Learning Optimal Decisions for Stochastic Hybrid Systems

Mathis Niehage
Westfälische Wilhelms-Universität
Münster, Germany
mathis.niehage@wwu.de

Arnd Hartmanns
University of Twente
Enschede, The Netherlands
a.hartmanns@utwente.nl

Anne Remke
Westfälische Wilhelms-Universität
Münster, Germany
anne.remke@wwu.de

ABSTRACT

We apply reinforcement learning to approximate the optimal probability that a stochastic hybrid system satisfies a temporal logic formula. We consider systems with (non)linear continuous dynamics, random events following general continuous probability distributions, and discrete nondeterministic choices. We present a discretized view of states to the learner, but simulate the continuous system. Once we have learned a near-optimal scheduler resolving the choices, we use statistical model checking to estimate its probability of satisfying the formula. We implemented the approach using Q-learning in the tools HYPEG and modes, which support Petri net- and hybrid automata-based models, respectively. Via two case studies, we show the feasibility of the approach, and compare its performance and effectiveness to existing analytical techniques for a linear model. We find that our new approach quickly finds near-optimal prophetic as well as non-prophetic schedulers, which maximize or minimize the probability that a specific signal temporal logic property is satisfied.

ACM Reference Format:
Mathis Niehage, Arnd Hartmanns, and Anne Remke. 2021. Learning Optimal Decisions for Stochastic Hybrid Systems. In *19th ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE '21), November 20–22, 2021, Beijing, China.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3487212.3487339

1 INTRODUCTION

Cyber-physical systems, like critical infrastructures, smart homes, and electric vehicles, play an ever-increasing role in our daily lives. Their dependability is of utmost importance to society and industry. They combine discrete and continuous behavior in complex ways, requiring hybrid modeling formalisms. Once we incorporate timed events following general probability distributions, e.g. failures and repairs, we are in the realm of *stochastic hybrid models* (SHM).

Nondeterminism is often used in SHM to model concurrency or to account for underspecification. As a modeling feature, it is a standard part of most non-stochastic hybrid automata formalisms. In contrast, stochastic Petri net models often resolve nondeterminism through probabilistic choice *as part of the model* [8], since its combination with (non-Markovian) probability distributions poses a serious challenge when evaluating these models. Existing analytical approaches for SHM heavily rely on discretization and overapproximation to reason about *maximum reachability probabilities*, i.e. the worst-case resolution of nondeterminism to reach an unsafe state. They discretize the state space [1, 14] or the support of random variables [20], and face state space explosion.

Simulative approaches for SHM rarely address the optimal resolution of nondeterminism. Most statistical model checkers, e.g. UPPAAL SMC [7] or PRISM [29], resolve nondeterminism probabilistically *as part of the analysis*. A particular resolution of all nondeterminism in a model is captured by a *scheduler*; these tools implicitly use the uniform random scheduler. While the Modest Toolset's [22] modes simulator [4] supports SHM with linear dynamics as well as lightweight scheduler sampling [32] to approximate optimal schedulers, it provides the latter only for non-hybrid models [10, 12]. Hybrid Petri nets with general transitions (HPnGs) [19] form a restricted subclass of SHM [39] with linear dynamics. Their inherent nondeterminism was optimally resolved analytically in [38, 42] via (non)prophetic [23] scheduling. The HPnG simulator HYPEG [37] has recently been extended to support linear and nonlinear differential equations in HPnG [36, 40], but again cannot optimize over nondeterminism. An investigation of scheduler classes for the (non-hybrid) model of stochastic automata showed that history-dependent prophetic schedulers are the strictly most powerful ones [11].

Our contribution is to bring optimization over nondeterminism to SHM with nonlinear continuous dynamics and discrete nondeterministic choices by finding optimal memoryless prophetic or nonprophetic schedulers through reinforcement. We use Q-learning to learn the maximal reward that can be achieved for every nondeterministic choice. The result induces an optimized (but not necessarily optimal) scheduler, under which we then simulate the SHM. Our approach then either computes a confidence interval for the probability that a certain STL formula is fulfilled, or uses hypothesis testing to evaluate whether the simulated probability fulfills a predefined threshold. While we always simulate the original continuous SHM, also during Q-learning, we only present a discretized view to the learner itself so that Q-table and scheduler remain finite objects. We describe this approach in detail in Sections 3 and 4. We implemented the approach in two tools: as an extension to HYPEG, and in modes. We show its feasibility on two case studies in Section 5. One has linear and one has nonlinear dynamics. Both are featured as a HPnG model for HYPEG and a stochastic hybrid automata [14] model specified in Modest [20] for modes.

Related work. We focus on (nonlinear) *continuous-time* hybrid systems; the discrete-time view admits many other solutions (e.g. [6]). The combination with general probability distributions and nondeterministic choices poses a challenging verification problem. Analytical approaches have various limitations: A recently proposed discretization in time and space is for switched diffusions only [30].

MEMOCODE '21, November 20–22, 2021, Beijing, China

ACM ISBN 978-1-4503-9127-6/21/11.
https://doi.org/10.1145/3487212.3487339

Pilch et al. [38] require piecewise constant derivatives and cannot solve for complex prophetic schedulers. prohver [20] provides upper (lower) bounds on maximum (minimum) probabilities only. Delta reachability in ProbReach [46] does not support nondeterminism. Stochastic SMT solving [15, 17] effectively works for nondeterministic choices over initial values/parameters only, limiting the types of schedulers that can be considered. All of these techniques hardly scale as the number of abstract/symbolic states or choice combinations explodes.

In this paper, we thus pursue a simulation-based approach. Simulation, however, is incompatible with nondeterminism as-is. For (finite-state) Markov decision processes (MDP), this gap is bridged by *sampling* schedulers [32] (which does not work well for continuous-state models [12]) or by *learning* them [3, 26]. We follow the latter idea of combining simulation and reinforcement learning. On reinforcement learning with MDP, various methods exist to improve its behavior and performance for safety objectives (e.g. [24]), to deal with *unknown* stochastic behavior [5, 25], and with linear-time logic specifications (e.g. [5, 21, 45]).

We, however, focus on (infinite-state) stochastic hybrid systems in continuous time. In the discrete-time setting, Lavaei et al. [31] recently proposed a way to learn on an abstraction of the continuous state space that provides guarantees on the error of the final results. Other than that, learning for these systems—in particular in continuous time—has not been well-explored from a formal perspective w.r.t. the scheduler class being considered and the related (potential) optimality of the result. Specifically, a variant of ProbReach combines Monte Carlo methods for the stochastic hybrid part with the cross-entropy method for nondeterminism [47], and the SMT-based approach has been twisted to use simulation with decision trees and AI planning algorithms to handle nondeterminism [13]. Both limit nondeterminism to initial value/parameter choices again, and thus the scheduler class. More generally, Geibel and Wysotzki [18], for example, propose an algorithm that is applicable to this setting "for it makes no prior assumptions" on these aspects, but use an ad-hoc scheduler class and also caution that their use of function approximation to deal with the continuous state space voids any convergence guarantees. The methods evaluated in [33] face similar issues. In contrast, we explicitly consider the difference between prophetic and nonprophetic scheduling, and show eventual convergence (Theorem 3.6 in combination with the known convergence properties of Q-learning). The work of Junges et al. [28] is notable in that it takes a formal approach that includes discretization of a continuous environment as well as reinforcement learning; however they take a learned strategy as *input* and then proceed with probabilistic model checking. Finally, Fulton et al. [16] provide an overview of other interesting application challenges related to learning in cyber-physical (i.e. hybrid) systems.

2 MODELS, SCHEDULERS, AND PROPERTIES

Our approach is agnostic to the specific modelling formalism as long as it can be expressed in terms of the SHM and DES "interfaces" that we define below. We then explain the class of schedulers and properties that we focus on.

Algorithm 1: One simulation run of a DES

```
1 startExecution(M, Γ_init, t_max)
2 (finished, Γ, A_Γ) = simulateInitialTrajectory()
3 while ¬ finished do
4 |   a = chooseAction(Γ, A_Γ)
5 |   (finished, Γ, A_Γ) = simulateTrajectory(a)
6 end
```

2.1 Discrete Event System

A discrete event system (DES) simulates an instance of a SHM with continuous dynamics described by first-order ordinary differential equations as follows:

Definition 2.1 (Stochastic Hybrid Model (SHM)). A *stochastic hybrid model* (SHM) M is in a state $\Gamma \in S$, where S is an uncountable state space over $n_c \in \mathbb{N}$ continuous variables $x_i \in \mathbb{R}$, $i \in \{1, \ldots, n_c\}$ and $n_d \in \mathbb{N}$ discrete modes $m_j \in \mathbb{N}$, $j \in \{1, \ldots, n_d\}$. For each mode m_j, a system of first-order ordinary differential equations $\dot{x} = f_j(t, x(t))$, where f_j is Lipschitz, describes the evolution of the continuous variables over time. For a time-bounded evolution of the model in state Γ without Zeno behavior, the specific semantics of the SHM then induces finite numbers of $n_t \in \mathbb{N}$ deterministic events and $n_r \in \mathbb{N}$ random delays. Each of them is associated with a timer, which is initialized with zero and equipped with an expiration time that is either deterministic or sampled from the corresponding continuous probability distribution. Timer x_i has derivative 1 if the corresponding jump or delay is active and 0 otherwise. All timers induce *actions* in the SHM, which are collected in the finite set $\mathcal{A}$. The next action is $a = \arg\min_{a' \in \mathcal{A}}(e_{a'} - x_{a'})$, where $x_{a'}$ is the current valuation of the timer and $e_{a'}$ its associated expiration time. We write $A_\Gamma \subseteq \mathcal{A}$ for the action set at state $\Gamma \in S$. If $|A_\Gamma| > 1$, then a nondeterministic *conflict* exists in Γ and if $|A_\Gamma| = 0$, only time can pass. Executing one action then updates the state Γ according to the specific semantics of the SHM.

The specific semantics of the SHM can be defined by e.g. stochastic hybrid automata (SHA) [20] or HPnGs [19, 40]. This determines $\mathcal{A}$ and the associated timers; these may e.g. be due to deterministic transitions in a HPnG or the satisfaction of edge guards over the continuous variables in an SHA.

Definition 2.2 (Discrete Event System (DES)). A *discrete event system* $DES(M, \Gamma, t)$ groups a SHM M, a state $\Gamma \in S$ of M, and a simulation time t. It is initialized as $DES(M, \Gamma_{\text{init}}, t_{\max})$ with an initial state Γ_{init} and a maximum time horizon $t_{\max}$. The state $(\Gamma, t) \in \Sigma = (S \times [0, t_{\max}])$ of a DES consists of the state of the SHM and the remaining simulation time t.

Algorithm 1 induces one simulation run of the DES. The execution starts in the initial configuration of the DES. First the random variables of M are sampled (line 1). The trajectories of the variables of M are continuously updated and the remaining simulation time t, included in the state of the DES, is reduced with derivative -1 (line 2). At a nondeterministic conflict, the system pauses, returning the state $\Gamma \in S$ of the SHM and the conflict set A_Γ. The system stops when the simulation time bound is reached, i.e. when t equals 0, represented by the Boolean *finished*. After a conflict, a selected

action $a \in A_\Gamma$ from the conflict set (line 4) is required as input and the simulation of the trajectory continues (line 5). Such an execution results in a discretization of time, with each conflict of the DES corresponding to a discretization point between 0 and $t_{\max}$. To resolve the conflicts, a scheduler can be specified which selects one action from the conflict set A_Γ.

2.2 Schedulers

Whenever a conflict occurs, a scheduler chooses among the actions $a \in A_\Gamma$ in state $\Gamma \in S$. Schedulers may map to discrete probability distributions over the available actions:

Definition 2.3 (Discrete Probability Distribution). A *discrete probability distribution* $\mu\colon X \to [0,1]$ assigns to all $x \in X$ a probability $\mu(x)$ s.t. $support(\mu) = \{x \in X \mid \mu(x) > 0\}$ is countable and $\sum_{x \in support(\mu)} \mu(x) = 1$. The set of all probability distributions over X is $\mathrm{Dist}(X)$.

Schedulers can have different knowledge of the system when choosing an action, resulting in schedulers with different power [11]. While *memoryless* schedulers only see the current state $\Gamma \in S$ of the SHM, *history-dependent* schedulers know the finite path that has been taken until (Γ, t). Furthermore, schedulers with knowledge on future random events, i.e. the sampled values, are called *prophetic* schedulers [23]. Schedulers without that knowledge are *nonprophetic*. For stochastic automata, we know [11] that prophetic memoryless and history-dependent schedulers are equally powerful for unbounded reachability properties, which is not the case for nonprophetic schedulers due to the memoryless scheduler being unable to deduce information on the random variables present in the history. All else equal, prophetic schedulers are in general more powerful than nonprophetic ones. We consider prophetic and nonprophetic memoryless schedulers:

Definition 2.4 (Memoryless Nonprophetic Scheduler). A *memoryless nonprophetic scheduler* for a SHM with state set S is a measurable function $\mathfrak{s}\colon S \to \mathrm{Dist}(\mathcal{A})$ that maps every state $\Gamma \in S$ to a discrete probability distribution with $support(\mathfrak{s}(\Gamma)) \subseteq A_\Gamma$. Let $\mathfrak{S}^n$ denote this class of schedulers.

Definition 2.5 (Memoryless Prophetic Scheduler). A *memoryless prophetic scheduler* for a SHM is a measurable function $\mathfrak{s}\colon S \times (\mathbb{R}^+)^{n_r} \to \mathrm{Dist}(\mathcal{A})$ that maps every state $\Gamma \in S$ and assignment of n_r real positive values for future random events $\mathbf{r}$ to a discrete probability distribution, with $support((\Gamma, \mathbf{r})) \subseteq A_\Gamma$. This class of schedulers is $\mathfrak{S}^p$.

The class of schedulers that map solely to Dirac distributions is called *deterministic* ($\mathfrak{S}^{c,d}$ for $c \in \{n, p\}$). The above definitions allow a wide variety of functions. For example, a function similar to the Dirichlet function, which assigns probability 1 to an action $a_1 \in A_\Gamma$ if the valuation of a continuous variable is rational and otherwise to action $a_2 \in A_\Gamma$, is a valid scheduler. We limit ourselves to *piecewise-constant* schedulers, which partition the state space in a finite set of intervals where the scheduler remains constant within each interval.This restriction is arguably practical, allowing such schedulers to be reasonably implemented.

2.3 Statistical Model Checking

We now consider the DES a black box, hiding the simulation details. A model checker monitors the state evolution of the simulation and decides whether the system execution satisfies a formula in signal temporal logic (STL) [34], which expresses linear-time properties over the state of a hybrid model.

Definition 2.6 (Signal Temporal Logic (STL)). A *signal temporal logic* (STL) property can be constructed from the context-free grammar

$$\Psi ::= tt \mid AP \mid \neg\Psi \mid \Psi \wedge \Psi \mid \Psi\, U^{[t_1,t_2]}\, \Psi$$

consisting of true (tt) and atomic properties (AP), which can be negated and combined with a logical *and* or with a time-bounded *until*-operator. An atomic property compares a function $f : S \to \mathbb{R}$ with the value zero: $f(\Gamma) > 0$.

Note that STL as in [34] only supports continuous signals, i.e. continuous variables in our context. Definition 2.6 allows referring to discrete modes of the SHM; this can be mapped to continuous signals. For the semantics of STL in the context of HPnGs, we refer the interested reader to [41].

We now cover statistical model checking of STL properties via multiple simulation runs. For each run of the DES with time bound $t_{\max}$, a property Ψ can be checked. Given a scheduler, n simulation runs result in a sequence of independent values $p_1, \ldots, p_n \in \{0, 1\}$, each indicating whether the run fulfils Ψ. Each p_i is a sample of a Bernoulli distribution. The arithmetic mean $\hat{p} = \frac{1}{n}\sum_{i=1}^{n} p_i$ is an estimator of the actual probability p of Ψ. A confidence interval $[a, b]$ can be computed such that $\hat{p} \in [a, b]$ and $b - a = 2w$ for a given half width $w \in (0, 1]$. For a given confidence $\delta \in [0, 1]$, it is guaranteed that $100 \cdot \delta\,\%$ of the computed confidence intervals contain p. The number of runs required depends on w and δ. For a detailed explanation, we again refer to [41].

Since the probabilities are computed for a specific scheduler, choosing prophetic or nonprophetic schedulers $\mathfrak{S}^c$, $c \in \{n, p\}$, results in a range of possible probabilities $[p^{\mathfrak{S}^c}_{min}(\Psi, t), p^{\mathfrak{S}^c}_{max}(\Psi, t)]$ for a given formula Ψ. The interval's bounds are the infimum, respectively the supremum, over the probabilities induced by all schedulers in that class, i.e.:

$$p^{\mathfrak{S}^c}_{min}(\Psi, t) = \inf_{\mathfrak{s} \in \mathfrak{S}^c} p(\Psi, \mathfrak{s}, t) \text{ and } p^{\mathfrak{S}^c}_{max}(\Psi, t) = \sup_{\mathfrak{s} \in \mathfrak{S}^c} p(\Psi, \mathfrak{s}, t),$$

where $p(\Psi, \mathfrak{s}, t)$ is the probability of the satisfaction of Ψ at time point t under scheduler $\mathfrak{s}$. We refer to these optimal probabilities as minimum and maximum.

In addition to estimating a probability, statistical model checking can also use hypothesis testing to decide whether the probability that a specific STL property Ψ is fulfilled at time t is bounded by a predefined value p. In our case this translates to checking whether the maximal or minimal probability is bounded, i.e. $p^{\mathfrak{S}^c}_m(\Psi, t) \bowtie p$, for $\bowtie = \{<, \leq, >, \geq\}$ and $m \in \{min, max\}$. Different hypothesis tests can be used, as discussed in [43]. In the following, we use the Azuma test [44] which guarantees predefined errors of first (false positive) and second (false negative) kind, even for small $|p - p^{\mathfrak{S}^c}_m(\Psi, t)|$. Potentially many SMC runs are required to obtain a result.

3 Q-LEARNING FOR HYBRID SYSTEMS

Q-learning, which we summarize below, requires a finite state space [48]. Since SHM induce an infinite state space, we propose an abstraction which is obtained by discretizing the continuous variables. We finally explain the interactions between discretization and schedulers.

3.1 Q-learning

Q-learning is a reinforcement learning method, and hence learns from experience, by rewarding the positive ones. Given a discrete and finite state space $\mathcal{S}$ and an action space $\mathcal{A}$, the learned function $Q\colon \mathcal{S}\times\mathcal{A}\to\mathbb{R}$ estimates the optimal action-value function which judges how good taking a certain action at a given state is. A predefined amount of training runs, also called *episodes*, is performed which modify the Q-function for the occurring state-action pairs. Initially, the Q-function is arbitrary for all $(s,a)\in\mathcal{S}\times\mathcal{A}$. For each training run the state $s\in\mathcal{S}$ is set to its initial value, potentially chosen from a set. Afterwards the system is simulated until a terminal state is reached, which in our case always corresponds to reaching the time bound. At each state $s\in\mathcal{S}$ that requires a decision, an available action $a\in\mathcal{A}$ is selected which leads to state s'. Additionally a reward $R:\mathcal{S}\to\mathbb{R}$ is defined as a function of s'. $Q(s,a)$ is then updated as follows:

$$Q(s,a) = Q(s,a) + \alpha\left(R(s') + \gamma \max_{\tilde{a}} Q(s',\tilde{a}) - Q(s,a)\right). \quad (1)$$

The current value $Q(s,a)$ is increased by the learning rate $\alpha\in[0,1]$ times the sum of the reward and the potential reward from state s', which in turn is discounted by $\gamma\in[0,1]$, minus the current value of the learning function. Note that the open interval of gamma is relaxed to the closed range due to episodic learning. The potential reward is determined as the maximum value of Q over all possible actions.

Q-learning can use different *policies* to choose an action during the training process. For example, the ϵ-greedy policy chooses the action which results in the maximum value of Q with probability $1-\epsilon$ and an arbitrary action otherwise [48]. Thus, every action will be sampled infinitely often assuming an infinite number of episodes, which ensures the convergence to the optimal action-value function. Then taking the action with the maximum value at a given state grants the optimal deterministic scheduler after infinitely many runs [48].

3.2 Discretization of the state space

The state space S of a SHM consists of continuous variables and discrete modes. Considering the time bounded evolution of a SHM with time bound $t_{\max}$, each continuous variable $x_i\in\mathbb{R}$ can only take a bounded range of values before the time bound is reached, since the variable evolution is Lipschitz continuous. This range naturally has a lower bound $l_i^x\in\mathbb{R}$ and an upper bound $u_i^x\in\mathbb{R}$. To discretize the state space w.r.t. x_i, we split the interval $[l_i^x, u_i^x]$ into k_i intervals which yields a finite set of (half-open) intervals. All intervals have the same width and each interval is represented by one value that lies inside that interval, w.l.o.g. the minimum value in the interval is chosen. Hence, the set $\mathfrak{X}_\mathbf{i}(k_i) = \{l_i^x + j\frac{u_i^x - l_i^x}{k_i} \mid j\in\{0,\dots,k_i-1\}\}$ is the discretized grid of the continuous variable x_i with k_i intervals.

Let $\mathbf{k^c}\in\mathbb{N}^{n_c}$ be the vector containing all k_i. Since the number of continuous variables is finite, also the Cartesian product of their discretized grids is finite: $\mathfrak{X}(\mathbf{k^c}) = \bigtimes_{i>0}^{n_c}\mathfrak{X}_i(k_i)$.

To obtain a discretized state space, also the discrete modes must be included. Each discrete mode $m_j\in\mathbb{N}$ has an upper bound u_j^m and lower bound l_j^m due to the finite time horizon and excluding Zeno behavior. Thus, the reachable state space of variable m_j is $\mathfrak{D}_\mathbf{j} = \{n \mid n\in\{l_j^m,\dots,u_j^m\}\subseteq\mathbb{N}\}$. The Cartesian product of the reachable discrete state spaces is also finite: $\mathfrak{D} = \bigtimes_{i>0}^{n_d}\mathfrak{D}_i$. The finite discretization of the state space S of the SHM can thus be constructed as follows:

Definition 3.1 (Discretized State Space). The discretized state space is the Cartesian product of the finite representation of the continuous variables and the discrete modes:

$$\mathcal{S}(\mathbf{k^c}) = \mathfrak{X}(\mathbf{k^c})\times\mathfrak{D}.$$

The relation between the state space and its finite discretized representation is described by a mapping function:

Definition 3.2 (Map on Finite Discretized State Space). The function $\mathbf{f} = (f_1,\dots,f_{n_c+n_d})\colon S\to\mathcal{S}(\mathbf{k^c})$ maps the state space on the finite discretized state space with $\mathbf{v} = (x_1,\dots,x_{n_c},m_1,\dots,m_{n_d})$:

$$f_i(\mathbf{v}) = \begin{cases} l_i^x + \left\lfloor \frac{x_i - l_i^x}{\Delta_i}\right\rfloor \Delta_i, & i = 1,\dots,n_c \wedge x_i\in[l_i^x,u_i^x],\\ m_i, & i = n_c+1,\dots,n_c+n_d\\ & \wedge m_{i-n_c}\in[l_{i-n_c}^m, u_{i-n_c}^m],\\ \bot, & else, \end{cases}$$

where $\Delta_i = \frac{u_i^x - l_i^x}{k_i^c}$ is the interval size of $\chi_i\in\mathfrak{X}_i(k_i^c)$ and $\bot$ represents $\emptyset$.

Prophetic scheduling requires knowledge of the random variables' valuations. Thus, they must be included in the state space. Due to the finite time horizon, lower and upper bounds l_i^r and u_i^r exist and $\mathfrak{R}_i(k_i^r) = \{l_i^r + j\frac{u_i^x - l_i^x}{k_i^r} \mid j\in\{0,\dots,k_i-1\}\}$ is constructed similarly to the other continuous variables for all n_r random variables. Note that this also applies to random variables with unbounded support since only time bounded evolution is considered. With $\mathbf{k^r}$ containing all $k_i^r, i\in\{1,\dots,n_r\}$, the finite grid of the valuations of the random variables is

$$\mathfrak{R}(\mathbf{k^r}) = \bigtimes_{i>0}^{n_r}\mathfrak{R}_i(k_i^r).$$

Definition 3.3 (Prophetic Discretized State Space). The discretized state space for prophetic scheduling is the Cartesian product of the discretized state space $\mathcal{S}(\mathbf{k^c})$ and the finite representation of the valuations of the the n_r random variables $\mathfrak{R}(\mathbf{k^r})$ with $\mathbf{k} = (\mathbf{k^c},\mathbf{k^r})$:

$$\mathcal{P}(\mathbf{k}) = \mathcal{S}(\mathbf{k^c})\times\mathfrak{R}(\mathbf{k^r}).$$

The mapping function $\mathbf{f}^r : S\times\mathbb{R}^{n_r}\to\mathcal{P}(\mathbf{k})$ extends the mapping from $\mathbf{f}$ in Definition 3.2 by mapping all valuations of the random variables with $j = i - n_c - n_d$:

$$l_j^r + \left\lfloor \frac{x_j - l_j^r}{\Delta_j}\right\rfloor \Delta_j, \quad i = n_c+n_d+1,\dots,n_c+n_d+n_r$$
$$\wedge\, r_j\in[l_j^r, u_j^r].$$

The presented discretization generates a MDP without rewards since between states in the (prophetic) discretized state space transition probabilities are induced by the underlying SHM. It is possible to compute these transition probabilities and then further analyze the MDP, e.g. with value iteration [48]. However, this would not only be computationally expensive and does not scale for more precise discretizations, but more importantly the resulting MDP only represents an approximation of the stochastic hybrid model. Instead, we propose to statistically simulate the system as SHM and whenever a nondeterministic decision is required, the current state is mapped on the discretization where a scheduler is learned. Hence, only the set of learnable schedulers is restricted—which we further investigate in the following section—but the stochastic hybrid behavior remains unaffected.

3.3 Influence of Discretization on Optimal Schedulers

The discretization changes the information a scheduler can base its decisions on. It restricts a scheduler's preimage to $\mathcal{S}(\mathbf{k}) \subset S$ in the nonprophetic case (cf. Definition 2.4) and to $\mathcal{P}(\mathbf{k}) \subset S \times (\mathbb{R}^+)^{n_r}$ (cf. Definition 2.5) otherwise. Those classes of schedulers are denoted $\mathfrak{S}^n_{\mathbf{k}} \subset \mathfrak{S}^n$ and $\mathfrak{S}^p_{\mathbf{k}} \subset \mathfrak{S}^p$, respectively.

Proposition 3.4. *For $\mathfrak{S}^c$ being either the prophetic or nonprophetic schedulers, i.e. $c \in \{n, p\}$, an STL formula Ψ, and time bound $t_{\max}$, we have:*

- *The maximum probability $p^{\mathfrak{S}^c_{\mathbf{k}}}_{max}(\Psi, t_{\max})$ computed over the discretized scheduler class with parameter $\mathbf{k}$ is an underapproximation of $p^{\mathfrak{S}^c}_{max}(\Psi, t_{\max})$:*

$$p^{\mathfrak{S}^c_{\mathbf{k}}}_{max}(\Psi, t_{\max}) \leq p^{\mathfrak{S}^c}_{max}(\Psi, t_{\max}). \tag{2}$$

- *Correspondingly, the minimum $p^{\mathfrak{S}^c_{\mathbf{k}}}_{min}(\Psi, t_{\max})$ is an overapproximation of $p^{\mathfrak{S}^c}_{min}(\Psi, t_{\max})$:*

$$p^{\mathfrak{S}^c_{\mathbf{k}}}_{min}(\Psi, t_{\max}) \geq p^{\mathfrak{S}^c}_{min}(\Psi, t_{\max}). \tag{3}$$

Proof. Since $\mathfrak{S}^c_{\mathbf{k}} \subset \mathfrak{S}^c$, the following holds:

$$\begin{aligned} p^{\mathfrak{S}^c_{\mathbf{k}}}_{max}(\Psi, t_{\max}) &= \sup_{\mathfrak{s} \in \mathfrak{S}^c_{\mathbf{k}}} p(\Psi, \mathfrak{s}, t) \\ &\leq \sup_{\mathfrak{s} \in \mathfrak{S}^c} p(\Psi, \mathfrak{s}, t) = p^{\mathfrak{S}^c}_{max}(\Psi, t_{\max}). \end{aligned} \tag{4}$$

The same reasoning leads to "≥" in the minimum case. □

As stated in the following proposition, different discretizations change the amount of information lost in the discretization process:

Proposition 3.5. *Given the prerequisites of Proposition 3.4, there always exists a finer discretization $\tilde{\mathbf{k}} > \mathbf{k}$ (element-wise), which yields a possibly better scheduler:*

$$\begin{aligned} p^{\mathfrak{S}^c_{\tilde{\mathbf{k}}}}_{min}(\Psi, t_{\max}) &\leq p^{\mathfrak{S}^c_{\mathbf{k}}}_{min}(\Psi, t_{\max}), \quad \text{and} \\ p^{\mathfrak{S}^c_{\tilde{\mathbf{k}}}}_{max}(\Psi, t_{\max}) &\geq p^{\mathfrak{S}^c_{\mathbf{k}}}_{max}(\Psi, t_{\max}). \end{aligned}$$

Proof. Choosing $\tilde{\mathbf{k}} = 2(i+1)\mathbf{k}$ with $i \in \mathbb{N}$, it holds that $\mathfrak{S}^c_{\mathbf{k}} \subset \mathfrak{S}^c_{\tilde{\mathbf{k}}}$ due to the construction of the discretized state spaces in Definition 3.1 and 3.3. Due to the relation of the scheduler sets, the above holds as in the proof of Proposition 3.4. □

Theorem 3.6 (Convergence). *Given the prerequisites of Prop. 3.4, for element-wise $\mathbf{k} \to \infty$ the optimal probabilities are reached:*

$$\begin{aligned} p^{\mathfrak{S}^c_{\mathbf{k}}}_{min}(\Psi, t_{\max}) &\to p^{\mathfrak{S}^c}_{min}(\Psi, t_{\max}), \quad \text{and} \\ p^{\mathfrak{S}^c_{\mathbf{k}}}_{max}(\Psi, t_{\max}) &\to p^{\mathfrak{S}^c}_{max}(\Psi, t_{\max}). \end{aligned}$$

Proof. We show that for each arbitrary but firm $\epsilon > 0$, there exists a $\mathbf{k}$ such that

$$|p^{\mathfrak{S}^c_{\mathbf{k}}}_{max}(\Psi, t_{\max}) - p^{\mathfrak{S}^c}_{max}(\Psi, t_{\max})| < \epsilon. \tag{5}$$

Let $\mathfrak{s} \in \mathfrak{S}^c$ be an optimal scheduler. Given a $\mathbf{k}'$ such that Equation 5 does not hold, two states $\mathbf{v_1}, \mathbf{v_2} \in S$ exists where $\mathbf{f}(\mathbf{v_1}) = \mathbf{f}(\mathbf{v_2})$ but different distributions are required as $\mathfrak{s}(\mathbf{v_1}) \neq \mathfrak{s}(\mathbf{v_2})$. The proof of Proposition 3.5 provides the scheme for a $\tilde{\mathbf{k}} = 2 \cdot (i+1)\mathbf{k}$ with $i \in \mathbb{N}$ which narrows the discretization grid. Since $\mathbb{Q}$ is dense in $\mathbb{R}$, there is always a $\tilde{\mathbf{k}}$ such that $\mathbf{f}(\mathbf{v_1}) \neq \mathbf{f}(\mathbf{v_2})$.

While there might be more $\mathbf{v_i}$ with $\mathbf{f}(\mathbf{v_i}) = \mathbf{f}(\mathbf{v_1})$, which still require different distributions of the scheduler function, the restriction to piecewise constant schedulers ensures that the amount of jumps is finite. Hence, the construction scheme can be applied iteratively, until a sufficient discretization is reached such that Equation 5 holds. The same reasoning holds for the minumum case. □

Note that the above proof does not hold for arbitrary schedulers since an uncountable amount of jumps might prevent the iterative process described above from terminating; then the discretization scheme might never reduce the difference in Equation 5.

4 CONNECTION BETWEEN LEARNING AND SIMULATION

We use Q-learning in the DES to learn optimizing schedulers where the rewards depend on the satisfaction of a STL property. For a specific learned scheduler, the probability that the STL property holds can be approximated via a confidence interval, or a hypothesis test can be used to decide whether this probability matches a predefined bound.

4.1 Learning Optimizing Schedulers

The main idea of learning an optimal scheduler via simulation is to give a reward to every training run after it reaches the time bound specified in the STL formula. The reward depends on the optimization goal and the validity of said STL formula. Figure 1 visualizes the sequence of one training run, which is explained in the following. First, the model is simulated until a nondeterministic conflict occurs. At this first decision the current state $\mathbf{v_1} \in S$ is mapped onto the finite discretized state space $\mathcal{S}(\mathbf{k})$ using the function $\mathbf{f}$ (cf. Definition 3.2). The mapped state is then stored together with the action a_1 that is chosen (e.g. via the ϵ-greedy policy). In the prophetic case, $\mathbf{v_1} \in S \times \mathbb{R}^{n_r}$ is mapped with $\mathbf{f}^r$ on $\mathcal{P}(\mathbf{k})$. The following explanation covers the nonprophetic case; for the prophetic case, we replace $\mathbf{f}$ with $\mathbf{f}^r$ and $\mathcal{S}(\mathbf{k})$ with $\mathcal{P}(\mathbf{k})$.

The simulation continues with the original state for the chosen action until the next nondeterministic conflict occurs. At the first conflict the discretized state $s_1 = \mathbf{f}(\mathbf{v_1}) \in \mathcal{S}(\mathbf{k})$ and the taken

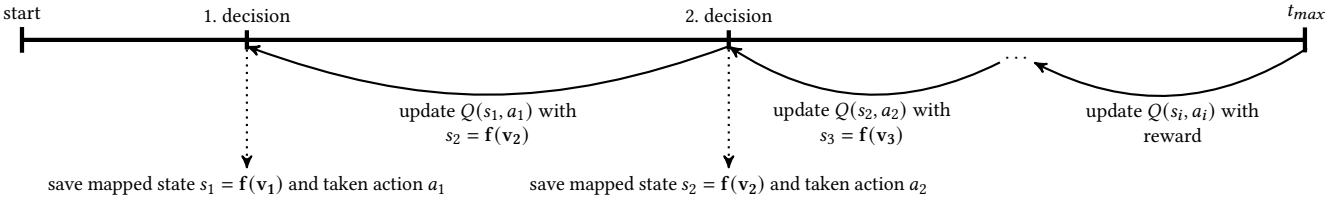

Figure 1: Timeline of one training run. At each decision the current state is mapped onto the discretization grid and saved together with the taken action. At the next decision Q is updated with the info of the subsequent state.

action a_1 had to be stored, and the Q-value for this state-action pair will be updated at the next decision, as the update depends on the subsequent state. Hence, for all following conflicts, the Q-value of the previous state-action pair must be updated in addition to storing the state-action pair. The update of the Q-value is done according to Equation 1. Note that it includes a reward, which stays zero ($R = 0$) until the time bound is reached and the Q-value for the previous decision is updated. With this last update, also the reward is set depending on the optimization goal. When maximizing, it is set to 1 if the training run satisfies the STL formula and to -1 if the STL property is violated, and vice-versa if the goal is to minimize.

A series of training runs modifies the Q-values of the state-action pairs; those occurring early during the training run are thus rated implicitly via the reward of the final decision. Recall from Section 3.1 that the scheduler is then obtained by choosing the action with the largest Q-value. Q-learning guarantees convergence to the optimal deterministic scheduler (modulo the discretization) only for an infinite number of training runs, which is practically impossible. Hence, two problems arise: the obtained schedulers are not necessarily optimal within the considered class of schedulers, and some states might not be explored yet.

On unexplored states, we employ a uniform distribution over all available actions to improve the training process. While the optimal scheduler with infinitely many training runs is always deterministic, this is not necessarily the case when probabilistic resolution is applied and the number of training runs is finite. The obtained scheduler is hence randomized and denoted $\mathfrak{s}_m^{\mathbf{k},n} \in \mathfrak{S}_{\mathbf{k}}^c$, for $m \in \{min, max\}$, $c \in \{n, p\}$, discretization $\mathbf{k}$, and n training runs.

To the best of our knowledge, it is impossible to provide guarantees on the quality of the learned scheduler for a finite number of training runs. Instead, we reason on the quality of a learned scheduler via the average probability and confidence intervals for a STL property. This is obtained from a statistical STL model checker with the original, non-discretized model and the learned scheduler.

In the following, we show that $\mathfrak{s}_m^{\mathbf{k},n}$ induces an (over- or under-)approximation of the optimal prophetic or nonprophetic probabilities, w.r.t. all scheduler classes.

Proposition 4.1. *The learned (minimal or maximal) prophetic or nonprophetic scheduler $\mathfrak{s}_m^{\mathbf{k},n} \in \mathfrak{S}_{\mathbf{k}}^c$, $m \in \{min, max\}, c \in \{n, p\}$ after a finite amount of n training runs induces an (over- or under-)approximation:*

$$p(\Psi, \mathfrak{s}_{\min}^{\mathbf{k},n}, t_{\max}) \geq p_{min}^{\mathfrak{S}^c}(\Psi, t_{\max}), \quad \textit{and}$$
$$p(\Psi, \mathfrak{s}_{\max}^{\mathbf{k},n}, t_{\max}) \leq p_{max}^{\mathfrak{S}^c}(\Psi, t_{\max}).$$

Proof. Since $\mathfrak{s}_m^{\mathbf{k},n} \in \mathfrak{S}_{\mathbf{k}}^c$, $m \in \{min, max\}$, it holds that

$$p(\Psi, \mathfrak{s}_{\min}^{\mathbf{k},n}, t_{\max}) \geq p_{min}^{\mathfrak{S}_{\mathbf{k}}^c}(\Psi, t_{\max}), \quad \text{and}$$
$$p(\Psi, \mathfrak{s}_{\max}^{\mathbf{k},n}, t_{\max}) \leq p_{max}^{\mathfrak{S}_{\mathbf{k}}^c}(\Psi, t_{\max}).$$

From Equation 2 and 3 in Proposition 3.4 the proof follows directly. □

The convergence of Q-learning ensures that after infinitely many training runs a deterministic scheduler is learned and

$$p(\Psi, \mathfrak{s}_m^{\mathbf{k},n}, t_{\max}) \to p_m^{\mathfrak{S}_{\mathbf{k}}^{c,d}}(\Psi, t_{\max})$$

for $m \in \{min, max\}$. Hence, the convergence of Theorem 3.6 only guarantees convergence towards the optimal deterministic scheduler, which might not necessarily be the optimal scheduler class. Note that the optimality of this class of schedulers, i.e. memoryless, piecewise constant, and deterministic, to the best of our knowledge is an open problem outside the scope of this paper.

4.2 Hypothesis Testing

Proposition 4.1 guarantees that the learned maximal scheduler computes an underapproximation $p(\Psi, \mathfrak{s}_{\max}^{\mathbf{k},n}, t_{\max})$ of $p_{max}^{\mathfrak{S}^c}(\Psi, t_{\max})$. Correspondingly, the learned minimal scheduler computes an overapproximation $p(\Psi, \mathfrak{s}_{\min}^{\mathbf{k},n}, t_{\max})$ of $p_{min}^{\mathfrak{S}^c}(\Psi, t_{\max})$. Thus, if a hypothesis test returns true for

$$p(\Psi, \mathfrak{s}_{\max}^{\mathbf{k},n}, t_{\max}) \bowtie p \text{ for } \bowtie \in \{>, \geq\}, \tag{6}$$

we conclude within the statistical error bounds of the test that

$$p_{max}^{\mathfrak{S}^c}(\Psi, t) \bowtie p. \tag{7}$$

In contrast, a rejection only holds for the current scheduler and it is impossible to derive an argument for the optimal scheduler. The same holds if hypothesis

$$p(\Psi, \mathfrak{s}_{\min}^{\mathbf{k},n}, t_{\max}) \bowtie p \text{ for } \bowtie \in \{<, \leq\}, \tag{8}$$

is accepted. We then know that indeed

$$p_{min}^{\mathfrak{S}^c}(\Psi, t) \bowtie p \tag{9}$$

holds within the statistical error of the hypothesis test.

Since we use the Azuma test which belongs to the third class of tests, as defined in [43], the statistical error is guaranteed to be bounded by the predefined errors α (false positive) and β (false negative). Hence, we can bound the overall statistical error resulting from discretization, learning, and statistical model checking by α. If hypothesis 6 or 8 is accepted, we conclude with statistical error α that 7 or 9 hold, respectively.

5 CASE STUDIES

We evaluate our approach on two case studies: First, in Section 5.1, we consider a model of a tank, with linear dynamics. We thoroughly discuss the results obtained for different parameter settings. In Section 5.2, we showcase our model of a satellite running different experiments in orbit. It is equipped with a constrained battery that we model by the kinetic battery model, whose continuous dynamics are nonlinear (described by a system of linear ODEs). To emphasize the applicability on different concrete modelling formalisms that instantiate the SHM interface, we implemented the approach in two separately developed tools which work with different models. Hence, for both case studies, we use a HPnG as well as a SHA model, the latter specified in the Modest language. Both tools deliver the same results (up to statistical error), providing some assurance that the implementations are correct.

For the HPnG models, we use the Java tool HYPEG as discrete-event simulator [37] and statistical model checker [41]. We use the state-space discretization proposed for HYPEG in [40] with quantum 0.1, which provides approximate simulations for continuous behaviour described by systems of linear ODEs. Note that the time discretization proposed for non-stiff systems of ODEs in [36] currently does not scale for a large number of training runs. We refer to this implementation of our approach as HYPEG ML. To support the Modest SHA models, we extended the modes tool [4] with support for simulating nonlinear differential equations using the time-discretizing integrators in the Apache Commons Math library. When running our learning framework, our new modes ML always uses this new simulation implementation. We use the 5(4) Dormand-Prince integrator [9] with a min. step of 10^{-4}, max. step of 100, and abs. and rel. tolerance of 10^{-4}. For the detection of events, using root finding, we use convergence 10^{-3} and a maximum check interval of 1 for the tank and 10^{-1} for the satellite. The parameters for Q-learning are set to learning rate $\alpha = 0.1$, discount factor $\gamma = 1$, and epsilon-greedy parameter $\epsilon = 0.15$ for both tools. We indicate the confidence level δ and the width w of the confidence intervals as CI(δ, w). Our experiments were conducted on two similar machines: the HPnG models ran on an Intel Core i5-8250U (4×1.6-3.4 GHz) system, the Modest models on an i5-6600T (4×2.7-3.5 GHz), both with 16 GB memory. Both tools are currently single-threaded only.

5.1 The Linear Tank

The first case study models a linear tank. We compare the results of our new approach to an existing analytic approach for HPnGs with linear evolution. Our water tank model is based on the one from [42] as shown in Figure 2. The maximum height of water in the tank is 20 m and initially set to 4 m. When it reaches 16 m, one of two valves draining the tank is activated nondeterministically. The model has a constant inflow and two controllable linear outflows. One valve reduces the height by 6 m h^{-1} and the other one by 4 m h^{-1}. Once activated, the larger outflow is active for 2 h and the smaller one for 1 h. After being deactivated, the outflow is blocked for a uniformly distributed time between 0 and 6 hours. When the water height rises above 16 and both valves are not blocked, a nondeterministic conflict occurs.

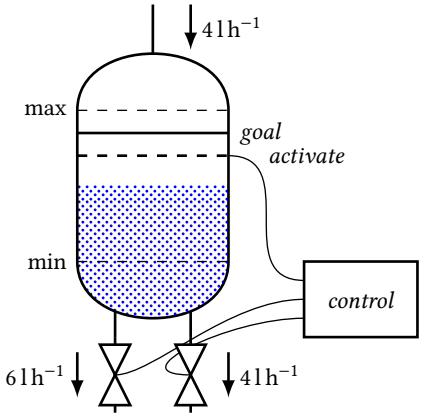

Figure 2: Tank with constant inflow and two controllable outflows [42].

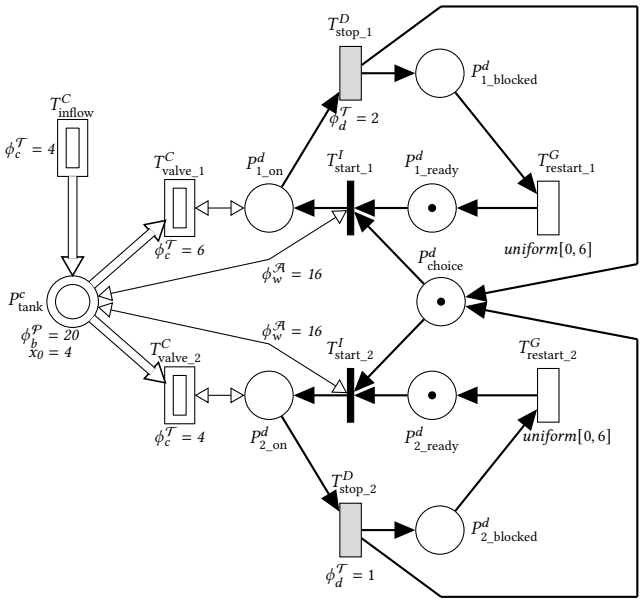

Figure 3: HPnG model describing the tank system.

A visual representation of the corresponding HPnG model is shown in Figure 3. We refer the interested reader to [19] for a thorough introduction to HPnGs. We draw discrete places as (solid) circles and continuous places as double-outline circles. We use solid black bars for immediate, grey filled bars for deterministic, single-outline bars for general and double-outline bars for continuous transitions. Solid arrows represent discrete arcs, double-outline arrows indicate continuous arcs and double arrows test arcs. The fluid in the tank is modeled by the continuous place P^c_{tank}, being

filled by the continuous transition T^C_{inflow}. The two valves are represented by $T^C_{\text{valve\_1}}$ and $T^C_{\text{valve\_2}}$, annotated with the rates of 6 and 4 meters per hour, respectively. The valves get enabled when the corresponding discrete places $P^d_{\text{1\_on}}$ respectively $P^d_{\text{2\_on}}$ hold a token. The two immediate transitions $T^I_{\text{start\_1}}$ and $T^I_{\text{start\_2}}$, together with the token in P^d_{choice}, model a non-deterministic conflict, such that a scheduler can decide which valve is activated. The immediate transitions get enabled when the fluid level of the tank exceeds 16 liters, modeled by the connecting guard arcs. In case of choosing $T^I_{\text{start\_1}}$ and after the tank is drained for 2 h, the deterministic transition $T^D_{\text{stop\_1}}$ moves a token to the place $P^d_{\text{1\_blocked}}$ and the valve is blocked for a random amount of time after deactivation. The time is modeled by the general transition $T^G_{\text{restart\_1}}$. Additionally, $T^I_{\text{stop\_1}}$ puts back a token to P^d_{choice}, such that the second valve can get activated while the first one is still blocked. When $T^G_{\text{restart\_1}}$ fires, a token is moved to $P^d_{\text{1\_ready}}$ and $T^I_{\text{start\_1}}$ can get enabled again. The process of (de-)activating the second valve is similar, where the probability distributions for the general transitions may differ (not indicated in the figure). For comparison, a hybrid automaton modelling the same system is shown in [42, Fig. 5].

We maximize the STL property that the fluid height x_{tank} reaches 18 m before time $t_{\max} \in \{7, \ldots, 11\}$:

$$\Psi = tt\ U^{[0,t_{\max}]}(x_{\text{tank}} \geq 18).^1$$

The probabilities obtained from the learned nonprophetic and prophetic scheduler are compared to analytical results obtained for history-dependent schedulers. While in general memoryless nonprophetic schedulers are less powerful than their history-dependent counterpart, this does not apply in this specific case study, since all previously sampled and unused values are resampled before a conflict occurs. Hence, no relevant history exist.

Validation and comparison. Table 1 shows the results obtained for the linear tank model via HYPEG ML and modes ML for the (non)prophetic case. For further validation, we compare with simulation results for the uniform scheduler $\mathfrak{s}_{\text{unif}}$, and with analytical results according to [38] in the nonprophetic case and [42] in the prophetic case. Both analytical methods are exact up to a numerical error induced by multidimensional integration which is statistically estimated to be at most 10^{-4}. The analytical results lie within the confidence intervals of our simulation-based approach except in the 11 h prophetic scenario, which will be explained later in detail. Using the ad-hoc uniform scheduler leads to significantly lower estimates, which illustrates the benefits of optimizing (non)prophetic schedulers. The runtimes depend on the number of runs (simulation and training, respectively) and the considered time horizon. The runtimes in the analytical approach increase considerably with the time horizon; it is impossible to analytically compute results for larger $t_{\max}$. In contrast, our new ML approach scales better and results for larger $t_{\max}$ can easily be obtained. The training times are not even doubled for HYPEG ML and modes ML. Despite the differences in modelling formalism and the employed solution method for the differential equations, the runtimes of HYPEG ML and modes ML differ only by a factor of ≈ 2, with modes ML being slower here. In particular, both tools scale equally.

[1] We write $x_{\text{tank}} \geq 18$ as short-hand for the function $f(x) = x - c$ in STL.

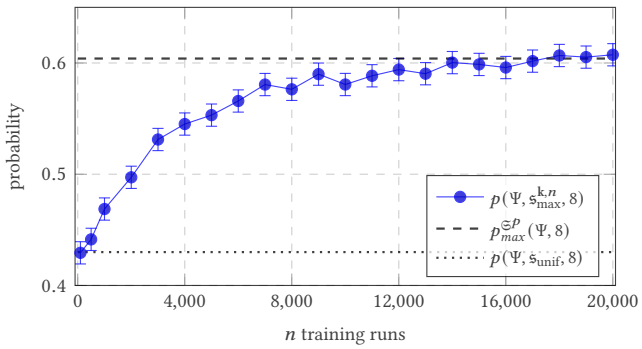

Figure 4: Average probabilities (in blue) with CI(0.99, 0.02), for the 8 h prophetic case in HYPEG ML after an increasing number of training runs (x-axis). The maximum probability (dashed line) is obtained via [42].

Number of training runs. The prophetic case required more training to match the analytical results since the discretized state space, on which decisions are based, is larger than in the nonprophetic case. For Figure 4, we used HYPEG ML to compare the learned prophetic schedulers after different amounts of training runs to the optimal probability achieved by the analytic approach for $t_{\max} = 8$ h. The mean of the confidence intervals after 100 to 20 000 training runs increases with the amount of training from the uniform result until the optimal probability is reached. Note that the mean surpasses the optimal probability for some high numbers of training runs, which can be attributed to the statistical error.

Influence of discretization. Recall that for 11 h the analytical result for the prophetic scheduler lies outside the confidence intervals computed with HYPEG ML and modes ML. This is further investigated in Table 2, which shows the effect of different discretizations on the quality of the learned scheduler and the required number of training runs in HYPEG ML. The results are computed using schedulers obtained after 10^3 to 10^7 training runs. The discretization truncates after $\{0, 1, 2\}$ decimal places. Decisions not encountered during training are resolved by a uniform distribution, possibly resulting in randomized schedulers. This reflects in smaller simulation times (SMC time) for a larger number of training runs, as fewer random decisions have to be taken during individual simulation runs. The results show that it is crucial to find a good combination of discretization and number of training runs: a finer discretization requires more training runs to learn a scheduler which comes close to the optimal one while a coarse discretization may inhibit better results. Note that the mean of the confidence intervals increases with more training runs and the same discretization. The coarsest discretization performs better with fewer training runs but is not able to improve much with more training. A finer discretizations requires more training runs such that all reachable states are explored sufficiently.

Table 1: Results for different amounts of training with uniform and (non)prophetic scheduling. Discretization truncates after the first decimal place and CI(0.99, 0.02).

		$t_{\max}$	7	8	9	10	11
uniform	modes [4]	p	0	0.43	0.60	0.60	0.68
		sim. time	0.3 s	3.0 s	3.1 s	3.3 s	3.1 s
		sim. runs	270	16 230	15 940	15 870	14 330
nonprophetic	HPnG [38]	$p_{\max}$	0	0.5347	0.6264	0.6445	0.7375
		runtime	0.0 s	4.5 s	12.9 s	20.5 s	52.0 s
	HYPEG ML 5000 t. runs	$p_{\max}$	0	0.54	0.63	0.65	0.74
		trainingtime	0.8 s	0.8 s	1.0 s	1.3 s	1.3 s
		sim. time	0.1 s	1.4 s	1.8 s	1.6 s	1.4 s
		sim. runs	1000	16 503	15 468	15 123	12 671
	modes ML 5000 t. runs	$p_{\max}$	0	0.53	0.62	0.64	0.74
		trainingtime	1.0 s	1.1 s	1.1 s	1.3 s	1.4 s
		sim. time	0.2 s	2.7 s	3.1 s	3.3 s	3.0 s
		sim. runs	270	16 510	15 620	15 240	12 790
prophetic	Flowpipe [42]	$p_{\max}$	0	0.6041	0.6488	0.6513	0.7555
		runtime	0.2 s	1.7 s	9.2 s	55.0 s	2289.9 s
	HYPEG ML 20000 t. runs	$p_{\max}$	0	0.60	0.64	0.65	0.72
		trainingtime	1.6 s	1.9 s	2.4 s	2.4 s	2.6 s
		sim. time	0.1 s	1.3 s	1.3 s	1.4 s	1.2 s
		sim. runs	1000	15 921	15 271	15 077	13 346
	modes ML 20000 t. runs	$p_{\max}$	0	0.60	0.64	0.65	0.74
		trainingtime	3.0 s	3.4 s	4.1 s	3.9 s	4.0 s
		sim. time	0.2 s	2.6 s	2.9 s	2.8 s	2.6 s
		sim. runs	270	15 980	15 270	15 040	12 820

Table 2: Average reachability probabilities for prophetic scheduling at $t = 11$ and CI(0.99, 0.02). Training/computation times and number of simulation runs are provided for different numbers of training runs and discretizing at $\{0, 1, 2\}$ decimal places.

training runs		1000	10 000	100 000	1 000 000	10 000 000
HYPEG ML 0 decimal places	mean $p_{\max}$	0.702	0.716	0.719	0.720	0.715
	train time	0.4 s	1.6 s	7.6 s	67.0 s	688.3 s
	SMC time	1.9 s	1.5 s	1.2 s	1.2 s	1.1 s
	SMC runs	13 895	13 494	13 418	13 384	12 564
HYPEG ML 1 decimal place	mean $p_{\max}$	0.700	0.726	0.747	0.748	0.745
	train time	0.4 s	1.5 s	8.2 s	68.0 s	670.9 s
	SMC time	2.0 s	1.4 s	1.2 s	1.1 s	1.1 s
	SMC runs	13 927	13 201	12 540	12 512	12 625
HYPEG ML 2 decimal places	mean $p_{\max}$	0.682	0.686	0.695	0.723	0.739
	train time	0.5 s	1.8 s	8.0 s	68.5 s	679.1 s
	SMC time	2.1 s	1.6 s	1.3 s	1.2 s	1.2 s
	SMC runs	14 386	14 294	14 070	13 285	12 792

Hypothesis testing. Table 3 presents the results of the Azuma[44] hypothesis test for the same number of training runs and discretization settings as in Table 2. The hypothesis checked is

$$P_{\geq 0.73}(tt\ U^{[0,11]}(x_{\text{tank}} \geq 18)).$$

Recall that a rejection only holds for the currently trained scheduler, whereas an accept also holds for the optimal scheduler as discussed in Section 4.2. The training times are similar to Table 2. The Azuma test requires more runs whenever the computed probability lies close to the probability bound to decide whether a given hypothesis holds. In return, the statistical error is bounded by $\alpha = 0.05$ and

Table 3: Hypothesis test whether the probability of the tank being below height 18 until $t = 11$ is greater than 0.73: $P_{\geq 0.73}(tt\ U^{[0,11]}(x_{\mathbf{tank}} \geq 18)$ for a prophetic scheduler. Results, computation times and number of simulation runs are provided for different numbers of training runs, discretizing at $\{0, 1, 2\}$ decimal places, $\alpha = \beta = 0.05$ and guess 0.745.

training runs		1000	10 000	100 000	1 000 000	10 000 000
HYPEG ML 0 decimal places	$P_{\geq 0.73}$	false	false	false	false	false
	train time	0.4 s	1.9 s	8.3 s	68.3 s	651.9 s
	SMC time	215.6 s	663.8 s	758.5 s	1217.9 s	354.7 s
	SMC runs	3 248 519	9 976 274	11 586 698	18 423 552	5 521 225
HYPEG ML 1 decimal place	$P_{\geq 0.73}$	false	no result	true	true	true
	train time	0.4 s	1.6 s	7.8 s	67.1 s	669.4 s
	SMC time	24.5 s	3354.4 s	936.4 s	431.8 s	606.1 s
	SMC runs	354 284	50 000 000	14 101 368	6 566 772	9 147 820
HYPEG ML 2 decimal places	$P_{\geq 0.73}$	false	false	false	no result	true
	train time	0.4 s	1.8 s	8.3 s	69.3 s	682.5 s
	SMC time	10.9 s	12.2 s	18.2 s	3381.7 s	934.6 s
	SMC runs	145 612	174 717	270 462	50 000 000	13 792 904

$\beta = 0.05$ even for small $|p - 0.73|$. We specify a maximum number of 50 million simulation runs, which in some cases is not enough to arrive at a decision.

For the coarsest discretization, the hypothesis test always returns false. In contrast, the hypothesis test for the discretization to one decimal place returns true for the scheduler computed with 100 000 training runs, after not being able to terminate for a scheduler computed with 10 000 runs.

Note that the experiments summarized in Table 3 are carried out as independent computations, which required a varying number of simulation runs. Hence, we performed 3 executions and display the median. The SMC times linearly grow with the required number of SMC runs, which is related to the difference between the computed probability and the probability bound 0.73.

Using a heuristic could help to sequentially identify a parameter setting (discretization and number of training runs), which leads to the hypothesis being accepted. Note that a sequential computation would require to adjust α and β to correct for the multiple tests.

5.2 The Satellite Case Study

As a second case study, we consider an abstract model of a satellite inspired by [2], featuring nonlinear continuous dynamics via its battery. To the best of our knowledge, no other approach can optimize probabilities for such a model.

The satellite orbits the earth and performs experiments. It overflies the ground station every 15 hours; at this point, if it is idle, it can receive two new experiment tasks. One experiment has a high power demand (4 W h) and needs a short (Uniform[1, 4] distributed) amount of time while the other takes longer (Exponential[$\frac{1}{8}$] distributed) but requires less power (0.5 W h). Initially, the satellite is idle and will pass the ground station in 1 h. The satellite has a solar panel which generates 1 W h of power when it is in the sun, which is the case for 12 hours, followed by 12 hours of darkness. Initially the sun shone already 6 h. The satellite has a battery which can be charged and discharged and allows operation when the power demand surpasses the generation of the solar panel. The battery is modeled according to the Kinetic Battery Model (KiBaM) (cf. [27]), which divides the battery in two wells: the available charge a and the bound charge b. The KiBaM incorporates two important nonlinear effects: A battery has less effective capacity if it is drained with a high discharge rate, which is the *rate-capacity effect*. Some usable charge is recovered if not or slowly discharged, which is the *reccovery effect*. The capacity of the wells is given by the total capacity $C = 60$ and $c = 0.5$ such that $a_{\max} = c \cdot C$ and $b_{\max} = (1 - c)C$. The following ODEs describe the evolution of the wells over time, where I is the (dis-)charging of a, and p the flow between the wells:

$$\dot{a}(t) = -I + p\left(\frac{b(t)}{1-c} - \frac{a(t)}{c}\right), \quad \text{and}$$
$$\dot{b}(t) = p\left(\frac{a(t)}{c} - \frac{b(t)}{1-c}\right). \tag{10}$$

Since a battery cannot overflow, the first ODE changes when $a = a_{\max}$ and $-I \geq \dot{b}(t)$ to $\dot{a}(t) = 0$ until I changes sign, while $\dot{b}(t)$ remains the same.

The property of interest is whether the battery is drained before the time horizon $t_{\max} = 30$, i.e. $\Psi = tt\ U^{[0,30]}(a = 0)$. The model nondeterministically schedules the order of the two available tasks after each communication with the ground station, which always induces two new tasks unless the satellite is still busy processing an earlier task.

Table 4 presents maximum and minimum probabilities for the above STL property, obtained with HYPEG ML and modes ML for prophetic, nonprophetic, and uniform scheduling. We see that the results of both tools match well, i.e. the means lie within the respective other's confidence intervals. (Non)prophetic scheduling improves the probabilities w.r.t. uniform scheduling, and prophetic scheduling achieves the lowest/highest probability depending on the optimization goal while nonprophetic scheduling does not reach these probabilities, as expected. In this case study, a single training/simulation run is faster in modes ML by a factor of 2-3 compared to HYPEG ML. This stems partly from the state-space discretization method, which induces more recomputations in the nonlinear case.

Table 4: Average optimal reachability probabilities for (non)prophetic or uniform scheduling at $t = 30$ (2 orbits) in the satellite (CI(0.99, 0.01), discretizing to integers).

		min			max	
		prophetic	nonproph.	uniform	nonproph.	prophetic
HYPEG ML 100 000 t. runs	p	0.333	0.348	0.455	0.567	0.579
	train time	233.9 s	224.6 s	n/a	220.2 s	233.9 s
	SMC time	155.7 s	137.3 s	151.7 s	145.0 s	152.6 s
	SMC runs	58 960	60 186	65 825	64 842	58 960
modes ML 100 000 t. runs	p	0.329	0.345	0.458	0.566	0.583
	train time	93.0 s	95.3 s	n/a	86.9 s	82.8 s
	SMC time	54.4 s	58.9 s	56.4 s	55.3 s	52.2 s
	SMC runs	58 630	59 960	65 880	65 180	64 520

Furthermore, the encoding of the same conceptual model as HPnGs and Modest/SHA may induce further performance variations.

6 CONCLUSION

We introduced reinforcement learning to SHM. We obtain memoryless prophetic or nonprophetic schedulers for discrete nondeterminism via Q-learning, using a discretized state-space view for learning only. We restrict to piecewise constant schedulers and show that the learned scheduler induces an approximation of the optimal probabilities that converges towards the optimum of this scheduler class when making the grid finer and letting the number of training runs go to infinity. Statistical model checking then evaluates the learned schedulers on the original model. We obtained promising results with our approach both on HPnGs and Modest models, building on the existing (non-learning) statistical model checkers for both. The approach can easily be integrated into discrete-event simulators for other stochastic hybrid models. The tradeoff between accuracy and performance prevents identifying a discretization and number of training runs in advance that allows to prove a given hypothesis. Future work will investigate suitable heuristics. We also plan to develop smarter discretizations that make the grid finer at states reached with high probability, and to investigate the optimality of the computed class of memoryless, piecewise constant, deterministic schedulers in the (non)prophetic setting for SHM.

DATA AVAILABILITY

The tools used and data generated in our experimental evaluation as well as instructions to replicate the experiments are archived and available at DOI 10.4121/16635823 [35].

ACKNOWLEDGMENTS

This work was in part supported by NWO VENI grant no. 639.021.754.

REFERENCES

[1] Alessandro Abate, Joost-Pieter Katoen, John Lygeros, and Maria Prandini. 2010. Approximate Model Checking of Stochastic Hybrid Systems. *European Journal of Control* 16, 6 (Jan. 2010), 624–641. https://doi.org/10.3166/ejc.16.624-641

[2] Morten Bisgaard, David Gerhardt, Holger Hermanns, Jan Krčál, Gilles Nies, and Marvin Stenger. 2019. Battery-aware scheduling in low orbit: the GomX–3 case. *Form Asp Comp* 31, 2 (April 2019), 261–285. https://doi.org/10.1007/s00165-018-0458-2

[3] Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Vojtěch Forejt, Jan Křetínský, Marta Kwiatkowska, David Parker, and Mateusz Ujma. 2014. Verification of Markov Decision Processes Using Learning Algorithms. In *Automated Technology for Verification and Analysis*. Vol. 8837. Springer International Publishing, Cham, 98–114. https://doi.org/10.1007/978-3-319-11936-6_8 Series Title: Lecture Notes in Computer Science.

[4] Carlos E. Budde, Pedro R. D'Argenio, Arnd Hartmanns, and Sean Sedwards. 2020. An efficient statistical model checker for nondeterminism and rare events. *Int J Softw Tools Technol Transfer* 22, 6 (Dec. 2020), 759–780. https://doi.org/10.1007/s10009-020-00563-2

[5] Mingyu Cai, Hao Peng, Zhijun Li, and Zhen Kan. 2021. Learning-Based Probabilistic LTL Motion Planning With Environment and Motion Uncertainties. *IEEE Trans. Automat. Contr.* 66, 5 (May 2021), 2386–2392. https://doi.org/10.1109/TAC.2020.3006967

[6] Nathalie Cauchi and Alessandro Abate. 2019. StocHy: Automated Verification and Synthesis of Stochastic Processes. In *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 11428. Springer International Publishing, Cham, 247–264. https://doi.org/10.1007/978-3-030-17465-1_14 Series Title: Lecture Notes in Computer Science.

[7] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. 2015. Uppaal SMC tutorial. *Int J Softw Tools Technol Transfer* 17, 4 (Aug. 2015), 397–415. https://doi.org/10.1007/s10009-014-0361-y

[8] René David and Hassane Alla. 2005. *Discrete, continuous, and hybrid Petri Nets* (1st ed.). Springer, Berlin, Heidelberg.

[9] J. R. Dormand and P. J. Prince. 1980. A family of embedded Runge-Kutta formulae. *J. Comput. Appl. Math.* 6, 1 (1980), 19–26.

[10] Pedro D'Argenio, Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. 2015. Smart sampling for lightweight verification of Markov decision processes. *Int J Softw Tools Technol Transfer* 17, 4 (Aug. 2015), 469–484. https://doi.org/10.1007/s10009-015-0383-0

[11] Pedro R. D'Argenio, Marcus Gerhold, Arnd Hartmanns, and Sean Sedwards. 2018. A Hierarchy of Scheduler Classes for Stochastic Automata. In *Foundations of Software Science and Computation Structures*. Vol. 10803. Springer International Publishing, Cham, 384–402. https://doi.org/10.1007/978-3-319-89366-2_21 Series Title: Lecture Notes in Computer Science.

[12] Pedro R. D'Argenio, Arnd Hartmanns, and Sean Sedwards. 2018. Lightweight Statistical Model Checking in Nondeterministic Continuous Time. In *Leveraging Applications of Formal Methods, Verification and Validation. Verification*. Vol. 11245. Springer International Publishing, Cham, 336–353. https://doi.org/10.1007/978-3-030-03421-4_22 Series Title: Lecture Notes in Computer Science.

[13] Christian Ellen, Sebastian Gerwinn, and Martin Fränzle. 2015. Statistical model checking for stochastic hybrid systems involving nondeterminism over continuous domains. *Int J Softw Tools Technol Transfer* 17, 4 (Aug. 2015), 485–504. https://doi.org/10.1007/s10009-014-0329-y

[14] Martin Fränzle, Ernst Moritz Hahn, Holger Hermanns, Nicolás Wolovick, and Lijun Zhang. 2011. Measurability and Safety Verification for Stochastic Hybrid Systems. In *Proceedings of the 14th international conference on Hybrid systems: computation and control - HSCC '11*. ACM Press, Chicago, IL, USA, 43. https://doi.org/10.1145/1967701.1967710

[15] Martin Fränzle, Tino Teige, and Andreas Eggers. 2010. Engineering constraint solvers for automatic analysis of probabilistic hybrid automata. *The Journal of Logic and Algebraic Programming* 79, 7 (Oct. 2010), 436–466. https://doi.org/10.1016/j.jlap.2010.07.003

[16] Nathan Fulton, Nathan Hunt, Nghia Hoang, and Subhro Das. 2020. Formal Verification of End-to-End Learning in Cyber-Physical Systems: Progress and

Challenges. *arXiv:2006.09181 [cs, stat]* (June 2020). http://arxiv.org/abs/2006.09181 arXiv: 2006.09181.

[17] Yang Gao and Martin Fränzle. 2015. A Solving Procedure for Stochastic Satisfiability Modulo Theories with Continuous Domain. In *Quantitative Evaluation of Systems*. Vol. 9259. Springer International Publishing, Cham, 295–311. https://doi.org/10.1007/978-3-319-22264-6_19 Series Title: Lecture Notes in Computer Science.

[18] P. Geibel and F. Wysotzki. 2005. Risk-Sensitive Reinforcement Learning Applied to Control under Constraints. *jair* 24 (July 2005), 81–108. https://doi.org/10.1613/jair.1666

[19] Marco Gribaudo and Anne Remke. 2016. Hybrid Petri nets with general one-shot transitions. *Performance Evaluation* 105 (Nov. 2016), 22–50. https://doi.org/10.1016/j.peva.2016.09.002

[20] Ernst Moritz Hahn, Arnd Hartmanns, Holger Hermanns, and Joost-Pieter Katoen. 2013. A Compositional Modelling and Analysis Framework for Stochastic Hybrid Systems. *Form Methods Syst Des* 43, 2 (Oct. 2013), 191–232. https://doi.org/10.1007/s10703-012-0167-z

[21] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. 2020. Faithful and Effective Reward Schemes for Model-Free Reinforcement Learning of Omega-Regular Objectives. In *Automated Technology for Verification and Analysis*. Vol. 12302. Springer International Publishing, Cham, 108–124. https://doi.org/10.1007/978-3-030-59152-6_6 Series Title: Lecture Notes in Computer Science.

[22] Arnd Hartmanns and Holger Hermanns. 2014. The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification. In *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 8413. Springer Berlin Heidelberg, Berlin, Heidelberg, 593–598. https://doi.org/10.1007/978-3-642-54862-8_51 Series Title: Lecture Notes in Computer Science.

[23] Arnd Hartmanns, Holger Hermanns, and Jan Krčál. 2016. Schedulers are no Prophets. In *Semantics, Logics, and Calculi*. Vol. 9560. Springer International Publishing, Cham, 214–235. https://doi.org/10.1007/978-3-319-27810-0_11 Series Title: Lecture Notes in Computer Science.

[24] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. 2020. Cautious Reinforcement Learning with Logical Constraints. (2020), 483–491. https://dl.acm.org/doi/abs/10.5555/3398761.3398821

[25] M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, and I. Lee. 2019. Reinforcement Learning for Temporal Logic Control Synthesis with Probabilistic Satisfaction Guarantees. In *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, Nice, France, 5338–5343. https://doi.org/10.1109/CDC40024.2019.9028919

[26] David Henriques, Joao G. Martins, Paolo Zuliani, Andre Platzer, and Edmund M. Clarke. 2012. Statistical Model Checking for Markov Decision Processes. In *2012 Ninth International Conference on Quantitative Evaluation of Systems*. IEEE, London, United Kingdom, 84–93. https://doi.org/10.1109/QEST.2012.19

[27] Holger Hermanns, Jan Krčál, and Gilles Nies. 2015. Recharging Probably Keeps Batteries Alive. In *Cyber Physical Systems. Design, Modeling, and Evaluation*. Vol. 9361. Springer International Publishing, Cham, 83–98. https://doi.org/10.1007/978-3-319-25141-7_7 Series Title: Lecture Notes in Computer Science.

[28] Sebastian Junges, Nils Jansen, Joost-Pieter Katoen, Ufuk Topcu, Ruohan Zhang, and Mary Hayhoe. 2018. Model Checking for Safe Navigation Among Humans. In *Quantitative Evaluation of Systems*. Vol. 11024. Springer International Publishing, Cham, 207–222. https://doi.org/10.1007/978-3-319-99154-2_13 Series Title: Lecture Notes in Computer Science.

[29] Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Computer Aided Verification*. Vol. 6806. Springer Berlin Heidelberg, Berlin, Heidelberg, 585–591. https://doi.org/10.1007/978-3-642-22110-1_47 Series Title: Lecture Notes in Computer Science.

[30] Luca Laurenti, Morteza Lahijanian, Alessandro Abate, Luca Cardelli, and Marta Kwiatkowska. 2021. Formal and Efficient Synthesis for Continuous-Time Linear Stochastic Hybrid Processes. *IEEE Trans. Automat. Contr.* 66, 1 (Jan. 2021), 17–32. https://doi.org/10.1109/TAC.2020.2975028

[31] Abolfazl Lavaei, Fabio Somenzi, Sadegh Soudjani, Ashutosh Trivedi, and Majid Zamani. 2020. Formal Controller Synthesis for Continuous-Space MDPs via Model-Free Reinforcement Learning. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, Sydney, Australia, 98–107. https://doi.org/10.1109/ICCPS48487.2020.00017

[32] Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. 2015. Scalable Verification of Markov Decision Processes. In *Software Engineering and Formal Methods*. Vol. 8938. Springer International Publishing, Cham, 350–362. https://doi.org/10.1007/978-3-319-15201-1_23 Series Title: Lecture Notes in Computer Science.

[33] Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Souradeep Dutta, Taylor J. Carpenter, Radoslav Ivanov, and Taylor T. Johnson. 2019. ARCH-COMP19 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants. In *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems (EPiC Series in Computing, Vol. 61)*. EasyChair, 103–119.

[34] Oded Maler and Dejan Nickovic. 2004. Monitoring Temporal Properties of Continuous Signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Vol. 3253. Springer Berlin Heidelberg, Berlin, Heidelberg, 152–166. https://doi.org/10.1007/978-3-540-30206-3_12 Series Title: Lecture Notes in Computer Science.

[35] Mathis Niehage, Arnd Hartmanns, and Anne Remke. 2021. Learning Optimal Decisions for Stochastic Hybrid Systems (Artifact). 4TU.ResearchData. https://doi.org/10.4121/16635823

[36] Mathis Niehage, Carina Pilch, and Anne Remke. 2020. Simulating Hybrid Petri nets with general transitions and non-linear differential equations. In *Proceedings of the 13th EAI International Conference on Performance Evaluation Methodologies and Tools*. ACM, Tsukuba Japan, 88–95. https://doi.org/10.1145/3388831.3388842

[37] Carina Pilch, Fabian Edenfeld, and Anne Remke. 2017. HYPEG: Statistical Model Checking for hybrid Petri nets: Tool Paper. In *Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools - VALUETOOLS 2017*. ACM Press, Venice, Italy, 186–191. https://doi.org/10.1145/3150928.3150956

[38] Carina Pilch, Arnd Hartmanns, and Anne Remke. 2020. Classic and Non-Prophetic Model Checking for Hybrid Petri Nets with Stochastic Firings. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*. ACM, Sydney New South Wales Australia, 1–11. https://doi.org/10.1145/3365365.3382198

[39] Carina Pilch, Maurice Krause, Anne Remke, and Erika Ábrahám. 2020. A Transformation of Hybrid Petri Nets with Stochastic Firings into a Subclass of Stochastic Hybrid Automata. In *NASA Formal Methods*. Vol. 12229. Springer International Publishing, Cham, 381–400. https://doi.org/10.1007/978-3-030-55754-6_23 Series Title: Lecture Notes in Computer Science.

[40] Carina Pilch, Mathis Niehage, and Anne Remke. 2018. HPnGs go Non-Linear: Statistical Dependability Evaluation of Battery-Powered Systems. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, Milwaukee, WI, 157–169. https://doi.org/10.1109/MASCOTS.2018.00024

[41] Carina Pilch and Anne Remke. 2017. Statistical Model Checking for Hybrid Petri Nets with Multiple General Transitions. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, Denver, CO, USA, 475–486. https://doi.org/10.1109/DSN.2017.41

[42] Carina Pilch, Stefan Schupp, and Anne Remke. 2021. Optimizing Reachability Probabilities for a Restricted Class of Stochastic Hybrid Automata via Flowpipe-Construction. In *Quantitative Evaluation of Systems*. Vol. 12846. Springer International Publishing, Cham, 435–456. https://doi.org/10.1007/978-3-030-85172-9_23 Series Title: Lecture Notes in Computer Science.

[43] Daniël Reijsbergen, Pieter-Tjerk de Boer, Werner Scheinhardt, and Boudewijn Haverkort. 2015. On hypothesis testing for statistical model checking. *Int J Softw Tools Technol Transfer* 17, 4 (Aug. 2015), 377–395. https://doi.org/10.1007/s10009-014-0350-1

[44] Daniël Reijsbergen, Werner Scheinhardt, and Pieter-Tjerk de Boer. 2015. A sequential hypothesis test based on a generalized Azuma inequality. *Statistics & Probability Letters* 97 (Feb. 2015), 192–196. https://doi.org/10.1016/j.spl.2014.11.018

[45] Dorsa Sadigh, Eric S. Kim, Samuel Coogan, S. Shankar Sastry, and Sanjit A. Seshia. 2014. A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In *53rd IEEE Conference on Decision and Control*. IEEE, Los Angeles, CA, USA, 1091–1096. https://doi.org/10.1109/CDC.2014.7039527

[46] Fedor Shmarov and Paolo Zuliani. 2015. ProbReach: verified probabilistic delta-reachability for stochastic hybrid systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. ACM, Seattle Washington, 134–139. https://doi.org/10.1145/2728606.2728625

[47] Fedor Shmarov and Paolo Zuliani. 2016. Probabilistic Hybrid Systems Verification via SMT and Monte Carlo Techniques. In *Hardware and Software: Verification and Testing*. Vol. 10028. Springer International Publishing, Cham, 152–168. https://doi.org/10.1007/978-3-319-49052-6_10 Series Title: Lecture Notes in Computer Science.

[48] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement learning: an introduction* (second edition ed.). The MIT Press, Cambridge, Massachusetts.