Game-based Abstraction and Controller Synthesis for Probabilistic Hybrid Systems

Ernst Moritz Hahn, *Saarland University, Germany*
Gethin Norman, *University of Glasgow, UK*
David Parker, *University of Oxford, UK*
Björn Wachter, *University of Oxford, UK*
Lijun Zhang, *Technical University of Denmark*

***Abstract*—We consider a class of hybrid systems that involve random phenomena, in addition to discrete and continuous behaviour. Examples of such systems include wireless sensing and control applications. We propose and compare two abstraction techniques for this class of models, which yield lower and upper bounds on the optimal probability of reaching a particular class of states. We also demonstrate the applicability of these abstraction techniques to the computation of long-run average reward properties and the synthesis of controllers. The first of the two abstractions yields more precise information, while the second is easier to construct. For the latter, we demonstrate how existing solvers for hybrid systems can be leveraged to perform the computation.**

I. Introduction

Formal analysis of modern applications involves many characteristics, including real-time, stochastic and hybrid dynamics. Often, probabilistic behaviour is abstracted away during the verification of such systems, due to the additional dimension of complexity. This level of abstraction restricts the analysis to qualitative properties. For systems such as wireless sensing and control applications, however, quantitative and performance properties are desired, thus motivating the study of probabilistic hybrid systems.

We consider a class of hybrid systems that involve random phenomena, in addition to discrete and continuous behaviour. We model these systems using *probabilistic hybrid automata* (PHAs), which extend standard hybrid automata with discrete probabilistic choices. In this paper, we tackle the problem of verifying two types of quantitative properties of PHAs: the minimum/maximum probability of reaching a target (e.g. "the maximum probability of the boiler's temperature exceeding its safe limit"); and the minimum/maximum long-run average reward (e.g. "the minimum average power consumption"). We also consider the problem of synthesising controllers for PHAs to achieve such optimum values.

The infinite-state nature of hybrid automata necessitates the use of *abstraction* for their analysis. In [1], an abstraction technique for PHAs was proposed that bounds the maximum probability of reaching a target, by exploiting the construction of finite abstractions from the non-probabilistic setting. The main drawback of this approach is the lack of knowledge about how far away the computed upper bound is from the real value. In this paper, we propose and compare two types of abstraction for PHAs that allow us to give both lower and upper bounds for such quantitative properties.

Our approach is based on the use of n*-player stochastic games*, finite-state automata incorporating decisions made by several distinct players and also random choices. Our motivation for game-based models is twofold. Firstly, in order to specify the problem of *controller synthesis*, we express the semantics of a PHA as a (stochastic) 2-player game: one player represents the controller and the other the environment (as done, for example, in the context of hybrid automata [2] or timed automata [3]). Secondly, we make use of the *game-based abstraction* of [4], which builds abstractions of Markov decision processes as stochastic games by adding an additional player to represent the abstraction. This approach has already been successfully applied to probabilistic timed automata [5], a simple subclass of PHAs.

We first introduce a *game-based abstraction* approach for PHAs. Representing the abstraction as a separate player in the game results in a 3-player stochastic game. We reduce this to a 2-player stochastic game and show that it provides lower and upper bounds on quantitative properties of interest (reachability probabilities and long-run average rewards). We also discuss how such abstractions can be constructed and how they can be *refined* to increase precision.

Next, we introduce a second type of abstraction, which we refer to as an *environment abstraction*. This makes use of an abstraction introduced in [6], [7]. The idea in this case is to, as above, introduce a 3rd player to represent abstraction, but then to make this new player *collaborate* with the player representing the environment in the PHA. The result is an abstraction that is easier to construct in practice and again yields numerical bounds, but gives values that are less precise than those from the game-based abstraction.

We also present techniques to *synthesise* controllers for a PHA, based on an analysis of its abstraction. Finally, we demonstrate how existing solvers for hybrid systems (in this case, PHAVer [8]) can be employed to construct environment abstractions, demonstrating this with results from a small case study. A full version of this paper, with proofs, is available as [9].

Related work. The probabilistic hybrid automata (PHA) model that we use in this paper is closely related to the one proposed by Sproston [10], who shows decidability of model checking for the subclass of rectangular PHAs. We extend the model to a game semantics in order to consider controller synthesis. Recently, there has been renewed interest in analysis techniques for PHAs. The closest to this paper is [1], which applies the probabilistic simulation-based abstraction of [11] to PHAs. We instead extend the abstractions of [4]

and [6], [7], which provide lower and upper bounds, and we additionally consider long-run average properties and controller synthesis. In [12], two approximation techniques for classical hybrid automata (clock approximation and linear phase-portrait approximation) are extended to the probabilistic case. Fränzle *et al.* present a decision procedure [13], [14] for a stochastic logic with applications to bounded model checking of probabilistic hybrid systems.

As mentioned above, for probabilistic timed automata, which are a simple subclass of PHAs, game-based abstraction has already been applied successfully [5]. In [15] a (concurrent) 2-player game extension is presented, but abstractions are not considered. There is also related work on non-probabilistic models. UPPAAL TIGA [3], for example, performs controller synthesis for safety/reachability objectives on timed automata games.

II. Preliminaries

We begin with some notation and background material relating to stochastic games. A *distribution* over a set S is a function $\mu : S \to [0,1]$ such that $\sum_{s \in S} \mu(s) = 1$. We only consider discrete distributions with finite support, that is $|\{s \in S \mid \mu(s) > 0\}| < \infty$. The set of all distributions over S is denoted $\Delta(S)$ and we use $[s_0 \mapsto p_0, \ldots, s_n \mapsto p_n]$ to denote the distribution that chooses s with probability $\sum_{s_i = s} p_i$.

Definition 1. *An n-player* stochastic game *(SG) is a tuple $\mathcal{G} = (P, S, \langle S_p \rangle_{p \in P}, Init, Steps)$ where:*

- *P is a list of n* players*;*
- *S is a (possibly uncountable) non-empty set of* states*, split into disjoint subsets S_p for each player $p \in P$;*
- *$Init \subseteq S$ is a set of* initial states*;*
- *$Steps : S \to 2^{\Delta(S)}$ is a* probabilistic transition function*.*

We use *turn-based* stochastic games, in which each state s is under the control of some player p (i.e. $s \in S_p$). Player p selects a distribution $\mu \in Steps(s)$ and the successor state is then determined randomly according to μ. A *play* of a game $\mathcal{G}$ is a finite or infinite sequence $s_0 \xrightarrow{\mu_0} s_1 \xrightarrow{\mu_1} s_2 \xrightarrow{\mu_2} \cdots$ such that $\mu_i \in Steps(s_i)$ and $\mu_i(s_{i+1}) > 0$ for $i \geq 0$. We denote the set of infinite (finite) plays by $Plays_\omega$ ($Plays_*$) and the last state of a finite path σ by $last(\sigma)$. For $S' \subseteq S$, we let $Plays_*^{S'} \stackrel{\text{def}}{=} \{\sigma \in Plays_* \mid last(\sigma) \in S'\}$. A state s is *reachable* if there exists a finite play starting in an initial state and ending in s.

A *strategy* for a player $p \in P$ is a mapping $\pi_p : Plays_*^{S_p} \to \Delta(\Delta(S))$ such that if $\pi_p(\sigma)(\mu) > 0$, then $\mu \in Steps(last(\sigma))$. We denote the set of all strategies for p by Π_p. A strategy $\pi_p \in \Pi_p$ is *deterministic* (pure) if we always have $\pi_p(\sigma) = [\mu \mapsto 1]$ for some $\mu \in \Delta(S)$, and *memoryless* if $\pi_p(\sigma_1) = \pi_p(\sigma_2)$ for all $\sigma_1, \sigma_2 \in Plays_*^{S_p}$ such that $last(\sigma_1) = last(\sigma_2)$. A strategy is *simple* if it is pure and memoryless, and thus can be seen as a mapping $\pi : S_p \to \Delta(S)$.

If $P' \subseteq P$ and $\pi_p \in \Pi_p$ for all $p \in P'$, then the *joint strategy* $\langle \pi_p \rangle_{p \in P'} : Plays_*^{\cup \{S_p \mid p \in P'\}} \to \Delta(S)$ is such that $\langle \pi_p \rangle_{p \in P'}(\sigma) = \pi_p(\sigma)$ when $last(\sigma) \in S_p$. We denote the set of *complete strategies*, i.e. strategies of the form $\langle \pi_p \rangle_{p \in P} : Plays_*^S \to \Delta(S)$, by Π.

A *value function* $\mathrm{val}_\mathcal{G} : \Pi \to (S \to \mathbb{R})$ for a game $\mathcal{G}$ maps complete strategies to assignments of reals to states. Assuming, for convenience, that $P = \{1, \ldots, n\}$ and given objectives $\mathrm{obj}^p \in \{\inf, \sup\}$ for each $p \in P$, the *optimal value* of the function $\mathrm{val}_\mathcal{G}$ in state $s \in S$ is given by:

$$\mathrm{val}_\mathcal{G}^{\langle \mathrm{obj}^p \rangle_{p=1}^n}(s) \stackrel{\text{def}}{=} \mathrm{obj}^1_{\pi_1 \in \Pi_1} \ldots \mathrm{obj}^n_{\pi_n \in \Pi_n} \mathrm{val}_\mathcal{G}(\langle \pi_i \rangle_{i=1}^n)(s).$$

A complete strategy π on game $\mathcal{G}$ together with an initial state induces a probability space over the set of infinite plays. We can then define the stochastic processes X_i^π and Y_i^π corresponding to the ith state and distribution of the play, respectively, when following strategy π [16]. For a set of target ("final") states $F \subseteq S$, the *reachability probability* $\mathrm{val}_{\mathcal{G},F} : \Pi \to (S \to \mathbb{R})$ is given by:

$$\mathrm{val}_{\mathcal{G},F}(\pi)(s) \stackrel{\text{def}}{=} \mathbf{P}[\exists i . X_i^\pi \in F \mid X_0^\pi = s] .$$

Given a pair of cost and reward functions $cost, rew : (S \times \Delta(S)) \to \mathbb{R}_{\geq 0}$, the *fractional long-run average* value $\mathrm{val}_{\mathcal{G},cost,rew}$, i.e. the average reward per cost in the long-run, is given by:

$$\mathrm{val}_{\mathcal{G},cost,rew}(\pi)(s) \stackrel{\text{def}}{=} \mathbf{E}\left[\lim_{n \to \infty} \frac{\sum_{i=0}^n rew(X_i^\pi, Y_i^\pi)}{\sum_{i=0}^n cost(X_i^\pi, Y_i^\pi)} \,\middle|\, X_0^\pi = s \right]$$

where, within the limit, we map undefined values to 0.

We can reduce n-player games to two or one player games if we subsume players with the same objective, i.e. either $\inf$ or $\sup$, by building the union of their state sets [17]. Furthermore, it is known [18] that the optimal values for reachability objectives on these games exist and can be obtained by a single simple strategy for all $s \in S$ (allowing us to replace $\inf$ and $\sup$ with $\min$ and $\max$). If not mentioned otherwise, we thus assume that all strategies are simple. We can also swap the positions of $\inf$ and $\sup$ in these formulae. Optimal reachability probabilities (and corresponding strategies) can be computed efficiently using, for example, value iteration [19].

To the best of our knowledge, there are currently no algorithms to solve fractional long-run average objectives for stochastic 2-player games. There are however approaches [18] to solve the problem for games in which each step has the same cost. Also, there exist algorithms [20], [21] to compute fractional long-run average values for one-player games, that is Markov decision processes. We are thus confident that the problem can be solved by an extension of existing methods.

III. Probabilistic Hybrid Automata

In this section, we describe our high-level modelling mechanism, probabilistic hybrid automata, and its semantics.

Definition 2. *A* probabilistic hybrid automaton *(PHA) is a tuple* $\mathcal{H} = (M, k, \overline{m}, \langle Post_m \rangle_{m \in M}, C)$ *where:*

- M *is a finite set of modes and* k *is the dimensionality of the continuous variables;*
- $\overline{m} \in M$ *specifies an* initial mode*;*
- $Post_m : \mathbb{R}^k \times \mathbb{R}_{>0} \to 2^{\mathbb{R}^k}$ *is a* constraint *for mode* m*;*
- C *is a finite set of* probabilistic guarded commands *of the form* $g \to (p_1 : u_1 + \cdots + p_n : u_n)$ *where* $g \subseteq M \times \mathbb{R}^k$ *is a* guard*,* $p_i > 0$ *for* $0 \leq i \leq n$*,* $\sum_{i=1}^{n} p_i = 1$ *and* $u_i : M \times \mathbb{R}^k \to 2^{M \times \mathbb{R}^k}$ *is an* update *such that* $u_i(s) \neq \emptyset$ *for* $s \in g$ *and* $1 \leq i \leq n$*.*

The discrete part of a PHA $\mathcal{H}$ is captured by its modes M and the continuous part by a set of k real-valued variables. The dynamics of $\mathcal{H}$ is represented by its flow constraints $\langle Post_m \rangle_{m \in M}$ and set of commands C. The latter describes *discrete* transitions, which are probabilistic. A probabilistic guarded command $c \in C$ comprises a guard g, which determines whether c can be performed in the current state, and n probability-update pairs $p_i : u_i$, which induces a distribution over the updates. Updates u_i are nondeterministic, giving a set of possible successors, dependent on the current state.

We assume that $\mathcal{H}$ has a single initial mode, but this can easily be generalised to either a set or a distribution. Notice also that components such as guards and updates in the definition above are described as sets. In practice (as in existing tools for non-probabilistic hybrid automata such as PHAVER [8]), these will be described by finite representations like polyhedra. Throughout the paper, we will identify these representations with the sets they denote.

The flow constraint operators $Post_m : \mathbb{R}^k \times \mathbb{R}_{>0} \to 2^{\mathbb{R}^k}$ are similar to the notion of Henzinger [22] and describe the possible evolution of the continuous variables. For current mode m and variable values v, $Post_m(v, t)$ gives the values of the continuous variables that may be reached by letting t time units elapse. If $Post_m(v, t) = \emptyset$, then t units of time flow is not possible, i.e. a command must be executed before t time units pass. This operator is often described by differential (in)equations together with a predicate over legal states, but our method also allows for other description methods. We impose the following requirements.

Assumption 1. *The flow constraints* $Post_m$ *of a PHA must satisfy, for any* $m \in M$ *and* $v \in \mathbb{R}^k$*:*

- $Post_m(v, t) = \cup \{ Post_m(v', t - t') \mid v' \in Post_m(v, t') \}$ *for all* $t > t' > 0$*;*
- *there exists* $t > 0$ *such that* $Post_m(v, t) = \emptyset$*.*

The first requirement is naturally fulfilled by the usual notions of time-flow. It means that, if we wait first for time t' and then for $t - t'$, then we can reach the same states as if we had waited for time t. The second point implies that time cannot elapse indefinitely without a jump occurring. This is not a severe restriction; any PHA in which the restriction does not hold can easily be transformed into one where it does, while maintaining the properties considered here.

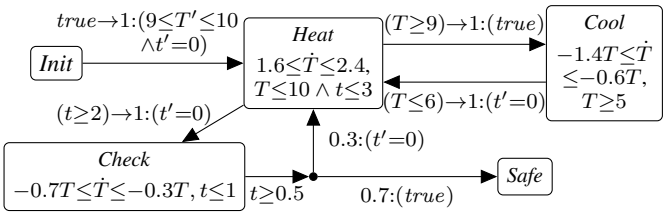

Figure 1. Thermostat example

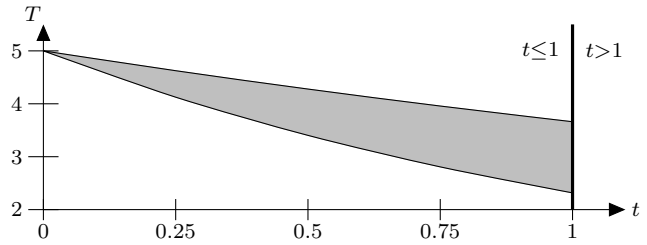

Figure 2. $Post_{Check}$, for initial values $t = 0$, $T = 5$

Example 1. Consider the example PHA shown in Figure 1, which models a thermostat. There are five modes: $M = \{Init, Cool, Heat, Check, Safe\}$ where $Init$ is the initial mode. Modes $Cool$ and $Heat$ implement cooling and heating functions. We assume that the thermostat is broken, which can only be detected when in mode $Check$. In this case, the failure is detected with probability 0.7 (whereby the thermostat shuts down and enters the mode $Safe$), while with probability 0.3 execution continues.

There are two continuous variables, t and T, where T represents the temperature and t the time since entering a mode (in each mode $\dot{t} = 1$). Commands are described on arrows in Figure 1. For instance, the only command available in mode $Check$ is $g \to (0.3 : u_1 + 0.7 : u_2)$, where $g = \{Check\} \times [0.5, \infty) \times \mathbb{R}$, $u_1((m, t, T)) = \{(Heat, 0, T)\}$ and $u_2((m, t, T)) = \{(Safe, t, T)\}$.

The flow constraint of each mode is described using differential equations. For instance, for the mode $Check$ we have $Post_{Check}((t, T), t') = \{(t + t', T') \mid f_l(T, t') \leq T' \leq f_u(T, t')\}$ if $t + t' \leq 1$ where $f_l(T, t') = \exp(-0.7t' + \ln(T))$ and $f_u(T, t') = \exp(-0.3t' + \ln(T))$ and equals $\emptyset$ otherwise. For the initial value $(0, 5)$, the behaviour is depicted in Figure 2. The grey area denotes the set of points which can be reached by a timed transition. The axis labelled with t denotes both values of the time which has passed as well as the variable t. The axis T denotes the temperature. Thus, after time 0.25, T can be in a range of about 4.13 to 4.63. ■

We now define the semantics of PHA. In order to distinguish between the choices made by a *controller* of a hybrid system and the nondeterministic behaviour of its *environment*, it is natural to a take a game-theoretic approach [2]. Here, we use 2-player stochastic games. To avoid time-convergence, we fix a minimal time $t_{\min} \in \mathbb{R}_{>0}$ which the

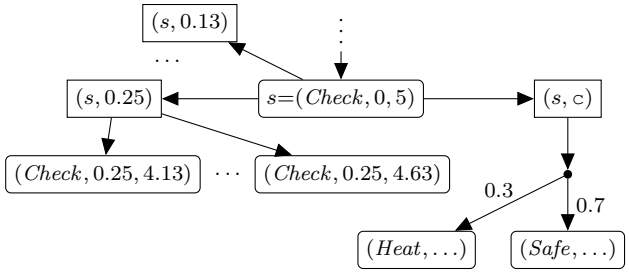

Figure 3. Fragment of the semantics for the PHA in Figure 1

controller may decide to let pass.

Definition 3. *Let* $\mathcal{H}{=}(M, k, \overline{m}, \langle Post_m\rangle_{m\in M}, C)$ *be a PHA. The* semantics *of* $\mathcal{H}$ *is given by the 2-player SG* $[\![\mathcal{H}]\!]{=}(\langle con, env\rangle, S, \langle S_{con}, S_{env}\rangle, \{(\overline{m}, \mathbf{0})\}, Steps)$ *where:*

- con *and* env *are players modelling the* controller *and* environment*, respectively;*
- $S_{con} = M{\times}\mathbb{R}^k$*;*
- $S_{env} = (S_{con}{\times}\mathbb{R}_{\geq t_{\min}}) \cup (S_{con}{\times}C)$*;*
- *the transition function* $Steps$ *is given by:*
  - $Steps(s){=}\{[(s,t){\mapsto}1] \mid t{\geq}t_{\min}{\wedge}Post_m(v,t){\neq}\emptyset\}{\cup}\{[(s,c){\mapsto}1] \mid c{=}(g{\to}\cdots) \in C \wedge s \in g\}$ *for* $s{=}(m,v) \in S_{con}$*;*
  - $Steps((s,t)){=}\{[(m,v'){\mapsto}1] \mid v'{\in}Post_m(v,t)\}$ *for* $(s,t) \in S_{env}$ *where* $s{=}(m,v)$*;*
  - $Steps((s,c)){=}\mathrm{jump}(s,c)$ *for* $(s,c) \in S_{env}$ *where for command* $c{=}g{\to}(p_1{:}u_1{+}\cdots{+}p_n{:}u_n)$ *we have* $[s_1{\mapsto}p_1, \ldots, s_n{\mapsto}p_n] \in \mathrm{jump}(s,c)$ *if and only if* $s \in g$ *and* $s_i \in u_i(s)$ *for all* $1{\leq}i{\leq}n$*.*

States in the set S_{con} of $[\![\mathcal{H}]\!]$ represent the possible configurations $(m, v) \in M{\times}\mathbb{R}^k$ of the PHA. We call $\mathbb{R}_{\geq 0}{\cup}C$ the set of *actions* of the controller player. In the semantics of a PHA, the controller player first chooses an action (i.e. either an amount of time t to pass or a command c to be taken), resulting in a transition to a state of S_{env}. Then, the environment player chooses either the flow of trajectories during the time t or the exact effect of command c, respectively. We assume that every reachable state has at least one possible successor.

The two players may have different goals. For instance, given a set of "unsafe" states, the controller would try to minimise the probability of reaching the set, while the environment tries to maximise. In another scenario, given a set of "good" systems states, the roles would be reversed.

Example 2. We depict part of the semantics for the PHA of our running example in Figure 3. Here, rounded boxes are controller states and rectangles are environment states. Assume we are in mode $Check$ with $t{=}0$ and $T{=}5$. The controller can decide to wait for at most 1 time unit, or execute the available command. If the controller waits for 0.25 time units, according to the $Post$ operator, the new value for t is fixed, but the environment can choose a value for T between 4.13 and 4.63. If the controller decides to execute the available command, there is only one choice for the environment, resulting in a probabilistic choice between the modes $Heat$ and $Safe$. ■

IV. Game-based Abstraction for PHAs

In this section, we define *game-based abstraction* techniques for PHAs. To do so, we extend the approach of [5] for abstracting probabilistic timed automata using the game-based abstraction principles proposed originally in [4]. This section focuses on probabilistic reachability properties. The case for long-run average rewards is covered in Section VII.

We will fix a PHA $\mathcal{H}{=}(M, k, \overline{m}, \langle Post_m\rangle_{m\in M}, C)$, and its semantics $[\![\mathcal{H}]\!]{=}(\langle con, env\rangle, S, \langle S_{con}, S_{env}\rangle, \{(\overline{m}, \mathbf{0})\}, Steps)$. For simplicity, we consider the optimal probability of reaching a target *mode* m_F, and let $F \stackrel{\text{def}}{=} \{m_F\}{\times}\mathbb{R}^k$. In the first part of this section, we describe how to define and construct a game-based abstraction. In the latter part, we discuss how to refine such abstractions to increase precision.

A. Abstract states and transitions

An *abstract state* of PHA $\mathcal{H}$ is a pair of the form $(m, \zeta) \in M{\times}2^{\mathbb{R}^k}$, representing a set of *concrete* states of $\mathcal{H}$. We say that a concrete state $(m, v) \in M{\times}\mathbb{R}^k$ is contained in an abstract state (m', ζ'), written $(m, v) \in (m', \zeta')$, if $m{=}m'$ and $v \in \zeta$. We require two abstract *predecessor* operations, which extend those for timed automata [23]. For a set $\mathcal{A}'{=}\{(m_i, \zeta_i)\}_{i=1}^k$ of abstract states and command $c{=}g{\to}(p_1{:}u_1{+}\cdots{+}p_n{:}u_n)$:

- $\mathsf{tpre}(\mathcal{A}') \stackrel{\text{def}}{=} (m, \{v' \mid \exists t{\geq}t_{\min}.\, (Post_m(v',t){\subseteq}\cup_{i=1}^k \zeta_i \wedge \forall 1{\leq}i{\leq}k.\, (Post_m(v',t){\cap}\zeta_i{\neq}\emptyset))\})$ is the *time predecessor* of $\mathcal{A}'$ when $m_i{=}m$ for all $1{\leq}i{\leq}k$ and is undefined otherwise;
- $\mathsf{dpre}[m, c, u_i](\mathcal{A}') \stackrel{\text{def}}{=} (m, \{v \mid (m,v) \in g \wedge u_i(m,v) \subseteq \cup\mathcal{A}' \wedge \forall \mathsf{z} \in \mathcal{A}'.\, (u_i(m,v) \cap \mathsf{z}){\neq}\emptyset\})$ is the *discrete predecessor* of $\mathcal{A}'$ with respect to predecessor mode m, command c and update u_i.

To build an abstraction of $\mathcal{H}$, we assume that we have a *covering* of its concrete states by abstract states.

Definition 4. *An* abstract state space *of* $\mathcal{H}$ *is a set* $\mathcal{A}{=}\{\mathsf{z}_1, \ldots, \mathsf{z}_q\}$ *where* $\mathsf{z}_i{=}(m_i, \zeta_i) \in M{\times}2^{\mathbb{R}^k}$ *and* $\cup\{\zeta \mid (m,\zeta) \in \mathcal{A}\}{=}\mathbb{R}^k$ *for all* $m \in M$*.*

For presentational simplicity, we assume that the abstract states cover the whole state space. In practice, most of the unreachable states will not be included in the abstraction. Notice that a concrete state (m, v) must be contained in at least one abstract state, but can be contained in several. An *abstract mapping* α chooses one such abstract state, formally, it is a function $\alpha : S{\to}\mathcal{A}$ such that $s \in \alpha(s)$ for all $s \in S$. Using a reasonable representation (e.g. [8]), each abstract state can represent infinitely many concrete states with a finite amount of memory.

Example 3. Consider again the thermostat example of Figure 1. Assume that $\mathsf{z}_1{=}(Init, \mathbb{R}^2)$, $\mathsf{z}_2{=}(Heat, \mathbb{R}^2)$,

$\mathsf{z}_3{=}(\mathit{Cool},\mathbb{R}^2)$, $\mathsf{z}_4{=}(\mathit{Safe},\mathbb{R}^2)$, $\mathsf{z}_5{=}(\mathit{Check},(-\infty,0.4]{\times}\mathbb{R})$, $\mathsf{z}_6{=}(\mathit{Check},[0.3,0.9]{\times}\mathbb{R})$ and $\mathsf{z}_7{=}(\mathit{Check},[0.8,\infty){\times}\mathbb{R})$, then $\mathcal{A}{=}\{\mathsf{z}_1,\ldots,\mathsf{z}_7\}$ forms an abstract state space. ■

Next we introduce the concept of *abstract transitions* [5]. An abstract transition of $\mathcal{H}$ with respect to the abstract state space $\mathcal{A}$ takes the form:

$$\theta = \big(\mathsf{z}, a, \langle(p_i,\mathcal{A}_i)\rangle_{i=1}^n\big) \in \mathcal{A}\times(\mathsf{C}\cup\{\mathtt{time}\})\times([0,1]{\times}2^{\mathcal{A}})^+$$

where, if $a{=}\mathtt{time}$, then $n{=}1$ and $p_1{=}1$, while if $a\in\mathsf{C}$, then a is of the form $g\to(p_1{:}u_1{+}\cdots{+}p_n{:}u_n)$.

Intuitively, if $a{=}\mathtt{time}$, then there exists a concrete state $s{\in}\mathsf{z}$ for which it is possible for the controller to let time elapse (greater or equal to $t_{\min}$) in s such that the subsequent environment's choice is encoded by the set of abstract states $\mathcal{A}_1$. On the other hand, if $a{=}\mathsf{c}$, then there exists $s{\in}\mathsf{z}$ such that, when the controller performs the command $\mathsf{c} = g{\to}(p_1{:}u_1{+}\cdots{+}p_n{:}u_n)$ in s and u_i is chosen (with probability p_i), the environment's choice is encoded by $\mathcal{A}_i$.

Formally, we capture this idea with the notion of *validity* for abstract transitions. More precisely, for abstract transition $\theta{=}((m,\zeta),a,\langle(p_i,\mathcal{A}_i)\rangle_{i=1}^n)$, we define the set of valid variable assignments $\mathit{valid}(\theta)\subseteq\mathbb{R}^k$ such that $v\in\mathit{valid}(\theta)$ if and only if $v\in\zeta$ and the following conditions are satisfied:

- if $a{=}\mathtt{time}$, then there exists $t{\geq}t_{\min}$ and abstract mapping $\alpha : S{\to}\mathcal{A}$ such that $\mathcal{A}_1{=}\{\alpha((m,v'))\,|\,v'\in \mathit{Post}_m(v,t)\}$;
- if $a{=}g{\to}(p_1{:}u_1{+}\cdots{+}p_n{:}u_n)$, then $(m,v)\in g$ and for any $1{\leq}i{\leq}n$ we have $\mathcal{A}_i{=}\{\alpha_i(s_i)\,|\,s_i\in u_i((m,v))\}$ for some abstract mapping $\alpha_i : S{\to}\mathcal{A}$.

We say that θ is valid if $\mathit{valid}(\theta)\neq\emptyset$. Using the symbolic predecessor operations given above, the set $\mathit{valid}(\theta)$ can be computed as follows:

- $\mathit{valid}(\theta)=\zeta\cap\zeta'$ if $a{=}\mathtt{time}$ and $\mathsf{tpre}(\mathcal{A}_1){=}(m,\zeta')$;
- $\mathit{valid}(\theta)=\zeta\cap(\cap_{i=1}^n\zeta_i)$ if $a{=}(g{\to}p_1{:}u_1{+}\cdots{+}p_n{:}u_n)$ and $\mathsf{dpre}[m,\mathsf{c},u_i](\mathcal{A}_i){=}(m,\zeta_i)$ for $1{\leq}i{\leq}n$.

B. Constructing and analysing the abstraction

Next, we define the notion of *abstract graphs* for $\mathcal{H}$.

Definition 5. *An* abstract graph $(\mathcal{A},\mathcal{R})$ *for $\mathcal{H}$ comprises an abstract state space $\mathcal{A}$ and a set of* valid *abstract transitions $\mathcal{R}\subseteq\mathcal{A}{\times}(\mathsf{C}\cup\{\mathtt{time}\}){\times}([0,1]{\times}2^{\mathcal{A}})^+$ such that, for any $\mathsf{z}\in\mathcal{A}$ and $s{=}(m,v)\in\mathsf{z}$:*

- *if $t{\geq}t_{\min}$ and $\mathit{Post}_m(v,t){\neq}\emptyset$, then $\mathcal{R}$ contains an abstract transition $\theta{=}(\mathsf{z},\mathtt{time},\langle(1,\mathcal{A}_1)\rangle)$ such that $\mathcal{A}_1{=}\{\alpha((m,v'))\,|\,v'\in\mathit{Post}_m(v,t)\}$ for some abstract mapping $\alpha : S{\to}\mathcal{A}$;*
- *if $\mathsf{c}=g{\to}(p_1{:}u_1+\cdots+p_n{:}u_n)\in\mathsf{C}$ and $s\in g$, then $\mathcal{R}$ contains an abstract transition $\theta{=}(\mathsf{z},\mathsf{c},\langle(p_i,\mathcal{A}_i)\rangle_{i=1}^n)$ such that $v\in\mathit{valid}(\theta)$.*

An abstract transition $\theta{=}(\mathsf{z},a,\langle(p_i,\mathcal{A}_i)\rangle_{i=1}^n)$ induces the following probability distribution λ_θ over $2^{\mathcal{A}}$ and set of probability distributions Λ_θ over $\mathcal{A}$:

$$\lambda_\theta \stackrel{\text{def}}{=} [\mathcal{A}_1{\mapsto}p_1,\ldots,\mathcal{A}_n{\mapsto}p_n]$$
$$\Lambda_\theta \stackrel{\text{def}}{=} \{\,[\mathsf{z}_1{\mapsto}p_1,\ldots,\mathsf{z}_n{\mapsto}p_n]\,|\,\mathsf{z}_i\in\mathcal{A}_i \text{ for all } 1{\leq}i{\leq}n\,\}$$

We also extend the notion of *validity* for abstract transitions to *sets* of abstract transitions with the same source. For abstract state $\mathsf{z}\in\mathcal{A}$, we let $\mathcal{R}(\mathsf{z})$ denote the set of abstract transitions in $\mathcal{R}$ with source z. Then, for a set of abstract transitions $\Theta\subseteq\mathcal{R}(\mathsf{z})$, we define:

$$\mathit{valid}(\Theta) \stackrel{\text{def}}{=} (\cap_{\theta\in\Theta}\mathit{valid}(\theta))\setminus\big(\cup_{\theta\in\mathcal{R}(\mathsf{z})\setminus\Theta}\mathit{valid}(\theta)\big)$$

and say that Θ is *valid* if $\mathit{valid}(\Theta)$ is non-empty. It follows that Θ is valid if and only if there exists a state $s\in\mathsf{z}$ such that it is possible to perform a transition encoded by each abstract transition $\theta\in\Theta$, but it is not possible to perform a transition encoded by any other abstract transition of $\mathcal{R}(\mathsf{z})$.

Finally, we can define game-based abstraction for PHAs.

Definition 6. *Let $(\mathcal{A},\mathcal{R})$ be an abstract graph of $\mathcal{H}$. The* game-based abstraction $\mathsf{game}(\mathcal{H})$ *of $\mathcal{H}$ with respect to $(\mathcal{A},\mathcal{R})$ is the 3-player stochastic game $(\langle \mathit{abs},\mathit{con},\mathit{env}\rangle, S, \langle S_{\mathit{abs}},S_{\mathit{con}},S_{\mathit{env}}\rangle, \mathit{Init}, \mathit{Steps})$ where:*

- $S_{\mathit{abs}}{=}\mathcal{A}$;
- $S_{\mathit{con}}{=}\{\Theta\subseteq\mathcal{R}\,|\,\Theta \textit{ is valid}\}$;
- $S_{\mathit{env}}{=}2^{\mathcal{A}}$;
- $\mathit{Init}=\{\mathsf{z}\in\mathcal{A}\mid(\overline{m},\mathbf{0})\in\mathsf{z}\}$
- $\mathit{Steps}(\mathsf{z}){=}\{[\Theta{\mapsto}1]\,|\,\Theta{\subseteq}\mathcal{R}(\mathsf{z}) \textit{ is valid}\}$ *for* $\mathsf{z}\in S_{\mathit{abs}}$;
- $\mathit{Steps}(\Theta){=}\{\lambda_\theta\,|\,\theta\in\Theta\}$ *for* $\Theta\in S_{\mathit{con}}$;
- $\mathit{Steps}(\mathcal{A}'){=}\{[\mathsf{z}{\mapsto}1]\,|\,\mathsf{z}\in\mathcal{A}'\}$ *for* $\mathcal{A}'\in S_{\mathit{env}}$.

The basic idea is that the abstraction process is a *game*, with an additional player (abs) representing the abstraction. By allowing abs to either minimise or maximise the probability of reaching the target, we will obtain lower and upper bounds, respectively, on the actual optimal probability. A coarser abstraction will give a greater degree of control to player abs, resulting in less precise bounds.

Intuitively, the three players (abs, con and env) take consecutive turns, operating as follows. From an abstract state $\mathsf{z}\in\mathcal{A}$, player abs chooses a concrete state[1] from z. Next, the controller (player con) picks one of the available actions. The outcome of this is a (probabilistic) transition to a *set* of abstract states. Finally, the environment (player env) chooses which successor abstract state is actually taken.

Example 4. In Figure 4 we give a part of the game-based abstraction for our running example. Circles denote states of the abstraction player, rounded boxes are controller states and rectangles are environment states. In z_6, there are states in which the command from mode *Check* is enabled, and those where it is not; in both cases, time can also pass. When the command *is* enabled, the abstract controller player can

[1]This is the intuition: in fact, abs chooses a set of abstract transitions Θ that represents multiple concrete states that have an equivalent set of actions available under the current abstraction.

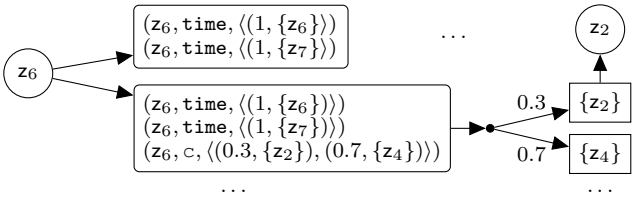

Figure 4. Fragment of the game-based abstraction for the thermostat

choose to execute this command. Then, with probability 0.3, the transition to the environment player state consisting of the singleton set $\{z_2\}$ is taken. Because this set is a singleton, the environment player may only choose to move to z_2. ■

Finally, we show that the SG $\mathsf{game}(\mathcal{H})$ yields lower and upper bounds on reachability probabilities in the original PHA $\mathcal{H}$. Formally, this is captured by the following result.

Proposition 1. *Let $(\mathcal{A}, \mathcal{R})$ be an abstract graph for $\mathcal{H}$ and $\mathrm{obj}_c, \mathrm{obj}_e \in \{\inf, \sup\}$. If $\mathsf{game}(\mathcal{H})$ is the game-based abstraction of $\mathcal{H}$ with respect to $(\mathcal{A}, \mathcal{R})$, and $\mathsf{z}_F{=}(m_F, \mathbb{R}^k)$ is the target abstract state, then:*

$$\mathrm{val}^{\inf,\mathrm{obj}_c,\mathrm{obj}_e}_{\mathsf{game}(\mathcal{H}),\{\mathsf{z}_F\}}(\mathsf{z}) \leq \mathrm{val}^{\mathrm{obj}_c,\mathrm{obj}_e}_{[\![\mathcal{H}]\!],F}(s) \leq \mathrm{val}^{\sup,\mathrm{obj}_c,\mathrm{obj}_e}_{\mathsf{game}(\mathcal{H}),\{\mathsf{z}_F\}}(\mathsf{z})$$

for all states s of $[\![\mathcal{H}]\!]$ and $\mathsf{z} \in \mathcal{A}$ such that $s \in \mathsf{z}$.

Essentially, this states that, for any combination of objectives $\mathrm{obj}_c, \mathrm{obj}_e \in \{\inf, \sup\}$ for the controller and environment of the original PHA $\mathcal{H}$, lower and upper bounds on the corresponding optimal value for $\mathcal{H}$ are obtained from optimal values of $\mathsf{game}(\mathcal{H})$. For both players, we can also derive a concrete strategy which guarantees these bounds.

C. Refinement

Lastly in this section, we extend our game-based abstraction approach with *refinement*. As done in [5] for probabilistic timed automata, this provides a means to automatically generate abstractions, in the style of "counterexample-guided abstraction refinement". This works by taking an initially coarse abstraction and iteratively refining until it is precise enough to yield useful verification results. Crucial to this approach are the lower/upper bounds from the game-based abstraction, the difference between which which give a quantitative measure of the abstraction's precision.

Refinement is defined in terms of an abstract graph $(\mathcal{A}, \mathcal{R})$: it splits one or more abstract states in $\mathcal{A}$ and then modifies the abstract transitions of $\mathcal{R}$ accordingly. This process is guided by the analysis of the corresponding stochastic game abstraction, i.e. the bounds for the value of interest and the abstraction player strategies that attain these bounds.

We now outline the refinement of a single abstract state (m, ζ) for which the bounds differ and for which distinct simple strategies of the abstraction player yield each bound. Assuming the bounds differ in some state, then it follows from the results of [4] that such an abstract state exists. By construction, a simple strategy for the abstraction player chooses, in abstract state (m, ζ), a valid set of abstract transitions from $\mathcal{R}((m, \zeta))$. Let $\Theta_{lb}, \Theta_{ub} \subseteq \mathcal{R}((m, \zeta))$ be distinct lower and upper bound respectively. Since the validity conditions for Θ_{lb} and Θ_{ub} give precisely the variable assignments in ζ for which the corresponding transitions of $[\![\mathcal{H}]\!]$ are possible, we split (m, ζ) into:

- $(m, \mathit{valid}(\Theta_{lb}))$;
- $(m, \mathit{valid}(\Theta_{ub}))$;
- $(m, \zeta \setminus (\mathit{valid}(\Theta_{lb}) \cup \mathit{valid}(\Theta_{ub})))$.

By construction, $\mathit{valid}(\Theta_{lb})$ and $\mathit{valid}(\Theta_{ub})$ are both non-empty. Furthermore, since $\Theta_{lb} \neq \Theta_{ub}$, from the definition of validity, we have $\mathit{valid}(\Theta_{lb}) \cap \mathit{valid}(\Theta_{ub}) = \emptyset$, and hence the split of (m, ζ) produces a strict refinement of $\mathcal{A}$.

```
1  ζ_lb := valid(Θ_lb)
2  ζ_ub := valid(Θ_ub)
3  A^new := {(m, ζ_lb), (m, ζ_ub), (m, ζ\(ζ_lb∪ζ_ub))} \ {∅}
4  A^ref := (A \ {(m, ζ)}) ⊎ A^new
5  R^ref := ∅
6  for θ = (z, a, ⟨(p_1, A_1), ..., (p_n, A_n)⟩) ∈ R
7    if (m, ζ) ∉ {z} ∪ A_1 ∪ ··· ∪ A_n then
8      R^ref := R^ref ∪ {θ}
9    else Θ^new := {(z', a, ⟨A'_1, ..., A'_n⟩)} where
10       if z=(m, ζ) then z' ∈ A^new
11       else z'=z
12       if (m, ζ) ∈ A_i then
13       A'_i=(A_i\(m, ζ)) ∪ A' and A'(≠∅) ⊆ A^ref
14       else A'_i = A_i
15     for θ^new ∈ Θ^new such that valid(θ^new) ≠ ∅
16       R^ref := R^ref ∪ {θ^new}
17 return (A^ref, R^ref)
```

Figure 5. Algorithm $\mathsf{Refine}(\mathcal{A}, \mathcal{R}, (m, \zeta), \Theta_{lb}, \Theta_{ub})$ to refine abstract state (m, ζ) in abstract graph $(\mathcal{A}, \mathcal{R})$

Figure 5 presents the complete refinement algorithm. It first refines $\mathcal{A}$ (lines 1–4) as described above. The set of abstract transitions $\mathcal{R}$ is then updated (lines 5–16). The result is a new abstract graph, for which the corresponding stochastic game is a refined abstraction of the PHA, as shown by the following proposition.

Proposition 2. *Let $(\mathcal{A}, \mathcal{R})$ be an abstract graph for $\mathcal{H}$. If $(\mathcal{A}^{ref}, \mathcal{R}^{ref})$ is the result of applying algorithm* Refine *(see Figure 5) to $(\mathcal{A}, \mathcal{R})$, then $(\mathcal{A}^{ref}, \mathcal{R}^{ref})$ is an abstract graph for $\mathcal{H}$. Furthermore, if $\mathsf{game}(\mathcal{H})$ and $\mathsf{game}(\mathcal{H})^{ref}$ are the game-based abstractions with respect to $(\mathcal{A}, \mathcal{R})$ and $(\mathcal{A}^{ref}, \mathcal{R}^{ref})$ respectively and $\mathrm{obj}_c, \mathrm{obj}_e \in \{\inf, \sup\}$, then:*

$$\begin{aligned} \mathrm{val}^{\inf,\mathrm{obj}_c,\mathrm{obj}_e}_{\mathsf{game}(\mathcal{H}),\{\mathsf{z}_F\}}((m,\zeta)) &\leq \mathrm{val}^{\inf,\mathrm{obj}_c,\mathrm{obj}_e}_{\mathsf{game}(\mathcal{H})^{ref},\{\mathsf{z}_F\}}((m,\zeta^{ref})) \\ \mathrm{val}^{\sup,\mathrm{obj}_c,\mathrm{obj}_e}_{\mathsf{game}(\mathcal{H})^{ref},\{\mathsf{z}_F\}}((m,\zeta^{ref})) &\leq \mathrm{val}^{\sup,\mathrm{obj}_c,\mathrm{obj}_e}_{\mathsf{game}(\mathcal{H}),\{\mathsf{z}_F\}}((m,\zeta)) \end{aligned}$$

for all $(m, \zeta) \in \mathcal{A}$ and $(m, \zeta^{ref}) \in \mathcal{A}^{ref}$ such that $\zeta^{ref} \subseteq \zeta$.

Elsewhere, for example in [5], refinement of game-based abstractions is used in a refinement loop to automatically construct suitable abstractions. For probabilistic timed au-

tomata [5], this loop is guaranteed to terminate thanks to the existence of a finite underlying region graph. In the context of hybrid systems, no such guarantee can be obtained. In fact, model checking for much simpler subclasses, such as (non-probabilistic) linear hybrid automata, is already known to be undecidable [24].

Implementing game-based abstractions requires the construction of abstract graphs and, in particular, the computation of valid sets through predecessor operations. Current tools for the verification of non-probabilistic hybrid automata can only compute approximations of these sets. This motivates the weaker abstraction considered next.

V. Environment Abstraction for PHAs

In this section, we define a second type of abstraction for PHAs called *environment abstraction*, which is coarser than the one introduced in Section IV but, as already mentioned, is easier to implement on top of existing hybrid systems solvers. As for the game-based abstraction, we fix a PHA $\mathcal{H}{=}(M, k, \overline{m}, \langle Post_m\rangle_{m\in M}, \subset)$, abstract state space $\mathcal{A}$ of $\mathcal{H}$ and target mode m_F. We consider the optimal probability of reaching the set of states $F = \{m_F\}{\times}\mathbb{R}^k$.

The environment abstraction is also defined in terms of an abstract graph (AG); see Definition 5. However, it is based on a modified notion of validity and so, to differentiate between the different types of abstract graphs, we will denote this new class by *environment AG* and the one introduced in Section IV by *game AG*. Formally, given a maximum time duration $t_{\max}{\geq}t_{\min}$, the modified version of validity is defined as follows: transition $\theta{=}(\mathsf{z}, a, \langle(p_i, \mathcal{A}_i)\rangle_{i=1}^n)$ is *valid* if:

- $a{\in}\texttt{time}$ and $\{\alpha((m, v')) \mid (m, v){\in}\mathsf{z} \wedge t_{\min}{\leq}t{\leq}t_{\max} \wedge v'{\in}Post_m(v, t)\} \subseteq \mathcal{A}_1$ for some abstract mapping $\alpha : S{\to}\mathcal{A}$;
- $a = g{\to}(p_1{:}u_1{+}\cdots{+}p_n{:}u_n)$ and for any $1{\leq}i{\leq}n$ there exists $s \in \mathsf{z} \cap g$ such that $\{\alpha(s_i) \mid s_i \in u_i(s)\} \subseteq \mathcal{A}_i$ for some abstract mapping $\alpha_i : S{\to}\mathcal{A}$.

In the modified definition, for a timed abstract transition $(\mathsf{z}, \texttt{time}, \langle(1, \mathcal{A}_1)\rangle)$ to be valid we require that all concrete states outside z that can be reached by letting up to time $t_{\max}$ pass in z are contained in an abstract state of $\mathcal{A}_1$. This corresponds to the timed transitions often obtained from hybrid automata tools which correspond to letting time pass up to a certain limit (denoted by $t_{\max}$ here).

On the other hand, in the case of guarded commands, the modified definition of validity is weakened such that we consider each update independently. More precisely, we only require that for each update u_i, there exists a concrete state $s \in \mathsf{z} \cap g$ such that after u_i is performed the environment's choice is encoded by $\mathcal{A}_i$. Again this corresponds to transitions obtained from hybrid automata tools where probabilistic branching is not considered.

Below we define the *environment abstraction*. The intuition behind this is that we again introduce a third player to represent abstraction, but make it *collaborate* with the player representing the environment. This results in a game with just two players, one corresponding to the controller (con), and one to the combined environment/abstraction (abs). The abstraction (represented by player abs) works in similar fashion to the abstraction introduced in [6], [7]. Like in that approach, if an abstract state contains concrete states for which a certain action a cannot be performed, we add extra transitions to a special *sink state* $\bot$. These special transitions can be seen as disadvantageous for the controller.

Definition 7. *Let $(\mathcal{A}, \mathcal{R})$ be an environment AG of PHA $\mathcal{H}$. The* environment abstraction $\mathsf{env}(\mathcal{H})$ *of $\mathcal{H}$ with respect to $(\mathcal{A}, \mathcal{R})$ is the 2-player stochastic game $(\langle con, abs\rangle, S, \langle S_{con}, S_{abs}\rangle, Init, Steps)$ where:*

- $S_{con}{=}\mathcal{A} \cup \{\bot\}$*;*
- $S_{abs}{=}\{(\mathsf{z}, a) \mid \mathsf{z} \in \mathcal{A} \wedge \exists(\mathsf{z}, a, \langle(p_i, \mathcal{A}_i)\rangle_{i=1}^n) \in \mathcal{R}\}$*;*
- $Init = \{\mathsf{z} \in \mathcal{A} \mid (\overline{m}, \mathbf{0}) \in \mathsf{z}\}$
- $Steps(\mathsf{z}){=}\{[(\mathsf{z}, a){\mapsto}1] \mid (\mathsf{z}, a){\in}S_{abs}\}$ *for all $\mathsf{z} \in \mathcal{A}$;*
- $Steps(\bot){=}\{[\bot{\mapsto}1]\}$*;*
- $Steps(\mathsf{z}, a){=}\{\lambda \mid \lambda{\in}\Lambda_\theta \wedge \theta{=}(\mathsf{z}, a, \langle(p_i, \mathcal{A}_i)\rangle_{i=1}^n){\in}\mathcal{R}\} \cup sink_{(\mathsf{z},a)}$ *for all $(\mathsf{z}, a) \in S_{abs}$*

and $sink_{(\mathsf{z},a)}$ equals $\{[\bot{\mapsto}1]\}$ if $a{=}\texttt{time}$ and there exists $(m, v){\in}\mathsf{z}$ such that $Post_m(v, t){=}\emptyset$ for all $t_{\min}{\leq}t{\leq}t_{\max}$ or $a{=}g{\to}(\cdots)$ and $\mathsf{z}\backslash g{\neq}\emptyset$, and equals $\emptyset$ otherwise.

In the environment abstraction, the time duration is chosen by the abstraction rather than the controller player. As the controller can only choose whether to let time pass or to execute a guarded command, the smaller $t_{\max}$ is, the less choices there are for the opposing player. This means that decreasing $t_{\max}$ increases the power of the controller player. From Assumption 1, it follows that we can divide timed transitions in which more than $t_{\max}$ time passes into several timed transitions of time no longer than $t_{\max}$. We can thus simulate timed transitions longer than $t_{\max}$ by a chain of time transitions.

The following proposition states that the environment abstraction yields lower and upper bounds on reachability objectives in the corresponding PHA.

Proposition 3. *If $(\mathcal{A}, \mathcal{R})$ is an environment AG for $\mathcal{H}$ and $\mathsf{env}(\mathcal{H})$ is the environment abstraction with respect to $(\mathcal{A}, \mathcal{R})$, then for $\mathrm{obj} \in \{\inf, \sup\}$:*

$$\begin{aligned}\mathrm{val}^{\inf,\inf}_{\mathsf{env}(\mathcal{H}),\{\mathsf{z}_F,\bot\}}(\mathsf{z}) \leq \quad \mathrm{val}^{\inf,\mathrm{obj}}_{[\![\mathcal{H}]\!],F}(s) \leq \quad \mathrm{val}^{\inf,\sup}_{\mathsf{env}(\mathcal{H}),\{\mathsf{z}_F,\bot\}}(\mathsf{z}),\\ \mathrm{val}^{\sup,\inf}_{\mathsf{env}(\mathcal{H}),\{\mathsf{z}_F\}}(\mathsf{z}) \leq \quad \mathrm{val}^{\sup,\mathrm{obj}}_{[\![\mathcal{H}]\!],F}(s) \leq \quad \mathrm{val}^{\sup,\sup}_{\mathsf{env}(\mathcal{H}),\{\mathsf{z}_F\}}(\mathsf{z})\end{aligned}$$

for all states s of $[\![\mathcal{H}]\!]$ and $\mathsf{z} \in \mathcal{A}$ such that $s \in \mathsf{z}$.

Notice that, in the case where the controller player tries to minimise the reachability probability, we need to include $\bot$ in the reachability set. Otherwise, the controller player could choose an action which is invalid for some states of an abstract state, but advantageous for the controller in the remaining ones. In this case, the second player would either

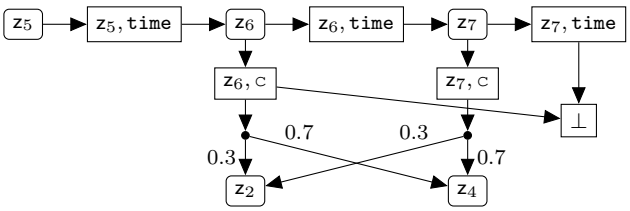

Figure 6. Fragment of the environment abstraction for the thermostat

have to choose an abstract transition which is advantageous for the controller, or choose a transition to $\bot$, which would also be advantageous for the controller if it is not included in the reachability set. For similar reasons, $\bot$ is not included in the case where the probability is being maximised.

Example 5. In Figure 6 we show a part of a possible environment abstraction using the abstract state space of Example 3. In z_5, there is no concrete state which fulfils the guard of the mode *Check*, but in all states it is possible to let time pass. Thus, in $(\mathsf{z}_5, \mathtt{time})$ there is only a transition to z_6 but not to the sink state. In z_6, we can execute the command $\mathtt{c}$ of *Check*, but not for all concrete states, as seen from transitions of $(\mathsf{z}_6, \mathtt{c})$. In all states of z_7 it is possible to execute the command, but there are some states in which a time flow is no longer possible. ■

VI. Controller Synthesis

Next, we consider the problem of *synthesising* a controller for a PHA that makes "optimal" decisions. That is, if the objective is to maximise (minimise) the reachability probability, then the controller takes decisions such that this probability is maximised (minimised) under all possible reactions of the environment. The controller can base its decisions on the current state of the PHA, including the values of the continuous variables. In practice, such a controller would be implemented with the aid of sensors and timers.

We will construct controllers from the strategies of the player *con* in the abstraction of a PHA. The quality of the controller of course depends on the precision of the abstraction computed. Conversely, the complexity of the implemented controller will increase when extracted from a finer abstraction. Due to space limitations, we only discuss how to use environment abstraction to synthesise controllers. We begin by defining a relation between strategies in an environment abstraction and in the concrete semantics.

Definition 8. *Let $\mathcal{H}$ be a PHA with semantics $[\![\mathcal{H}]\!]=(\langle con, env\rangle, S, \langle S_{con}, S_{env}\rangle, \{(\overline{m}, \mathbf{0})\}, Steps)$ and $\mathsf{env}(\mathcal{H})=(\langle con, abs\rangle, S^{\mathsf{abs}}, \langle S^{\mathsf{abs}}_{con}, S^{\mathsf{abs}}_{abs}\rangle, Init^{\mathsf{abs}}, Steps^{\mathsf{abs}})$ an environment abstraction for $\mathcal{H}$ for some environment AG $(\mathcal{A}, \mathcal{R})$ and maximum time bound $t_{\max}$. For an abstract controller $\pi^{\mathsf{abs}} : S^{\mathsf{abs}}_{con} \to \Delta(S^{\mathsf{abs}}_{abs})$, we define a* concretisation *$\pi : S_{con} \to \Delta(S_{env})$ of π^{abs} as follows. For $s=(m, v) \in S_{con}$, we choose an abstract state $\mathsf{z} \in S^{\mathsf{abs}}_{con}$ with $s \in \mathsf{z}$ and if $\pi^{\mathsf{abs}}(\mathsf{z})=[(\mathsf{z}, a)\mapsto 1]$, then*

$$\pi(s)=\begin{cases}[(s,a)\mapsto 1] & \text{if } a \in C\\ [(s,t)\mapsto 1] & \text{if } a = \mathtt{time}\end{cases}$$

where $t=\min\{\sup\{t' \mid Post_m(v,t')\neq\emptyset\}, t_{\max}\}$. However, if there is no such abstract state z which yields a valid choice in state s, we let π choose an arbitrary valid action in s.

Using Definition 8, we can implement a controller function **exec** controlling the PHA: at first, the function checks in which abstract state(s) the current configuration is contained. Depending on the result, it either executes the guarded command given by the abstract controller strategy or, if the abstract controller chooses `time`, it waits as long as possible but no longer than $t_{\max}$. By executing **exec** in an infinite loop, we can control the system such that it fulfils reachability probability bounds that we can compute a priori.

These bounds are defined as follows. Consider an environment abstraction objective $\mathrm{obj}_e \in \{\inf, \sup\}$ for the probability of reaching target set F. For an SG $\mathcal{G}$ and a (simple) strategy π_p for player p in $\mathcal{G}$, let $\mathcal{G}[\pi_p]$ denote the modified SG in which choices in each state $s \in S_p$ have been resolved according to $\pi(s)$. Then, letting $\mathcal{G}_\pi = [\![\mathcal{H}]\!][\pi]$ and $\mathcal{G}^{\mathsf{abs}}_\pi = \mathsf{env}(\mathcal{H})[\pi^{\mathsf{abs}}]$ we have:

$$\begin{aligned}\mathrm{val}^{\inf,\inf}_{\mathcal{G}^{\mathsf{abs}}_\pi,\{\mathsf{z}_F,\bot\}}(\mathsf{z}) \le \quad \mathrm{val}^{\inf,\mathrm{obj}_e}_{\mathcal{G}_\pi,F}(s) \le \mathrm{val}^{\inf,\sup}_{\mathcal{G}^{\mathsf{abs}}_\pi,\{\mathsf{z}_F,\bot\}}(\mathsf{z}),\\ \mathrm{val}^{\sup,\inf}_{\mathcal{G}^{\mathsf{abs}}_\pi,\{\mathsf{z}_F\}}(\mathsf{z}) \le \quad \mathrm{val}^{\sup,\mathrm{obj}_e}_{\mathcal{G}_\pi,F}(s) \le \quad \mathrm{val}^{\sup,\sup}_{\mathcal{G}^{\mathsf{abs}}_\pi,\{\mathsf{z}_F\}}(\mathsf{z}).\end{aligned}$$

The correctness of these bounds can be established in the same manner as those for Proposition 3.

VII. Abstraction for Long-run Average Values

Now, we extend our method to obtain bounds for *long-run average* properties. Rewards will be accumulated by executing commands or letting time elapse in system states, and costs will correspond to the elapsed time. Formally, a *reward structure* for a PHA $\mathcal{H} = (M, k, \overline{m}, \langle Post_m\rangle_{m\in M}, \mathtt{C})$ is a tuple $(\mathtt{R}_\mathtt{C}, \mathtt{R}_\mathtt{T})$ where $\mathtt{R}_\mathtt{C} : \mathtt{C}\to\mathbb{R}$ and $\mathtt{R}_\mathtt{T} : (M\times\mathbb{R}^k)\times\mathbb{R}_{\ge 0} \to \mathbb{R}$. The function $\mathtt{R}_\mathtt{C}$ assigns the reward $\mathtt{R}_\mathtt{C}(\mathtt{c})$ when the command $\mathtt{c} \in \mathtt{C}$ is executed, while $\mathtt{R}_\mathtt{T}$ assigns, when starting in state $(m, v) \in M\times\mathbb{R}^k$, the reward $\mathtt{R}_\mathtt{T}((m, v), t)$ if t time units elapse. We require that $\mathtt{R}_\mathtt{T}((m, v), t) = \mathtt{R}_\mathtt{T}((m, v), t')+\mathtt{R}_\mathtt{T}((m, v'), t-t')$ for all $(m, v) \in M\times\mathbb{R}^k$, $t \in \mathbb{R}_{\ge 0}$, $0\le t'\le t$ and $v' \in Post_m(v, t')$. This means that we obtain the same reward if waiting for the same time, independently of the number of timed steps. A more general formulation of reward assignment is possible, but is omitted for clarity of presentation.

The corresponding reward and cost functions on $[\![\mathcal{H}]\!]$ are such that for $s=(m, v) \in M\times\mathbb{R}^k$, $\mathtt{c} \in \mathtt{C}$ and $t \in \mathbb{R}_{\ge 0}$:

- $rew(s, [(s, \mathtt{c})\mapsto 1]) \stackrel{\text{def}}{=} \mathtt{R}_\mathtt{C}(\mathtt{c})$;
- $rew(s, [(s, t)\mapsto 1] \stackrel{\text{def}}{=} \mathtt{R}_\mathtt{T}(s, t)$;
- $cost(s, [(s, t)\mapsto 1] \stackrel{\text{def}}{=} t$;

and otherwise the functions take value 0. As explained above, the cost function *cost* measures the elapsed time.

To define our abstractions, we consider the modified PHA $\mathcal{H}'=(M, k+2, \overline{m}, \langle Post'_m\rangle_{m\in M}, \subset')$ which adds two additional variables $\mathtt{t},\mathtt{r}\in\mathbb{R}_{\geq 0}$ to track the time and reward accumulated since a jump step. Formally, for $(m,v)\in M\times\mathbb{R}^k$ and $\mathtt{t},\mathtt{r},t\in\mathbb{R}_{\geq 0}$, $Post'_m((v,\mathtt{t},\mathtt{r}),t)$ equals:

$$\{(v', \mathtt{t}+t, \mathtt{r}+\mathtt{R}_{\mathrm{T}}((m,v),t)) \mid v'\in Post_m(v,t)\}$$

and $g'\to(p_1{:}u'_1+\cdots+p_n{:}u'_n)\in\subset'$ if and only if there exists $g\to(p_1{:}u_1+\cdots+p_n{:}u_n)\in\subset$ such that $g'=(g\times\mathbb{R}^2)$ and $u'_i((m,v,\mathtt{t},\mathtt{r}))=u_i((m,v))\times\{(0,0)\}$ for all $(m,v)\in M\times\mathbb{R}^k$, $\mathtt{t},\mathtt{r}\in\mathbb{R}_{\geq 0}$ and $0\leq i\leq n$.

It is straightforward to integrate this change into a hybrid systems solver in which continuous dynamics are described by differential (in)equations and constraints on the states in which time may pass. It suffices to add one variable with constant derivative 1 for the time, and another variable describing $\mathtt{R}_{\mathrm{T}}$. Because of the restrictions on the $Post$ operator from Assumption 1 and since we reset the two new variables to zero each time a command is executed, these values will stay below a certain limit. Thus, if a hybrid systems solver is able to compute a good approximation for the original model, it is likely to be able to do so for the modified version.

We next construct the corresponding reward and cost functions for our abstraction methods. We suppose we have a fixed abstract state space $\mathcal{A}$. For $\mathrm{obj}\in\{\inf,\sup\}$ and $\mathsf{z}\in\mathcal{A}$, let $\mathtt{R}^{\mathrm{obj}}(\mathsf{z})\stackrel{\text{def}}{=}\mathrm{obj}\{\mathtt{r}\mid(s,\mathtt{t},\mathtt{r})\in\mathsf{z}\}$, $\mathtt{T}^{\inf}(\mathsf{z})\stackrel{\text{def}}{=}\sup\{\mathtt{t}\mid(s,\mathtt{t},\mathtt{r})\in\mathsf{z}\}$ and $\mathtt{T}^{\sup}(\mathsf{z})\stackrel{\text{def}}{=}\inf\{\mathtt{t}\mid(s,\mathtt{t},\mathtt{r})\in\mathsf{z}\}$. Notice, since we divide by the cost values (i.e. the elapsed time), we have swapped $\inf$ and $\sup$ in the definition of $\mathtt{T}^{\mathrm{obj}}$. To compute these values we need to examine the abstract states. For instance, in PHAVER abstract states are given as a mode plus a conjunction of linear constraints on the continuous variables. Thus, using linear programming, such values for reward and time can easily be found, even for non-linear dynamics.

We now assign reward structures to the abstract transitions such that the abstraction overapproximates the concrete semantics: if we have a minimising abstract player (environment player in the environment abstraction), the fractional values we obtain will be lower than in the semantics, whereas in the case of a maximising one they will be higher.

For the game based abstraction $\mathsf{game}(\mathcal{H}')$ we introduce reward and cost functions rew^{obj} and $cost^{\mathrm{obj}}$ for $\mathrm{obj}\in\{\inf,\sup\}$ such that, if Θ is valid set of abstract transitions and $\theta=\big(\mathsf{z},\subset,\langle(p_i,\mathcal{A}_i)\rangle_{i=1}^n\big)\in\Theta$, then:

- $rew^{\mathrm{obj}}(\Theta,\lambda_\theta)=\mathtt{R}^{\mathrm{obj}}(\mathsf{z})+\mathtt{R}_{\subset}(\subset)$;
- $cost^{\mathrm{obj}}(\Theta,\lambda_\theta)=\mathtt{T}^{\mathrm{obj}}(\mathsf{z})$

and the functions assign zero to all other state-transition pairs.

On the other hand, for the environment abstraction $\mathsf{env}(\mathcal{H}')$ we construct reward and cost functions $rew^{\mathrm{obj}_{con},\mathrm{obj}_{abs}}$ and $cost^{\mathrm{obj}_{con},\mathrm{obj}_{abs}}$ such that:

- $rew^{\mathrm{obj}_{con},\mathrm{obj}_{abs}}((\mathsf{z},\subset))=\mathtt{R}_{\subset}(\subset)+\mathtt{R}^{\mathrm{obj}_{abs}}(\mathsf{z})$;
- $rew^{\inf,\mathrm{obj}_{abs}}(\bot,\mu)=\infty$;
- $rew^{\sup,\mathrm{obj}_{abs}}(\bot,\mu)=-\infty$;
- $cost^{\mathrm{obj}_{con},\mathrm{obj}_{abs}}((\mathsf{z},\subset))=\mathtt{T}^{\mathrm{obj}_{abs}}(\mathsf{z})$;
- $cost^{\mathrm{obj}_{con},\mathrm{obj}_{abs}}(\bot,\mu)=1$

and zero is assigned to all other state-transition pairs. Notice that, as in the case of reachability, we assign to $\bot$ the value which is the most disadvantageous for the controller player.

In the functions above, we postpone the assignment of rewards and cost corresponding to timed passage to guarded command transitions (jump-steps). The reason is that this method can provide better reward bounds than would be obtained by constructing bounds on the reward and time accumulated over each transition separately.

Proposition 4. *Let $\mathcal{H}$ be a PHA and $\mathsf{game}(\mathcal{H}')$ the game-based abstraction of the modified PHA $\mathcal{H}'$ for some game AG $(\mathcal{A},\mathcal{R})$. Using the reward and cost functions defined above for $[\![\mathcal{H}]\!]$ and $\mathsf{game}(\mathcal{H}')$, and omitting them from the subscripts of* val *for clarity, for any $\mathsf{z}\in\mathcal{A}$ and $(s,\mathtt{t},\mathtt{r})\in\mathsf{z}$, we have:*

$$\mathrm{val}^{\inf,\mathrm{obj}_c,\mathrm{obj}_e}_{\mathsf{game}(\mathcal{H}')}(\mathsf{z})\leq\mathrm{val}^{\mathrm{obj}_c,\mathrm{obj}_e}_{[\![\mathcal{H}]\!]}(s)\leq\mathrm{val}^{\sup,\mathrm{obj}_c,\mathrm{obj}_e}_{\mathsf{game}(\mathcal{H}')}(\mathsf{z}).$$

Furthermore, if $\mathsf{env}(\mathcal{H}')$ is the environment abstraction of $\mathcal{H}'$ for some environment AG $(\mathcal{A}',\mathcal{R}')$, then for the reward and cost functions defined above (again omitting subscripts for clarity) we have for any $\mathsf{z}'\in\mathcal{A}'$ and $(s',\mathtt{t}',\mathtt{r}')\in\mathsf{z}'$:

$$\begin{array}{rcccl}\mathrm{val}^{\inf,\inf}_{\mathsf{env}(\mathcal{H}')}(\mathsf{z}') & \leq & \mathrm{val}^{\inf,\mathrm{obj}_e}_{[\![\mathcal{H}]\!]}(s') & \leq & \mathrm{val}^{\inf,\sup}_{\mathsf{env}(\mathcal{H}')}(\mathsf{z}'),\\ \mathrm{val}^{\sup,\inf}_{\mathsf{env}(\mathcal{H}')}(\mathsf{z}') & \leq & \mathrm{val}^{\sup,\mathrm{obj}_e}_{[\![\mathcal{H}]\!]}(s') & \leq & \mathrm{val}^{\sup,\sup}_{\mathsf{env}(\mathcal{H}')}(\mathsf{z}').\end{array}$$

Using this result we can apply the controller synthesis approach described in Section VII to long-run properties.

Example 6. The thermostat already uses t to record time passage. We reason about the relative time spent in *Check* by assigning reward 0 to all commands. In the environment abstraction, we let $rew^{\inf,\sup}_{\mathsf{env}(\mathcal{H}')}((m,\mathsf{z}),\mu)=\mathtt{T}^{\sup}(\mathsf{z})$ if $m=Check$ and 0 otherwise, and $cost^{\inf,\sup}_{\mathsf{env}(\mathcal{H}')}(\mathsf{z},\mu)=\mathtt{T}^{\inf}(\mathsf{z})$. In Example 5 we thus have $rew^{\inf,\sup}_{\mathsf{env}(\mathcal{H}')}((\mathsf{z}_6,\subset),\mu)=0.9$ and $cost^{\inf,\sup}_{\mathsf{env}(\mathcal{H}')}((\mathsf{z}_6,\subset),\mu)=0.3$. ■

VIII. Experiments

We have implemented the environment abstraction in an extension of our tool ProHVer [1]. To do so, we transform a labelled transition system, obtained from a modified version of PHAVer, into an SG that abstracts the PHA under consideration. ProHVer then computes reachability probabilities in this game using value iteration [19]. This also yields a strategy for player *con* that can be used to synthesise a controller as described in Section VI.

We return to the thermostat example and consider the probability that a failure is detected (i.e. that *Safe* is entered) within time bound B. To this end we measure the total time that has elapsed and disable the transition to *Check* after more than B units of time. Formally, we add a variable x, initialised to 0 in *Init*, add flow constraint $\dot{x}=1$ to each mode and change the guard of the command in mode *Check* to $(t\geq 0)\wedge(x\leq B)$. We assume that the controller player tries to maximise the probability to reach *Safe*. Concerning the

B	interval length 0.5			interval length 0.2		
	prob.	build(s)	#states	prob.	build(s)	#states
1	[0.000, 0.000]	0	20	[0.000, 0.000]	0	79
4	[0.000, 0.700]	10	917	[0.000, 0.700]	44	3590
5	[0.000, 0.910]	14	1051	[0.700, 0.910]	54	4066
10	[0.910, 0.992]	81	4330	[0.910, 0.992]	413	16773
15	[0.973, 0.999]	50	3216	[0.992, 0.999]	2578	53289
20	[0.998, 1.000]	214	10676	[0.999, 1.000]	1435	41313
25	[0.999, 1.000]	160	8671	[1.000, 1.000]	928	32864

Table I
THERMOSTAT RESULTS

environment abstraction player, we consider both minimising and maximising behaviour, in order to obtain upper and lower bounds on the concrete reachability probability (see Proposition 3).

Table I gives probability bounds and performance results for different values of B. Thereby "interval length" (i) denotes a parameter of PHAVER related to $t_{\max}$ by $i{>}t_{\max}$.

The time (in seconds) needed to build the abstraction is given in "build(s)" and the number of abstract states in "#states". Probability bounds obtained are given in row "prob.", rounded to 3 decimal places. We see that increasing analysis precision gives tighter probability bounds but requires more resources in general. Because of the way PHAVER computes abstractions however, increase in memory and time usage is not monotonic, as we discuss in more detail in an earlier paper [1].

IX. CONCLUSIONS

In this paper, we have presented novel abstraction techniques for probabilistic hybrid automata based on the use of n-player stochastic games. These yield lower and upper bounds on optimal reachability probabilities and long-run average rewards. Our *game-based* abstraction gives more precise bounds but is harder to construct in practice; our *environment* abstraction is simpler to build but coarser. For the latter, we demonstrated an implementation built on top of PHAVER. We also showed how it can be used to synthesise controllers for PHAs. Future work includes implementation of the game-based abstraction, initially on simpler subclasses such as linear PHAs, and extending our approaches to parity-based properties and stochastic stability.

Acknowledgements. The authors are supported in part by: the DFG/NWO Bilateral Research Programme ROCKS; the DFG, as part of SFB/TR 14 AVACS; the EC FP-7 programme under grant agreement no. 214755 (QUASIMODO); the ERC-funded project VERIWARE; the EPSRC-funded project APEX (grant no. EP/G069158); and MT-LAB, a VKR Centre of Excellence.

REFERENCES

[1] L. Zhang, Z. She, S. Ratschan, H. Hermanns, and E. M. Hahn, "Safety verification for probabilistic hybrid systems," in *Proc. CAV'10*, ser. LNCS, vol. 6174. Springer, 2010.

[2] J. Lygeros, D. Godbole, and S. Sastry, "A game-theoretic approach to hybrid system design," in *Proc. Hybrid Systems*, ser. LNCS, vol. 1066. Springer, 1995.

[3] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. Larsen, and D. Lime, "UPPAAL-Tiga: Time for playing games!" in *Proc. CAV'07*, ser. LNCS, vol. 4590. Springer, 2007.

[4] M. Kwiatkowska, G. Norman, and D. Parker, "Game-based abstraction for Markov decision processes," in *Proc. QEST'06*. IEEE Press, 2006.

[5] ——, "Stochastic games for verification of probabilistic timed automata," in *Proc. FORMATS'09*, ser. LNCS, vol. 5813. Springer, 2009.

[6] B. Wachter and L. Zhang, "Best Probabilistic Transformers," in *Proc. VMCAI'10*, ser. LNCS, vol. 5944. Springer, 2010.

[7] B. Wachter, "Refined probabilistic abstraction," Ph.D. dissertation, Saarland University, 2010.

[8] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech," in *Proc. HSCC'05*, ser. LNCS, vol. 3414. Springer, 2005.

[9] E. M. Hahn, G. Norman, D. Parker, B. Wachter, and L. Zhang, "Game-based abstraction and controller synthesis for probabilistic hybrid systems," SFB/TR 14 AVACS, AVACS Technical Report No. 74, 2011.

[10] J. Sproston, "Decidable model checking of probabilistic hybrid automata," in *Proc. FTRTFT'00*, ser. LNCS, vol. 1926. Springer, 2000.

[11] P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen, "Reachability analysis of probabilistic systems by successive refinements," in *Proc. PAPM/PROBMIV'01*, ser. LNCS, vol. 2165. Springer, 2001.

[12] J. Desharnais and J. Assouramou, "Analysis of non-linear probabilistic hybrid systems," in *Proc. QAPL'11*, 2011.

[13] M. Fränzle, H. Hermanns, and T. Teige, "Stochastic satisfiability modulo theory: A novel technique for the analysis of probabilistic hybrid systems," in *Proc. QAPL'08*, 2008.

[14] M. Fränzle, T. Teige, and A. Eggers, "Engineering constraint solvers for automatic analysis of probabilistic hybrid automata," *JLAP*, vol. 79, no. 7, 2010.

[15] V. Forejt, M. Kwiatkowska, G. Norman, and A. Trivedi, "Expected reachability-time games," in *Proc. FORMATS'10*, ser. LNCS, vol. 6246. Springer, 2010.

[16] M. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.

[17] M. Ummels, "Stochastic multiplayer games: Theory and algorithms," Ph.D. dissertation, RWTH Aachen, 2010.

[18] J. Filar and K. Vrieze, *Competitive Markov Decision Processes*. Springer, 1996.

[19] A. Condon, "The Complexity of Stochastic Games," *Information and Computation*, vol. 96, no. 2, 1992.

[20] L. de Alfaro, "How to specify and verify the long-run average behavior of probabilistic systems," in *Proc. LICS'98*. IEEE Press, 1998.

[21] C. von Essen and B. Jobstmann, "Synthesizing systems with optimal average-case behavior for ratio objectives," in *Proc. iWIGP'11*, 2011.

[22] T. Henzinger, "The theory of hybrid automata," in *Proc. LICS'96*. IEEE Press, 1996.

[23] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic model checking for real-time systems," *Inf. and Comp.*, vol. 111, no. 2, 1994.

[24] T. Henzinger, P. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" *Journal of Computer and System Sciences*, vol. 57, 1998.