APAC: A Tool for Reasoning About Abstract Probabilistic Automata

Benoît Delahaye*, Kim G. Larsen†, Axel Legay*, Mikkel L. Pedersen†, and Andrzej Wąsowski§

*INRIA/IRISA
Rennes, France
{benoit.delahaye, axel.legay}@irisa.fr

†Aalborg University
Denmark
{kgl,mikkelp}@cs.aau.dk

§IT University
Copenhagen, Denmark
wasowski@itu.dk

***Abstract*—We recently introduced Abstract Probabilistic Automata (APA), a new powerful abstraction formalism for probabilistic automata. Our theory is equipped with a series of aggressive abstraction techniques for state-space reduction as well as a specification theory for both logical and structural comparisons. This paper reports on the implementation of the approach in the Abstract Probabilistic Automata Checker toolset.**

I. Context

Probabilistic Automata (PA) [1] is a formalism for modeling systems that exhibit stochastic and non-deterministic behaviour, e.g., randomized distributed algorithms and fault tolerant systems. In each state, a PA resolves a non-deterministic choice and then moves to the successor state according to some probability distribution. Recently [2], [3], we proposed Abstract Probabilistic Automata (APA), a new powerful abstraction formalism for PAs equipped with (1) a series of aggressive abstraction techniques for state-space reduction, and (2) a specification theory for component-based design in the spirit of [4]. This short paper reports on the Abstract Probabilistic Automata Checker toolset (APAC), an implementation of the APAs theory (and hence of the first interface theory for stochastic systems), based on the SMT-solver Z3 [5]. The tool, tutorials and a manual can be found at http://www.cs.aau.dk/~mikkelp/apac.

The APA model. Syntactically, an APA is a PA whose transitions are equipped with may and must modalities [6], and whose probability distributions are replaced by constraints like in Constraint Markov Chains (CMC) [7]. Semantically, an APA represents a possibly infinite set of PAs that are its implementations. Each state is also equipped with a set of atomic propositions used to define additional hypotheses on the implementations. The must modality requires that the transition has to be present in any implementation, while the may modality permits its absence. In addition, any distribution associated to the transition must satisfy the constraint specified by the APA. Consider APA N_2 of Fig. 1b. Three transitions leave the state s'_1: an a-must-transition to constraint φ', an a-may-transition to a constraint assigning probability 1 to s'_5, and a b-may-transition going with probability 1 to s'_1. Any implementation of N_2 has an a-transition targeting a distribution satisfying φ'; the other transitions are optional.

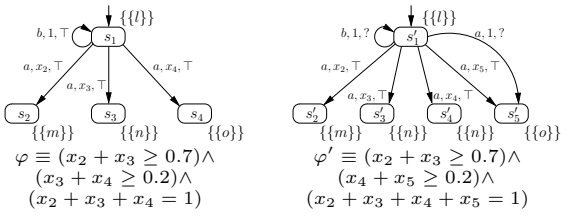
Abstraction. We propose two notions of abstraction. The first one is classical and aims at reducing the state-space of the system by lumping equivalence classes. The second one is used to abstract a constraint φ by the smallest intervals in which all satisfying distributions can be embedded.

Specification Theory. The APA model also serves as a specification theory for stochastic systems. This is used to decompose the design and hence reduce its complexity. We propose a structural composition that allows to combine components and a logical composition that allows to combine requirements (take intersection of sets of implementations). These operations unite those defined on modal automata [6] and CMCs [7]. For example, in order to structurally compose two APAs, we combine transitions labeled by the same letter. The combination of a may with a must or a may transition transition leads to a may transition. Constraints are combined in a product-like manner. Given constraints $\varphi_1 \in C(S_1)$ and $\varphi_2 \in C(S_2)$, their product $\varphi_1\varphi_2$ is defined such that for all distribution μ satisfying $\varphi_1\varphi_2$, written $\mu \in Sat(\varphi_1\varphi_2)$, there exists $\mu_1 \in Sat(\varphi_1)$ and $\mu_2 \in Sat(\varphi_2)$ such that $\mu(s_1, s_2) = \mu_1(s_1)\mu_2(s_2)$ for all $(s_1, s_2) \in S_1 \times S_2$.

Refinement. Refinement compares APAs and hence also sets of implementations. Intuitively, if N_1 refines N_2, then any must (resp. may) of N_2 (resp. N_1) should be matched in N_1 (resp. N_2) in an alternating simulation manner. Moreover, the matching has to agree on the constraints as illustrated hereafter. Consider the two APAs in Fig. 1 with state space S and S', respectively. $\mathcal{R} = \{(s_1, s'_1), (s_2, s'_2), (s_3, s'_3), (s_3, s'_4), (s_4, s'_5)\}$ is a refinement relation. Indeed, the a-must-transition from s'_1 is matched by a must-transition in s_1, and the b-must-transition from s_1 is matched in s'_1, and φ and φ' agree. Indeed, for all distributions μ that satisfy φ, the probability mass given to successor states by μ can be redistributed to equal a distribution μ' satisfying φ'. Let $\delta : S \to (S' \to [0, 1])$ be given as $(s_1, s'_1) \mapsto 1$, $(s_2, s'_2) \mapsto 1$, $(s_3, s'_3) \mapsto \gamma$, $(s_3, s'_4) \mapsto 1 - \gamma$, $(s_4, s'_5) \mapsto 1$, and 0 else, where $\gamma = \frac{0.7-\mu(s_2)}{\mu(s_3)}$, if $\mu(s_2) \leq 0.7$, and $\gamma = \frac{0.8-\mu(s_2)}{\mu(s_3)}$ else. For all distributions μ that satisfies φ, $\mu\delta$ will satisfy φ', and for all pairs (s, s') such that $\delta(s)(s') > 0$, $s \mathcal{R} s'$. Details can be found in [2], [3].

II. The APAC Tool

Input Language. APAC provides a simple intuitive textual language for specifying APAs and operations on them. The language follows the graphical description of APAs. For



(a) N_1 (b) N_2

Fig. 1: APAs N_1 and N_2.

```
INPUT:
Name: N1;
A:<a,b>;
AP:<l,m,n,o>;
state 1:<<l>>:
a!->x[1]=0.0 && x[2]+x[3]>=7/10 && x[3]+x[4]>=2/10;
b!->x[1]=1.0;
state 2:<<m>>:
state 3:<<n>>:
state 4:<<o>>:

Name: N2;
A:<a,b>;
AP:<l,m,n,o>;
state 1:<<l>>:
a!->x[1]=0.0 && x[2]+x[3]>=7/10 && x[4]+x[5]>=2/10;
a?->x[5]=1.0;
b?->x[1]=1.0;
state 2:<<m>>:
state 3:<<n>>:
state 4:<<n>>:
state 5:<<o>>:

check: N1 wref N2;
```

Fig. 2: Invoking refinement checking

example, the two APAs of Fig. 1 can be described as shown in Fig. 2. Each state is declared using the state keyword followed by a non-zero integer. Transitions are declared by their modalities (! for must and ? for may) followed by the keyword $\rightarrow$ and the constraint on the distribution on the successor states. For example, $x[2]+x[3]\geq 0.7$ imposes that the probability mass assigned to states 2 and 3 is greater than 0.7.

APAC, itself implemented in C#, parses the input language and builds internal representations for the corresponding APAs. Then, operations are applied to create new APAs. All the operations are reduced to suitable constraint manipulations in Z3 [5], an efficient SMT solver supporting quantifier elimination. For instance, in order to check refinement we perform quantifier elimination in the formula $\forall \mu \in Sat(\varphi), \exists \delta, \exists \mu' \in Sat(\varphi') : \mu\delta = \mu'$. As all variables are quantified, the procedure will evaluate the formula to true or false. If refinement does not hold, then APAC can generate a counter example.

Due to limitations of Z3, APAC only handles linear constraints. Fortunately, linearity of constraints is known to be preserved by all the operations in our theory, except structural composition. We solve this problem by using a linear abstraction of the constraints. Given constraints $\varphi_1 \in C(S_1)$ and $\varphi_2 \in C(S_2)$, their combination φ is defined such that for all $\mu \in Sat(\varphi)$, $\sum_{s_1 \in S_1} \mu(s_1, s_2) \in Sat(\varphi_2)$ and $\sum_{s_2 \in S_2} \mu(s_1, s_2) \in Sat(\varphi_1)$. Also in other areas, such as control theory, non-linear systems have to be abstracted by linear ones for efficiency reasons.

APAC supports generalized model checking of a disjunction-free extension of Hennessy-Milner logic [8] over APAs. For example, the formula $[a]_{\geq 0.2}\{\{n\}\}$ specifies that for all a-may-transitions leading to constraint φ, it holds that all satisfying distributions of φ give probability of at least 0.2 to states with valuations being subset of $\{\{n\}\}$. Obviously N_1 does not satisfy this formula. The full definition and input grammar of the logic can be found at http://www.cs.aau.dk/~mikkelp/apac. The logic is sound and complete i.e. an APA N satisfies a formula φ if and only if all implementations of N satisfy φ.

TABLE I: State abstraction

states	abstract states	time
500	5	9509 ms
500	10	16213 ms
500	50	62727 ms
500	100	96399 ms

III. Results and conclusion

APAC is clearly a research tool, still undergoing heavy development. While not yet mature enough to handle industrial case studies, the tool is already able to decide refinement of large-size case studies. Not surprisingly, the running time of quantifier elimination increases using an increasing number of quantified variables. In Table I, we see that time increases linearly with the precision of the state abstraction. More details are available on the APAC website. APAC is one of the very firsts implementation of stochastic interface theories with abstraction primitives.

In the future we intend to improve the efficiency of APAC by implementing heuristics such as bisimulation reduction for APAs. We also plan to implement a graphical user interface. Here the main challenge is identifying a simple and easy to manipulate representation for transitions encompassing multiple arrows related with a probability distribution constraint.

APAC is a part of a broader effort to develop and apply specification theories to industrial problems [9]. Recently, we have achieved success with the ECDAR toolset [10] and modeling of real time systems. It is of interest to merge ECDAR and APAC, but this requires developing a new specification theory first.

References

[1] R. Segala and N. Lynch, "Probabilistic simulations for probabilistic processes," in *CONCUR*, ser. LNCS, vol. 836. Springer, 1994.

[2] B. Delahaye, J.-P. Katoen, K. G. Larsen, A. Legay, M. L. Pedersen, F. Sher, and A. Wasowski, "Abstract Probabilistic Automata," in *VMCAI*. Springer, 2011.

[3] B. Delahaye, J.-P. Katoen, K. G. Larsen, A. Legay, M. L. Pedersen, F. Sher, and A. Wąsowski, "New Results on Abstract Probabilistic Automata," in *ACSD*. Springer, 2011.

[4] L. de Alfaro and T. A. Henzinger, "Interface automata," in *FSE*, 2001.

[5] L. De Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *TACAS*. Springer, 2008.

[6] K. G. Larsen, "Modal specifications," in *AVMS*, ser. LNCS, vol. 407. Springer, 1989, pp. 232–246.

[7] B. Caillaud, B. Delahaye, K. G. Larsen, A. Legay, M. L. Pedersen, and A. Wąsowski, "Compositional design methodology with constraint Markov chains," in *QEST*. IEEE, 2010.

[8] M. Hennessy and R. Milner, "Algebraic laws for nondeterminism and concurrency," *J. ACM*, vol. 32, pp. 137–161, January 1985.

[9] "Strep combest (component-based embedded systems design techniques)," http://www.combest.eu/home/.

[10] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wąsowski, "Timed I/O automata: a complete specification theory for real-time systems," in *HSCC*. ACM, 2010.