. de Alfaro and T. A. Henzinger. Interface automata. In *Proc. 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC / SIGSOFT FSE), Vienna, Austria*, pages 109–120. ACM Press, 2001.

[2] Benoît Delahaye, Joost-Pieter Katoen, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, Falak Sher, and Andrzej Wasowski. Abstract probabilistic automata. In *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings*, volume 6538 of *Lecture Notes in Computer Science*, pages 324–339. Springer, 2011.

[3] T. A. Henzinger and J. Sifakis. The embedded systems design challenge. In *Proc. 14th International Symposium on Formal Methods (FM), Hamilton, Canada*, volume 4085 of *lncs*, pages 1–15. Springer, 2006.

[4] H. Hermanns, U. Herzog, and J-P. Katoen. Process algebra for performance evaluation. *Theor. Comput. Sci.*, 274(1-2):43–87, 2002.

[5] K. G. Larsen. Modal specifications. In *Proc. International Workshop on Automatic Verification Methods for Finite State Systems (AVMS), Grenoble, France*, volume 407 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 1989.

[6] Nancy Lynch and Mark R. Tuttle. An introduction to Input/Output automata. *CWI-quarterly*, 2(3), 1989.

[7] Jean-Baptiste Raclet. *Quotient de spécifications pour la réutilisation de composants*. PhD thesis, Université de Rennes I, december 2007. (In French).