The Best of Both Worlds: Analytically-Guided Simulation of HPnGs for Optimal Reachability

Mathis Niehage(✉) and Anne Remke

Westfälische Wilhelms-Universität, 48149 Münster, Germany
{mathis.niehage,anne.remke}@uni-muenster.de

Abstract. Efficient reachability analysis, as well as statistical model checking have been proposed for the evaluation of Hybrid Petri nets with general transitions (HPnG). Both have different (dis-)advantages. The performance of statistical simulation suffers in large models and the number of required simulation runs to achieve a relatively small confidence interval increases considerably. The approach introduced for analytical reachability analysis of HPnGs however, becomes infeasible for a large number of random variables. To overcome these limitations, this paper applies statistical simulation for optimal reachability defined as *until* property in Stochastic Time Logic to a pre-computed symbolic state-space representation of HPnGs, i.e., the Parametric Location Tree (PLT), which has previously been used for model checking HPnGs. A case study on a water tank model shows the feasiblity of the approach and illustrates its advantages w.r.t. the original simulation and analysis approaches.

Keywords: Statistical simulation · State-space representation · Hybrid Petri nets with general transitions

1 Introduction

The dependability of critical infrastructures like water and energy distribution is of utmost importance to society and industry. Improving their dependability requires the efficient evaluation of complex models that combine discrete and continuous behaviour with stochasticity to model the duration of failure and repair times. Such models are so-called stochastic hybrid models (SHM) and a large variety of such models exists with different expressivity. Hybrid Petri nets with general transitions (HPnGs) [17] extend classical Petri nets by introducing continuous places and transitions, thus allowing to model hybrid behaviour, as well as general transitions whose firing time is distributed according to a probability density function. HPnGs have proven to be very useful for evaluating dependability of critical infrastructures [15,16] and (dis-)charging strategies of battery storage, e.g., in smart homes and electric vehicles [34,36].

E. Kalyvianaki and M. Paolieri (Eds.): VALUETOOLS 2023, LNICST 539, pp. 61–81, 2024.
https://doi.org/10.1007/978-3-031-48885-6_5

Nondeterminism is often used to model concurrency or to account for underspecification in the model. As a modeling feature, nondeterminism is traditionally present in automata models and preserved in most (non-stochastic) hybrid automata formalisms. In contrast, stochastic Petri net models often resolve nondeterminism through probabilistic choices [6], as the combination of nondeterminism and stochastic behaviour poses a serious challenge when evaluating such models. Existing analytical approaches for the computation of reachability probabilities in SHM heavily rely on discretization and over-approximation. They either discretize the state-space [1,12] or the support of random variables [18]. All analytical approaches face the problem of state-space explosion, when increasing the number of random variables [7,44], the number of continuous variables or the number of nondeterministic decisions in the model. Simulative approaches rarely address the optimal resolution of nondeterminism, especially for SHM, as they suffer from high computational costs when attempting to identify optimal schedulers due to the need to exhaustively explore the state space.

The contribution of this paper is an analytically-guided simulation for HPnGs. We pursue a simulation-based approach, which builds on a pre-computed symbolic state-space representation as PLT [27]. The PLT has previously been used for model checking HPnGs analytically [25]. Instead, we efficiently simulate paths through the PLT, using the information stored symbolically in every node of the PLT to speed up simulation. We show that a bijective function exists, which maps each path in the HPnG to a path in the PLT (and vice-versa). We further provide algorithms for analytically-guided simulation that allow us to optimize reachability probabilities defined as *until* property in Stochastic Time Logic in HPnGs for different scheduler classes, by restricting the information available to the scheduler. The advantage of the analytically-guided simulation is illustrated with respect to existing statistical and analytical methods in a case study.

Related work. For stochastic automata a hierarchy of scheduler classes has been defined in [9], which concludes that history-dependent prophetic schedulers (c.f. [21]) form the most powerful scheduler class for this modeling formalism. The inherent discrete nondeterminism in HPnGs has previously been resolved analytically in [36] for two scheduler classes, namely (i) history-dependent prophetic and (ii) discrete-history non-prophetic. Recently, Q-learning has been proposed to learn optimal memoryless (non-)prophetic schedulers in SHMs [32]. Together with formal guarantees provided by deductive verification, HPnGs have been used to analyze performability for provably safe SHM [2].

Statistical model checking has been proposed for different kinds of (hybrid) stochastic systems. For Hybrid Petri nets, statistical model checking has been proposed for linear evolutions in [35,38] and for non-linear evolutions in [33,38]. While the Modest Toolset's [20] modes simulator [4] supports SHM with linear dynamics as well as lightweight scheduler sampling [30] to approximate optimal schedulers, it provides the latter only for non-hybrid models [8,10]. Additionally, the Q-learning approach presented in [32] has also been integrated into the

modes simulator. However, it also suffers from high computational costs for large numbers of required training runs.

The term *guided simulation* is also used in rare-event simulation. In importance sampling, the guidance lies in modifying probability distributions to enhance the probability of certain events. In [28] a symbolic analysis is combined with simulation for stochastic priced timed automata in UPPAAL SMC. Importance splitting guides the simulation by stopping and branching simulation runs based on an importance function, e.g. [3,45]. In contrast, our approach modifies neither the distributions, nor the simulation runs, but is guided through a symbolic state-space representation.

Analytical approaches have various limitations: A recently proposed discretization in time and space is for switched diffusions only [29]. Pilch et al. [36] cannot solve for complex prophetic schedulers. HPnGs form a restricted subclass of SHMs, which can be used as a high-level formalism of singular automata with random clocks (c.f. [37]). Nondeterminism has been optimized for the resulting model class via discrete-history non-prophetic and history-dependent prophetic schedulers in [40,44]. prohver [18] provides upper (lower) bounds on maximum (minimum) probabilities only. Delta reachability in ProbReach [42] does not support nondeterminism. Stochastic SMT solving [13,14] 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.

Learning for SHMs is considered, e.g., in ProbReach, which applies the cross-entropy method for resolving nondeterminism [43]. Also [11] enriches an SMT-based approach with decision trees and AI planning algorithms to handle nondeterminism. Reinforcement learning has also been used on MDPs to improve performance and safety (e.g. [23]), as well as to deal with *unknown* stochastic behavior [5,22], and with linear-time logic specifications (e.g. [5,19,41]).

Outline. The paper is further organized as follows. Section 2 repeats the syntax and semantics of HPnGs. Section 3 explains the computation of the PLT, Sect. 4 introduces the analytically-guided simulation and Sect. 5 illustrates the feasibility of our approach. Section 6 concludes the paper.

2 Hybrid Petri Nets with General Transitions

Hybrid Petri nets with general transitions (HPnG) [17] are a high-level formalism for a restricted class of SHMs. Next to continuous and discrete variables, HPnGs include arbitrarily distributed random variables. In the following, HPnGs are defined as in [27,39] including multiple firings of general transitions.

A HPnG consists of places, transitions and arcs which are gathered in a tuple $\mathcal{H} = (\mathcal{P}, \mathcal{T}, \mathcal{A}, \mathbf{M_0}, \Phi)$ together with the initial marking and a tuple of parameter functions specifying the components of the HPnG.

The set of places $\mathcal{P} = \mathcal{P}^d \uplus \mathcal{P}^c$ contains the disjoint finite sets of discrete and continuous places. Continuous places $P_i^c \in \mathcal{P}^c$ contain a continuous marking

$x_i \in \mathbb{R}_0^+$. The parameter function $\Phi_b^{\mathcal{P}} : \mathcal{P}^c \to \mathbb{R}^+ \cup \infty$ assigns each continuous place a possibly infinite upper boundary. In contrast, discrete places $P_i^d \in \mathcal{P}^d$ contain a discrete marking $m_i \in \mathbb{N}_0$ and can not be bounded. The initial marking for all places is given by $\mathbf{M_0}$. Transitions change the markings of the places as follows: The finite set of transitions $\mathcal{T}$ consists of immediate $\mathcal{T}^I$, deterministic $\mathcal{T}^D$ and general $\mathcal{T}^G$ transitions which change the discrete marking and continuous transitions $\mathcal{T}^C$ which change the continuous marking.

Immediate transitions fire as soon as they are enabled. We refer to [17] for a detailed discussion of *enabling*. Deterministic transitions have a deterministic firing time, specified by the parameter function $\phi_d^{\mathcal{T}} : \mathcal{T}^D \to \mathbb{R}^+$. We assign a random variable which follows a cumulative absolute continuous probability distribution function assigned by $\phi_g^{\mathcal{T}}$ to every general transition. Each deterministic and general transition has an associated clock which runs while the transition is enabled. Upon firing, the clock value equals the firing time and it is reset. Continuous transitions $T \in \mathcal{T}^C$ have a constant nominal flow rate assigned by $\phi_f^{\mathcal{T}} : \mathcal{T}^C \to \mathbb{R}^+$.

Transitions and places are connected via the finite set of arcs $\mathcal{A}$. There are two types of arcs: (i) arcs defining which transitions modify the marking of certain places and (ii) arcs enabling transitions based on the marking of the connected places. Arcs of the first type are discrete $\mathcal{A}^d$ and continuous $\mathcal{A}^f$ arcs. Discrete places and discrete transitions are connected via discrete arcs and continuous transitions and continuous places by continuous arcs, respectively. An *input* place has an arc towards the transition and the marking of the place decreases when the transition fires, an arc from the transition to the *output* place increases the marking. Each arc is assigned a weight $\phi_w^{\mathcal{A}}$ which weights the change of marking when firing the transition. If a continuous place reaches one of its boundaries, rate adaption reduces the actual flow rates of the affected continuous transitions based on the share $\phi_s^{\mathcal{A}}$ and priority $\phi_p^{\mathcal{A}}$ of the corresponding continuous arcs to prevent under- or overflow of that place. The second type of arcs are guard arcs which consist of test $\mathcal{A}^t$ and inhibitor $\mathcal{A}^h$ arcs. Guard arcs enable and disable connected transitions based on the marking of the connected places. Test arcs enable the transition if the marking is larger or equal to the weight of the guard arc. Inhibitor arcs are antithetic as the transition is enabled if the marking is lower than the weight. Note that discrete transitions also require that their input places match the weight of the connecting arcs in order to be enabled.

Definition 1. The state of a HPnG $\Gamma = (\mathbf{m}, \mathbf{x}, \mathbf{c}, \mathbf{d}, \mathbf{g}, \mathbf{e})$ contains all information of the variables in the HPnG. The discrete marking of all discrete places is in the vector $\mathbf{m}$ and the continuous marking of all continuous places in the vector $\mathbf{x}$. The current clock values of the deterministic transitions are in the vector $\mathbf{c}$ and the clock values of the general transitions are in the vector $\mathbf{g}$. The drift of all continuous places $\mathbf{d}$ describes the current change of fluid for each place and the (boolean) enabling status of all transitions is contained in the vector $\mathbf{e}$.

The set of all states is denoted as $\mathcal{S}$. The drift and enabling status are included in the definition of a state to simplify later computations, but can be derived

uniquely from the given HPnG and the marking of the state. The initial state of the HPnG is defined as $\Gamma_0 = (\mathbf{m}_0, \mathbf{x}_0, \mathbf{0}, \mathbf{d}_0, \mathbf{0}, \mathbf{e}_0)$, where $\mathbf{m}_0$ and $\mathbf{x}_0$ are given by the initial marking $\mathbf{M}_0$, and $\mathbf{d}_0$ as well as $\mathbf{e}_0$ are derived from it.

Definition 2. (Event). An event $\Upsilon(\Gamma_i, \Gamma_{i+1}) = (\Delta\tau_{i+1}, \epsilon_{i+1})$ changes state Γ_i to state Γ_{i+1}. The model element that caused the event is given by $\epsilon_{i+1} \in \mathcal{P}^c \cup \mathcal{T} \setminus \mathcal{T}^C \cup \mathcal{A}^T$ and $\Delta\tau_{i+1}$ is the relative time from the previous event such that

1. a discrete transition fires and the discrete marking changes ($\epsilon_{i+1} \in \mathcal{T} \setminus \mathcal{T}^C$),
2. a lower or upper boundary is reached by a continuous place ($\epsilon_{i+1} \in \mathcal{P}^c$),
3. the fluid of a continuous place changes, such that a guard arc condition is now fulfilled or stops being fulfilled ($\epsilon_{i+1} \in \mathcal{A}^T$).

The first event type relates to a change of the discrete marking, the second changes the drift. The third type can have different consequences: If the enabling status of a continuous transition changes, the drift of at least one place also changes. If the enabling status of a discrete transition changes, initially only the enabling vector changes. This potentially influences the next event. We denote the set of events from a state $\Gamma_i \in S$ as $\mathcal{E}(\Gamma_i) = \{\Upsilon(\Gamma_i, \Gamma_{i+1})\}$ and its subset containing events with the minimum remaining time as:

$$\mathcal{E}^{min}(\Gamma_i) = \{\Upsilon(\Gamma_i, \Gamma_{i+1}) \in \mathcal{E}(\Gamma_i) \mid \nexists\{\Upsilon(\Gamma_i, \Gamma_j) \in \mathcal{E}(\Gamma_i) \colon \Delta\tau_j < \Delta\tau_{i+1}\}. \quad (1)$$

There may be multiple events with the same remaining time, i.e., $|\mathcal{E}^{min}(\Gamma_i)| > 1$.

Following [36], conflicts of discrete transition firings of the same type are considered as conflict and the existing order of events from [17] is maintained. Hence, we are only interested in conflicts between immediate and deterministic transitions, since the probability of multiple general transitions firings at the same time equals zero since their distributions are absolute continuous. Note that events caused by over- or underflow are not considered a conflict as rate adaption is executed after the firing of the transitions. Conflicts can be resolved, e.g., via weights [17] or using schedulers [32,36]. Events provide a symbolic partitioning of the infinite state-space into a finite number of sets. Within each set, the discrete part of the state and the drift of the continuous variables is constant. One time-bounded execution of the Zeno-free [24] model is represented by a finite path.

Definition 3. (Finite Path). A finite path with length $n \in \mathbb{N}$ is an alternating sequence of states and events

$$\sigma = \Gamma_0 \xrightarrow{(\Delta\tau_0, \epsilon_0)} \cdots \xrightarrow{(\Delta\tau_{n-1}, \epsilon_{n-1})} \Gamma_n,$$

where for every $i \in \{0, \ldots, n-1\}$ an event $\Upsilon(\Gamma_i, \Gamma_{i+1}) = (\Delta\tau_i, \epsilon_i) \in \mathcal{E}^{min}(\Gamma_i)$ exists. The set of all finite paths with arbitrary lengths $n \in \mathbb{N}$ starting from the initial state Γ_0 is denoted as $Paths_{\mathcal{H}}(\Gamma_0)$.

3 Partitioning the State Space

The parametric location tree (PLT) partitions the infinite state-space of the HPnG into a finite subset of parametric locations. The presentation of the PLT is based on [27]. The PLT consists of nodes and transitions, where the nodes are parametric locations which represent a set of uncountably many states of the HPnG and the root node includes the initial state. Each edge represents an event, such that the respective child locations include the entry states after executing the event and all states until the occurrence of the next event. Multiple child locations are possible, due to nondeterministic decisions or due to random variables. The number of parametric locations is finite for Zeno-free models, as only time-bounded properties are considered.

Definition 4. (Parametric Location [27])**.** A parametric location is a tuple $\Lambda = (ID, t, p, \Gamma, \mathbf{S}, \mathbf{o})$ with identifier $\Lambda.ID$. The possible firing times of previous general transition firings are included in the potential domain of the random variables $\Lambda.\mathbf{S}$ and the firing order of general transitions is stored as vector $\Lambda.\mathbf{o}$, which is unique in a given parametric location. If a conflict is resolved probabilistically, the conflict probability is given by $\Lambda.p$ which is derived from the weights of the conflicting transitions. The entry time $\Lambda.t : \Lambda.\mathbf{S} \rightarrow \mathbb{R}_0^+$ is a linear function of the firing times of previous general transition firings and the state of the HPnG $\Lambda.\Gamma : \Lambda.\mathbf{S} \rightarrow \mathcal{S}$ is a linear function describing the model state at the entry time in dependence of the previous general transition firings. The valuations of the continuous variables and clocks are, as the entry time, linear functions of the firing times. The discrete markings, the drift and the enabling status is constant.

The PLT orders the parametric locations starting from the root node, which includes the initial state, child nodes are determined via the possible subsequent events of the parent location.

Definition 5. (Parametric Location Tree (PLT) [27])**.** The PLT $(\mathbf{V}, \mathbf{E}, \Lambda_0)$ is a tree with a finite set of parametric locations $\mathbf{V}$. The finite set of edges $\mathbf{E} \subset \mathbf{V} \times \mathbf{V}$ contains an edge $e_i = (\Lambda_j, \Lambda_k) \in \mathbf{E}$ for each model element that may cause an event $M(\Lambda_j) = \{\epsilon \mid \exists \Delta\tau \in \mathbb{R}_0^+, \exists \mathbf{f}_j \in \Lambda_j.\mathbf{S} : (\Delta\tau, \epsilon) \in \mathcal{E}^{min}(\Lambda_j.\Gamma(\mathbf{f}_j))\}$. The root node of the tree is the initial location $\Lambda_0 = (0, 0, 1, \Gamma_0, \mathbf{S}^\infty, \mathbf{o}_0)$, where Γ_0 is the initial state, $\mathbf{S}^\infty$ is the not yet limited domain of all enabled random transitions in the initial state and $\mathbf{o}_0$ is the empty vector.

We extend the definition of child locations from [27] by the set of entry times.

Definition 6. (Child Locations). Given a PLT $(\mathbf{V}, \mathbf{E}, \Lambda_0)$ with parametric location $\Lambda \in \mathbf{V}$, the child locations of Λ are collected in the set $\mathbf{\Lambda}_c(\Lambda) = \{\Lambda_i | (\Lambda, \Lambda_i) \in \mathbf{E}\}$. The set $\Theta = \{\Lambda_i.t | \Lambda_i \in \mathbf{\Lambda_c}(\Lambda)\}$ contains all entry times of the child locations. The subset $\mathbf{\Lambda}_T(\Lambda) \subseteq \mathbf{\Lambda}_c(\Lambda)$ contains all child locations where the event corresponds to the firing of a non-general transition $\mathbf{\Lambda}_T(\Lambda) = \{\Lambda_i \in \mathbf{\Lambda}_c(\Lambda) || \Lambda.\mathbf{o}| = |\Lambda_i.\mathbf{o}|\}$.

Conflicts of transitions, as presented in Sect. 2, correspond to conflicts of parametric locations in the PLT. Conflicts have been defined in [37] via non-empty intersections of potential domains. In contrast our definition is based on the entry times of the child locations and is well-suited for simulation.

Definition 7. (Nondeterministic Conflict in PLT). Given a PLT $(\mathbf{V}, \mathbf{E}, \Lambda_0)$ and a parametric location $\Lambda \in \mathbf{V}$ with child locations $\mathbf{\Lambda}_c(\Lambda)$, for each entry time $t_e \in \Theta$, the set $\mathbf{\Lambda}_{t_e} = \{\Lambda_i \in \mathbf{\Lambda}_T(\Lambda) | \Lambda_i.t = t_e\}$ contains all parametric locations possibly reached if an event occurs at t_e. A nondeterministic conflict occurs, if $|\mathbf{\Lambda}_{t_e}| > 1$, and the corresponding conflict set of transitions is $C^{\mathcal{T}}_{t_e}(\Lambda) = \{T \in \mathcal{T} | \exists \Lambda_i \in \mathbf{\Lambda}_{t_e}, \exists \Delta\tau \in \mathbb{R}^+_0, \exists \mathbf{s} \in \Lambda.S, \exists \mathbf{s}_i \in \Lambda_i.S : \Upsilon(\Lambda.\Gamma(\mathbf{s}), \Lambda_i.\Gamma(\mathbf{s}_i)) = (\Delta\tau, T)\}$.

Note that potentially multiple conflict sets exist for one parent location. In addition to [27], we add a definition of paths through the PLT and their properties.

Definition 8. (Path through PLT). A path through a PLT $(\mathbf{V}, \mathbf{E}, \Lambda_0)$ is an alternating sequence of tuples $(\Lambda_i, \mathbf{f_i}) \in V \times (\mathbb{R}^+_0)^{|\Lambda_i.o|}$ for $i \in \{0, \ldots, n\}$ and model elements $\epsilon_j \in \mathcal{P}^c \cup \mathcal{T} \setminus \mathcal{T}^C \cup \mathcal{A}^T$ causing an event, for $j \in \{0, \ldots, n-1\}$. The vector $\mathbf{f_i} \in \Lambda_i.S$ contains the firing times of previous general transition firings according to the order stored in $\Lambda_i.o$. A path is then defined as:

$$\pi = (\Lambda_0, \mathbf{f}_0) \xrightarrow{\epsilon_0} \ldots \xrightarrow{\epsilon_{n-1}} (\Lambda_n, \mathbf{f}_n), \tag{2}$$

where for every $j \in \{0, \ldots, n-1\}$, we require that $\epsilon_j \in M(\Lambda_j)$ leads to Λ_{j+1}.The set of all paths with arbitrary length $n \in \mathbb{N}$ starting from the initial location Λ_0 and the empty firing vector $\mathbf{f}_0$ is denoted as $Paths_{PLT}(\Lambda_0)$. $\pi(i)$ refers to $(\Lambda_i, \mathbf{f}_i)$ in path π.

The correspondence between paths through the PLT and paths in the HPnG is discussed in the following theorem.

Theorem 1. *A bijective function $m : Paths_{\mathcal{H}}(\Gamma_0) \rightarrow Paths_{PLT}(\Lambda_0)$ exists for $\Gamma_0 = \Lambda_0.\Gamma$. The following function then maps a path $\sigma \in Paths_{\mathcal{H}}(\Gamma_0)$ in the HPnG to the corresponding path $\pi \in Paths_{PLT}(\Lambda_0)$ in the PLT:*

$$m(\Gamma_0 \xrightarrow{(\Delta\tau_0, \epsilon_0)} \cdots \xrightarrow{(\Delta\tau_{n-1}, \epsilon_{n-1})} \Gamma_n) = (\Lambda_0, \mathbf{f}_0) \xrightarrow{\epsilon_0} \ldots \xrightarrow{\epsilon_{n-1}} (\Lambda_n, \mathbf{f}_n). \tag{3}$$

Proof. For each $\sigma \in Paths_{\mathcal{H}}(\Gamma_0)$, a unique corresponding path $\pi \in Paths_{PLT}(\Lambda_0)$ is iteratively constructed starting from $\Gamma_0 = \Lambda_0.\Gamma$, which induces an initial location Λ_0 following Definition 5. Note that initially $\mathbf{f}_0$ is an empty vector.

For $i \in \{0, \ldots, n-1\}$ and $\Gamma_i = \Lambda_i.\Gamma(\mathbf{f}_i)$, each event $\Upsilon(\Gamma_i, \Gamma_{i+1}) = (\Delta\tau_i, \epsilon_i)$ between states Γ_i and Γ_{i+1} corresponds to an edge from location Λ_i that is caused by model element $\epsilon_i \in M(\Lambda_i)$ and leads to the child location $\Lambda_{i+1} \in \mathbf{\Lambda}_c(\Lambda)$. If $\epsilon_i \in \mathcal{T}^G$, the firing time of the general transition ϵ_i equals $f_{\epsilon_i} = \Gamma_i.\mathbf{g}_{\epsilon_i} + \Delta\tau_i$ which must be included in $\mathbf{f}_{i+1} = [\mathbf{f}_i, f_{\epsilon_i}]$. If the event is not caused by a general transition firing, $\mathbf{f}_{i+1} = \mathbf{f}_i$ holds. Hence, m is a function.

For every path $\pi = (\Lambda_0, \mathbf{f}_0) \xrightarrow{\epsilon_0} \ldots \xrightarrow{\epsilon_{n-1}} (\Lambda_n, \mathbf{f}_n) \in Paths_{PLT}(\Lambda_0)$, there is a path $\sigma \in Paths_{\mathcal{H}}(\Gamma_0)$ which maps to $\pi = m(\sigma)$. σ contains the states of the HPnG $\Gamma_i = \Lambda_i(\mathbf{f}_i)$ for $i \in \{0, \ldots, n\}$. The edges of the path consist of $\Delta\tau_j = \Lambda_{j+1}.t(\mathbf{f_{j+1}}) - \Lambda_j.t(\mathbf{f_j})$ for $j \in \{0, \ldots, n-1\}$ and the copied model element ϵ_j. Hence, m is surjective.

Given two arbitrary but fixed paths $\sigma_1, \sigma_2 \in Paths_{\mathcal{H}}(\Gamma_0)$ with $\sigma_1 \neq \sigma_2$ and respective length $n_1, n_2 \in \mathbb{N}$, there exists an index $i \in \mathbb{N}$ with $i \leq \min\{n_1, n_2\}$ for which the states of the paths differ, i.e., $\Gamma_i^{\sigma_1} \neq \Gamma_i^{\sigma_2}$. Then, for $\pi_1 = m(\sigma_1)$ and $\pi_2 = m(\sigma_2)$, the tuples $(\Lambda_i^{\pi_1}, \mathbf{f}_i^{\pi_1}) \neq (\Lambda_i^{\pi_2}, \mathbf{f}_i^{\pi_2})$ at index i also differ in the PLT path due to the linearity of $\Lambda_i.t$ and $\Lambda_i.\Gamma$. Hence, m is injective. □

Theorem 2. *Given a path through the PLT π which ends in state $(\Lambda_i, \mathbf{f}_i)$ with $n \in \mathbb{N}$ conflict sets $C^{\mathcal{T}}_{t_1}(\Lambda_i), \ldots, C^{\mathcal{T}}_{t_n}(\Lambda_i)$. The parametric location Λ of the following state in the path $(\Lambda, \mathbf{f}_i)$ is, regardless of the conflict resolution, always from exactly one conflict set $C^{\mathcal{T}}_{t_i}(\Lambda_i),\ i \in \{1, \ldots, n\}$.*

Proof. Each conflict set has a different entry time function and the probability that they evaluate to the same value, given a fixed $\mathbf{f}_i$, equals 0, as general transitions fire according to absolute continuous probability distribution. □

The above theorem ensures, that at any time of a simulation run, as will be explained in Sect. 4, at most one conflict set needs to be resolved.

4 Simulation

Statistical Model Checking executes several simulation runs and checks the validity of a property until a predefined accuracy for the probability estimate of the property is reached. During one run, starting from a given initial state, the successor state is computed iteratively by identifying the corresponding next event. All possible future events are identified by computing their remaining execution times and the next event has the minimum time. Simulation needs to keep track of the global time and the values of all discrete and continuous variables, since they are required for the computation of the next state and event. This process repeats until the predefined time-bound is reached. While discrete-event simulation usually jumps from one explicit state in the model to the next, our analytically-guided simulation works with symbolic states and simulates through the PLT (cf. Definition 8). This considerably simplifies the computation of the next event and its successor state as presented in the following.

4.1 Simulating Through the PLT

In the following we simulate through the PLT, starting in its root location. The simulation run is then determined by two things: (i) the firing times of the general transitions, and (ii) the conflict resolution in case of a nondeterministic conflict. The firing times of the general transitions are sampled from their corresponding probability density functions, as soon as the respective general

Algorithm 1 getNextState(simulation state L)

```
1: minEventTime = t_max
2: for Λ_i : Λ_c(L.Λ) do
3:    if (Δτ, ϵ) = ϒ(L.Λ.Γ, Λ_i.Γ) ∈ ℝ_0^+ × 𝒯^G then   # general firing leads to Λ_i
4:       if L.s[ϵ].empty() then                          # check if sample in sim. state
5:          sample = sampleValue(ϵ)                      # sample required
6:          L.s[ϵ] = sample                              # save sample in sim. state
7:       else
8:          sample = L.s[ϵ]                              # sampled in previous location
9:       end if
10:      entryTime = Λ_i.t([L.f, sample])
11:   else                                                # no general firing
12:      entryTime = Λ_i.t(L.f)
13:   end if
14:   if entryTime ¡ minEventTime then
15:      minEventTime = entryTime                        # Λ_i new prospect for next event
16:      chosenChild = Λ_i
17:      conflictSet.clear()                             # clear prior found conflict set
18:   else if entryTime == minEventTime then
19:      conflictSet.add(chosenChild,Λ_i)                # Λ_i and chosenChild in conflict
20:   end if
21: end for
22: if !conflictSet.empty() then                         # nondeterministic conflict
23:    chosenChild = resolveConflict(conflictSet)        # one child is chosen
24: end if
25: if (Δτ, ϵ) = ϒ(L.Λ.Γ, L.Λ_i.Γ) ∈ ℝ_0^+ × 𝒯^G then  # general firing event
26:    L.f = [L.f, L.s[ϵ]]                               # add sample to firings
27:    removeSample(L.s[ϵ])                             # remove sample from unexpired samples
28: end if
29: L.Λ = chosenChild                                    # move to child location
30: return L
```

transitions becomes enabled. Thus, a different sample could have resulted in a different firing order and hence, a different path. For the simulation through the PLT we define a *simulation state*, which tracks the current parametric location Λ and the samples of the random variables. During a simulation run, samples for the general transitions are generated which are sorted in two variables. Vector $\mathbf{f}$ contains the (relative) expired sampling times in order $\Lambda.\mathbf{o}$. These can be used for the computation of the current simulation time and the valuations of continuous variables. Map $\mathbf{s}$ contains the samples which did not expire yet, but is not a list of future events. Instead these are given by the child locations.

Definition 9. (Simulation State). The simulation state consists of the tuple $L = (\Lambda, \mathbf{f}, \mathbf{s})$, with the current parametric location Λ, the vector $\mathbf{f} \in \Lambda.S$ with all expired samples in their firing order $\Lambda.o$. All yet unexpired samples s_i of general transitions $T_i \in \mathcal{T}^G$ are in a map $\mathbf{s}$ with $\mathbf{s}[T_i] = s_i$ for $i \in \{0, \ldots, |\mathcal{T}^G| - 1\}$.

Algorithm 2 simulateOneRun(simulation state L, property Ψ)

```
1: propertyState = undecided
2: while (L.entryTime ≤ t_max) and (propertyState == undecided) do
3:    L' = getNextState(L)
4:    propertyState = checkProperty(L, L', Ψ)
5:    L = L'
6: end while
7: return propertyState
```

The initial simulation state is defined by the root location, an empty vector $\mathbf{f}$ and without unexpired samples in $\mathbf{s}$. Algorithm 1 sketches how the next event and the next simulation state are determined given a simulation state $L = (\Lambda, \mathbf{f}, \mathbf{s})$. For each child location $\Lambda_i \in \mathbf{\Lambda}_c(\Lambda)$, the entry time is computed by using the linear function $\Lambda.t$ of the previous fired general transitions $L.\mathbf{f}$. In case the event between Λ and child location Λ_i is a general transition firing (line 3), the entry time also depends on the sample of the firing general transition. A sample for a general transition $T_i \in \mathcal{T}^G$ is generated according to its distribution $\phi_g^{\mathcal{T}}(T_i)$ when it is enabled for the first time since the simulation start or the last firing. That is the case, if no sample in $\mathbf{s}$ exists (line 4). The entry times of other events, i.e., no general transition firing, only depend on already fired transitions and thus require the past firings (line 12). The child with the smallest entry time is chosen (lines 14–16) and if entry times are equal, a nondeterministic conflict occurs. In case of a nondeterministic conflict, a conflict set is built while iterating through all child locations (line 19) which is then resolved (line 23). Different types of conflict resolutions are discussed in Sect. 4.3. In case of a general transition firing, the firing vector and the unused samples must be updated (lines 25–28) additionally to the parametric location in the simulation state L (line 29). The computation of the entry time requires $|\mathbf{f}|$ multiplications and $|\mathbf{f}|+1$ additions (in case of a general transition firing event $+1$ each) per child location. All possible events are already pre-computed and their time of occurrence is given as the entry time function $\Lambda.t$ of respective Λ. More computations are not required, as the state of the HPnG is already contained in the parametric locations. Hence, the model variables and its derivatives are only required for model checking during the simulation which facilitates the simulation.

4.2 Statistical Model Checking

Stochastic Time Logic (STL) is used to express properties regarding the state of the model at a given time point t. We follow the definition from [39] for HPnGs.

Definition 10. (STL formula [39])**.** A STL formula $\varphi ::= P_{\bowtie\theta}(\Psi)$ with bound $\theta \in [0, 1]$, comparison operator $\bowtie \in \{<, >, \leq, \geq\}$ and property

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

denotes whether the probability that property Ψ fulfills $\bowtie \theta$. The property consists of true (tt) and atomic properties (AP), which can be negated and combined with and or *until*. A discrete atomic property $d_i \sim a$ compares a discrete marking d_i with a constant $a \in \mathbb{N}$ via $\sim \in \{=, <, >, \leq, \geq\}$. A continuous atomic property $x_i \sim b$ compares a continuous marking x_i with a constant $b \in \mathbb{R}$.

For the definition of the probability space for HPnGs and the satisfaction relation for STL, we refer to [39]. We are specifically interested in property $\Psi = \neg\Phi_1 \mathcal{U}^{[0,t_{max}]} \Phi_2$, where Φ_1 and Φ_2 are potentially nested STL properties, however without the until operator. The property Ψ then translates to reaching a state that fulfills Φ_2 within time $[0, t_{max}]$ along a path of states which avoid Φ_1. For a detailed description of the STL semantic, we refer to [39].

Every simulation starts at the predefined initial state, and traverses through the PLT as illustrated in Algorithm 2. According to line 2, each simulation run is executed until the validity of property Ψ can be decided or the time-bound of the property is reached. During every execution of the while loop, the next simulation state $L' = (\Lambda', \mathbf{f}', \mathbf{s}')$ is obtained in line 3 given simulation state $L = (\Lambda, \mathbf{f}, \mathbf{s})$. Note that the state stored in L' corresponds to the state at the event occurrence time, i.e., the entry time of the parametric location $\Lambda'.t(\mathbf{f}')$, as returned by Algorithm 1. The time spent between simulation states L and L' is denoted residence time and defined as $\Delta\tau = \Lambda.t(\mathbf{f}) - \min\{\Lambda'.t(\mathbf{f}'), t_{max}\}$, i.e., the difference between the entry time of the current simulation state $\Lambda.t(\mathbf{f})$ and the minimum of the entry time $\Lambda'.t(\mathbf{f}')$ of L' and the time-bound t_{max}.

The valuations of discrete variables are constant within a parametric location, hence checking the validity of a discrete atomic property between two simulation states is straight forward. In case of a continuous atomic property, the respective continuous variable x must further be tracked between two consecutive simulations states. The value between the states L and L' evolves according to $x_0(t) = \Lambda.\Gamma.x_0(\mathbf{f}) + \Lambda.\Gamma.d_{x_0} \cdot t$ for $t \in \{0, \ldots, \Delta\tau\}$. Hence, checking the validity of a continuous atomic property (line 4) requires information on the current state and the location residence time. The validity of nested STL formulas without *Until* can then be checked by evaluating the corresponding boolean expressions. The bijection shown in Theorem 1, ensures that the validity of atomic properties is maintained in the analytically-guided simulation.

The probability that property Ψ holds is estimated via multiple simulation runs. The result of each run is a Bernoulli-distributed random variable. Hence, the validity of Ψ corresponds to a sample of a $b \in \{0, 1\}$. For $n \in \mathbb{N}$ simulation runs, a sequence of independent values $b_1, \ldots, b_n$ is computed and the arithmetic mean $\hat{p} = \frac{1}{n} \sum_{i=1}^{n} b_i$ is an estimate of the actual probability.

Instead of the arithmetic mean, confidence intervals with $\hat{p} \in [a, b]$ can be considered for $a, b \in [0, 1]$. Repeating the simulation several times, $100 * \delta\%$ of the computed confidence intervals cover the real probability for a desired level of confidence $\delta \in [0, 1]$. The simulation can be executed, such that for a desired interval width $b - a = w$ with $w \in (0, 1]$ and confidence level δ, the number of simulation runs is adapted. We refer to [39] for a detailed explanation.

4.3 Nondeterminism

Schedulers solve nondeterministic conflicts. Different scheduler classes are discussed in [9] and have been adapted for HPnGs and singular automata with random clocks. This paper presents a unified definition of four different scheduler classes in HPnGs. Different scheduler classes are obtained depending on the information available to the scheduler.

Definition 11. (Scheduler). A scheduler is a measurable function $\mathfrak{s} : \Omega \to Dist(\mathcal{T})$ which maps states $\omega \in \Omega$ to a discrete probability distribution over the set of all transitions in conflict $C^{\mathcal{T}}(\omega)$ with $support(\mathfrak{s}(\omega)) \subseteq C^{\mathcal{T}}(\omega)$. Let $\mathcal{L}_{\mathcal{H}}$ be the set of all simulation states of a HPnG $\mathcal{H}$. The projection on parts of the simulation state r is denoted as $\mathcal{L}_{\mathcal{H}}|_r$, e.g., $\mathcal{L}_{\mathcal{H}}|_{\Lambda} = \{\Lambda \mid (\Lambda, \mathbf{f}, \mathbf{s}) \in \mathcal{L}_{\mathcal{H}}\}$. The type of scheduler then depends on the choice of Ω, as follows:

$$\Omega = \begin{cases} \mathcal{L}_{\mathcal{H}} & \text{history-dependent prophetic,} \\ \mathcal{L}_{\mathcal{H}}|_{\Lambda} & \text{discrete-history non-prophetic,} \\ \mathcal{S} & \text{memoryless non-prophetic,} \\ \mathcal{S} \times \mathcal{L}_{\mathcal{H}}|_{\mathbf{s}} & \text{memoryless prophetic.} \end{cases}$$

If the scheduler is memoryless non-prophetic, it only requires knowledge on the state-space of the HPnG $\mathcal{S}$. History-dependent schedulers require knowledge on the set of simulation states $\mathcal{L}_{\mathcal{H}}$. For $\Omega = \mathcal{L}_{\mathcal{H}}$, this corresponds to history-dependent prophetic schedulers. If the information available to the scheduler is restricted to the parametric location, i.e., the domain is $\Omega = \mathcal{L}_{\mathcal{H}}|_{\Lambda}$, the scheduler is discrete-history non-prophetic. If the unused samples $\mathbf{s}$ remain part of Ω, the scheduler also has information on the future and is hence prophetic. The learned schedulers which maximize or minimize probabilities in [32] are memoryless prophetic and non-prophetic with knowledge on the current state of the model $\Gamma \in \mathcal{S}$ and $\mathcal{L}_{\mathcal{H}}|_{\mathbf{s}}$ if prophetic. Q-Learning is used for optimization and requires a finite state-space, which is obtained by discretization with proven convergence to optimal probabilities [32]. While this approach could be extended to history-dependent schedulers in a classical simulation approach, this would lead to a much larger discretized state-space for Q-learning, and hence potentially to longer training times. Instead, we include Q-learning in the analytically-guided simulation as presented in Sect. 4.1. This allows to flexibly learn schedulers from the classes presented in Definition 11. For history-dependent schedulers, the discretized state-space remains small, as the parametric location identifier and the firings already include the whole history. The parametric location identifier suffices for discrete-history schedulers.

Memoryless schedulers require more computations as the valuations of the continuous variables are not evaluated during the simulation. The general transitions firings **f** must not be given to the scheduler, as they include information on the history. Hence, at every nondeterministic conflict the valuations of the continuous variables have to be computed while the markings of the discrete variables can be taken from the parametric location. The necessity of computing the valuations of the continuous variables occurs at nondeterministic conflicts and then reduces the advantage of the analytically-guided simulation.

5 Case Study

The analytically-guided simulation is implemented in hpnmg[1] [26] building on the PLT construction. Samples are generated with a C++ Mersenne Twister pseudo-random generator [31]. Our experiments have been executed single-threaded on a machine with an AMD Ryzen 7 PRO 5850U CPU and 32 GB of RAM (Fig. 1).

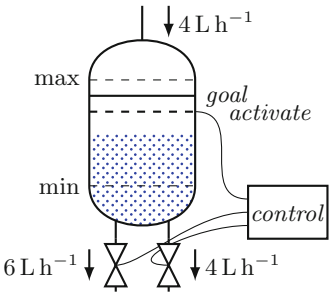

Fig. 1. A water tank with two controllable valves [44].

We reuse the water tank Fig. 1 in [40] to be able to compare our results with existing analytical and statistical results [32]. Initially, the tank of 20 m height is filled 4 m. When the tank reaches 16 m, a draining valve can be chosen nondeterministically. One valve reduces the water level by $6\,\mathrm{m\,h^{-1}}$ for 2 h, the other one by $4\,\mathrm{m\,h^{-1}}$ for 1 h. Every valve is blocked after usage for a uniformly distributed time between 0 and 6 hours. A visual representation of the underlying HPnG model is shown in [32, Fig. 3] and the corresponding hybrid automaton model is shown in [40, Fig. 5] (see Appendix A for convenience). We maximize the probability that the tank reaches a height h of 18 m before t_{max}: $\Psi = tt\ U^{[0,t_{max}]} h \geq 18$ for different values of $t_{max} \in \{7, \ldots, 11\}$.

[1] https://zivgitlab.uni-muenster.de/ag-sks/tools/hpnmg/-/tree/simulation.

Table 1. Comparison of results for different scheduling classes for the tank case study.

		$t_{\max}$	7	8	9	10	11
Discrete-history non-prophetic	HPnG [44]	p_{max}	0	0.5347	0.6264	0.6444	0.7375
		Runtime	0.01 s	32.74 s	55.64 s	62.23 s	141.45 s
	Analytically-guided simulation	Midpoint	0	0.5278	0.6219	0.6447	0.7372
		PLT constr.	0.001 s	0.002 s	0.005 s	0.012 s	0.031 s
		Train. time	0.001 s	0.005 s	0.004 s	0.004 s	0.005 s
		Sim. time	0.004 s	0.076 s	0.074 s	0.072 s	0.065 s
Memoryless non-prophetic	modes ML [32]	Midpoint	0	0.53	0.62	0.64	0.74
		Train. time	1.00 s	1.1 s	1.10 s	1.30 s	1.40 s
		Sim. time	0.20 s	2.70 s	3.10 s	3.30 s	3.00 s
	Analytically-guided simulation	midpoint	0	0.5386	0.6278	0.6397	0.7386
		PLT constr.	0.001 s	0.002 s	0.005 s	0.012 s	0.031 s
		Train. time	0.001 s	0.005 s	0.007 s	0.006 s	0.007 s
		Sim. time	0.005 s	0.086 s	0.084 s	0.083 s	0.070 s
Memoryless prophetic	Modes ML [32]	Midpoint	0	0.60	0.64	0.65	0.74
		Train. time	3.00 s	3.40 s	4.10 s	3.90 s	4.00 s
		Sim. time	0.20 s	2.60 s	2.90 s	2.80 s	2.60 s
	Analytically-guided simulation	Midpoint	0	0.6060	0.6471	0.6493	0.7439
		PLT constr.	0.001 s	0.002 s	0.005 s	0.012 s	0.031 s
		Train. time	0.002 s	0.035 s	0.037 s	0.041 s	0.045 s
		Sim. time	0.004 s	0.080 s	0.077 s	0.078 s	0.068 s
History-dep. prophetic	Flowpipe [44]	p_{max}	0	0.6041	0.6488	0.6514	0.7798
		Runtime	0.25 s	2.17 s	11.68 s	71.53 s	3277.94 s
	Analytically-guided simulation	Midpoint	0	0.6039	0.6467	0.6533	0.7710
		PLT constr.	0.001 s	0.002 s	0.005 s	0.012 s	0.031 s
		Train. time	0.121 s	1.625 s	1.592 s	1.456 s	1.846 s
		Sim. time	0.005 s	0.099 s	0.071 s	0.071 s	0.058 s

We compare previously published results from different tools, which compute optimized probabilities for different scheduler classes. We adapt our used scheduler class to match the corresponding tool. Both analytical approaches, HPnG and Flowpipe, are exact up to a statistically estimated error of at most 10^{-3} which stems from the multidimensional integration. modes ML and our approach compute confidence intervals, here with $\delta = 0.99$ and $w = 0.02$. The number of training runs and the training parameters are adapted from [32].

Table 1 presents the results of our analytically-guided simulation with different scheduler classes and results computed by the above mentioned tools. For modes ML and the analytical-guided simulation, the midpoint of the computed confidence interval is given together with the training time of a scheduler and the simulation time required to obtain a confidence interval with width w. To ensure a fair comparison, the run time of the analytically-guided simulation always indicates the time to generate the PLT. However, note that it would suffice to generate the PLT once for each t_{max}.

HPnG computes optimal probabilities for discrete-history non-prophetic schedulers. Our results match as the confidence intervals include the respective p_{max}. modes ML learns memoryless (non-)prophetic schedulers and in both cases, the results match, as the confidence intervals overlap. With the Flowpipe approach, discrete-history non-prophetic and history-dependent prophetic schedulers are considered. To keep the table clear, the former are not shown, but match the results of HPnG [44] with run times similar to the prophetic scheduler. To match the history-dependent prophetic scheduler in the Flowpipe, 1 million training runs were required. Note that in [32] 1 million training runs required 68 s, while it only requires 1.846 s in the analytically-guided simulation.

Our approach clearly outperforms both, the analytical approaches and modes ML in all cases. modes ML was executed on a different machine (i5-6600T, 16 GB memory), however the impact on performance should be negligible, as modes ML is single-threaded. While the run times of the analytical approaches increase considerably with t_{max}, simulation scales better. Training and simulation times only increase marginally with the time-bound. The time to build the PLT scales worse than the simulation and training time, but remains small in this model.

6 Conclusion

We presented analytically-guided simulation which has the ability to significantly reduce the computation effort when learning optimal schedulers based on simulating paths in the symbolic state-space representation of a PLT. By leveraging analytical knowledge to guide the simulation process, our approach outperforms existing analytical and statistical approaches for computing reachability probabilities optimized for different scheduler classes. As a consequence of Theorem 1 analytically-guided simulation explores the same state-space as discrete-event simulation over paths of a HPnG. Future work will apply the concept to more expressive model classes, e.g., rectangular automata with random clocks [7]. Further, we will integrate algorithms for rare-event probabilities.

A Visual Representation of the Tank Model

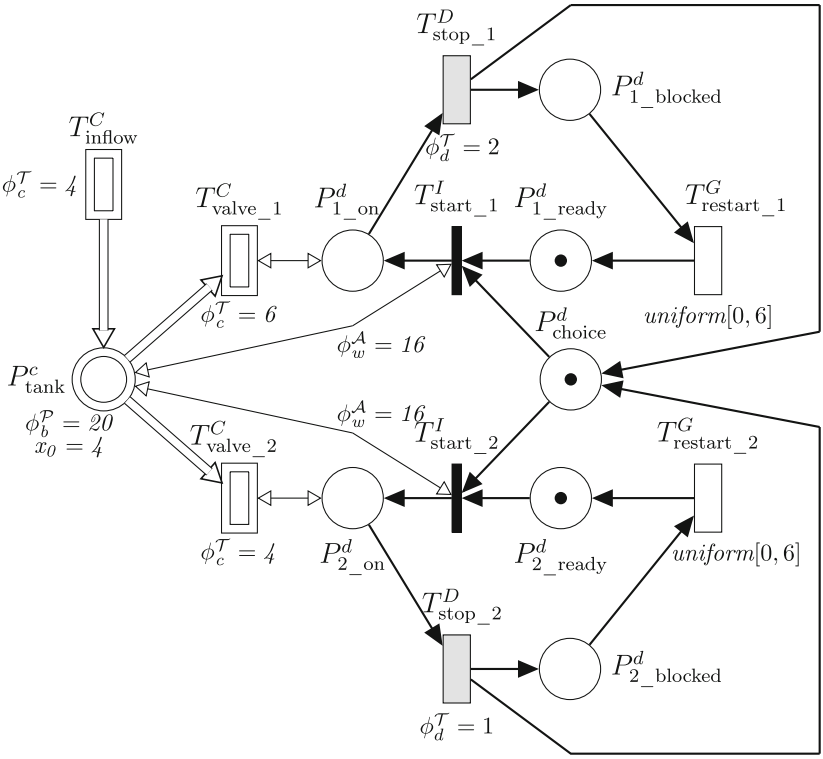

Fig. 2. Tank system modeled as HPnG as in [32]. The continuous place P^c_{tank} models the fluid level of the tank. The two valves are modeled by the continuous transitions $T^C_{\text{valve\_1}}$ and $T^C_{\text{valve\_2}}$, which are enabled while a token is in $P^d_{\text{1\_on}}$ respectively $P^d_{\text{2\_on}}$. The nondeterministic choice is modeled by the conflict of the immediate transitions $T^I_{\text{start\_1}}$ and $T^I_{\text{start\_2}}$ if a token is in P^d_{choice}.

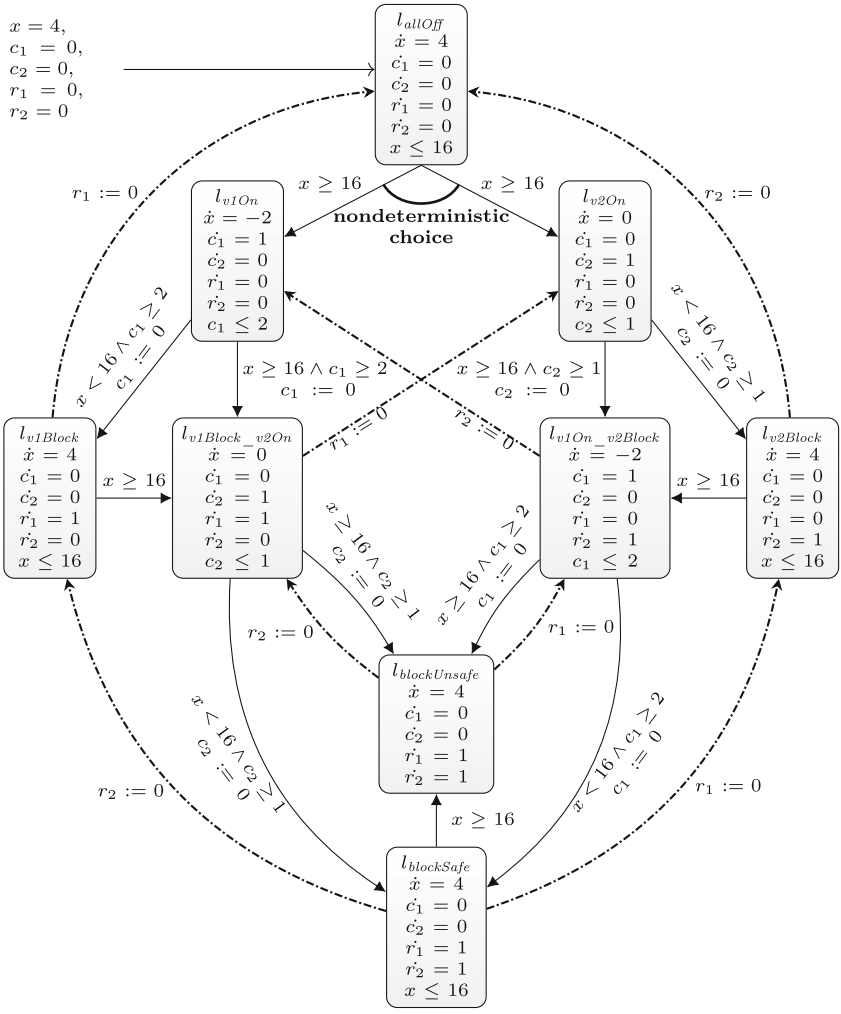

Fig. 3. Tank system modeled as singular automaton with random clocks [40]. The fluid level of the tank is modeled by the variable x. The nondeterministic choice is highlighted. The active time for the two valves is given by the clocks c_1 and c_2. Additionally the random blocking times for the valves are modeled by the random clocks r_1 and r_2.

References

1. Abate, A., Katoen, J.P., Lygeros, J., Prandini, M.: Approximate model checking of stochastic hybrid systems. Eur. J. Control. **16**(6), 624–641 (2010). https://doi.org/10.3166/ejc.16.624-641
2. Adelt, J., Herber, P., Niehage, M., Remke, A.: Towards safe and resilient hybrid systems in the presence of learning and uncertainty. In: Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles—11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22–30, 2022, Proceedings, Part I. LNCS, vol. 13701, pp. 299–319. Springer (2022). https://doi.org/10.1007/978-3-031-19849-6_18
3. Budde, C.E., D'Argenio, P.R., Hartmanns, A.: Better automated importance splitting for transient rare events. In: Dependable Software Engineering. Theories, Tools, and Applications, LNCS, vol. 10606, pp. 42–58. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-69483-2_3
4. Budde, C.E., D'Argenio, P.R., Hartmanns, A., Sedwards, S.: An efficient statistical model checker for nondeterminism and rare events. Int. J. Softw. Tools Technol. Transfer **22**(6), 759–780 (2020). https://doi.org/10.1007/s10009-020-00563-2
5. Cai, M., Peng, H., Li, Z., Kan, Z.: Learning-based probabilistic LTL motion planning with environment and motion uncertainties. IEEE Trans. Autom. Control **66**(5), 2386–2392 (2021). https://doi.org/10.1109/TAC.2020.3006967
6. David, R., Alla, H.: Discrete, Continuous, and Hybrid Petri Nets. Springer, Berlin Heidelberg, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-10669-9
7. Delicaris, J., Schupp, S., Ábrahám, E., Remke, A.: Maximizing reachability probabilities in rectangular automata with random clocks. In: 17th International Symposium on Theoretical Aspects of Software Engineering. LNCS, vol. 13931, pp. 1–19. Springer (2023). https://doi.org/10.1007/978-3-031-35257-7_10
8. D'Argenio, P., Legay, A., Sedwards, S., Traonouez, L.M.: Smart sampling for lightweight verification of Markov decision processes. Int. J. Softw. Tools Technol. Transfer **17**(4), 469–484 (2015). https://doi.org/10.1007/s10009-015-0383-0
9. D'Argenio, P.R., Gerhold, M., Hartmanns, A., Sedwards, S.: A hierarchy of scheduler classes for stochastic automata. In: Foundations of Software Science and Computation Structures, LNCS, vol. 10803, pp. 384–402. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-89366-2_21
10. D'Argenio, P.R., Hartmanns, A., Sedwards, S.: Lightweight statistical model checking in nondeterministic continuous time. In: Leveraging Applications of Formal Methods, Verification and Validation. Verification, LNCS, vol. 11245, pp. 336–353. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-030-03421-4_22
11. Ellen, C., Gerwinn, S., Fränzle, M.: Statistical model checking for stochastic hybrid systems involving nondeterminism over continuous domains. Int. J. Softw. Tools Technol. Transfer **17**(4), 485–504 (2015). https://doi.org/10.1007/s10009-014-0329-y
12. Fränzle, M., Hahn, E.M., Hermanns, H., Wolovick, N., Zhang, L.: Measurability and safety verification for stochastic hybrid systems. In: Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control—HSCC '11. p. 43. ACM Press, Chicago, IL, USA (2011). https://doi.org/10.1145/1967701.1967710
13. Fränzle, M., Teige, T., Eggers, A.: Engineering constraint solvers for automatic analysis of probabilistic hybrid automata. J. Logic Algebraic Program. **79**(7), 436–466 (2010). https://doi.org/10.1016/j.jlap.2010.07.003

14. Gao, Y., Fränzle, M.: A solving procedure for stochastic satisfiability modulo theories with continuous domain. In: 12th International Conference on Quantitative Evaluation of Systems (QEST). LNCS, vol. 9259, pp. 295–311. Springer (2015). https://doi.org/10.1007/978-3-319-22264-6_19
15. Ghasemieh, H., Remke, A., Haverkort, B.R.: Survivability evaluation of fluid critical infrastructures using hybrid petri nets. In: IEEE 19th Pacific Rim International Symposium on Dependable Computing, PRDC 2013, Vancouver, BC, Canada, December 2–4, 2013, pp. 152–161. IEEE Computer Society (2013). https://doi.org/10.1109/PRDC.2013.34
16. Ghasemieh, H., Remke, A., Haverkort, B.R.: Survivability analysis of a sewage treatment facility using hybrid petri nets. Perform. Evaluation **97**, 36–56 (2016). https://doi.org/10.1016/j.peva.2015.11.004
17. Gribaudo, M., Remke, A.: Hybrid Petri nets with general one-shot transitions. Perform. Eval. **105**, 22–50 (2016). https://doi.org/10.1016/j.peva.2016.09.002
18. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.P.: A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods Syst. Des. **43**(2), 191–232 (2013). https://doi.org/10.1007/s10703-012-0167-z
19. Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Faithful and effective reward schemes for model-free reinforcement learning of omega-regular objectives. In: Automated Technology for Verification and Analysis, LNCS, vol. 12302, pp. 108–124. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-59152-6_6
20. Hartmanns, A., Hermanns, H.: The modest toolset: an integrated environment for quantitative modelling and verification. In: Tools and Algorithms for the Construction and Analysis of Systems, LNCS, vol. 8413, pp. 593–598. Springer, Berlin Heidelberg, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_51
21. Hartmanns, A., Hermanns, H., Krčál, J.: Schedulers are no Prophets. In: Semantics, Logics, and Calculi, LNCS, vol. 9560, pp. 214–235. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-27810-0_11
22. Hasanbeig, M., Kantaros, Y., Abate, A., Kroening, D., Pappas, G.J., Lee, I.: Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In: 2019 IEEE 58th Conference on Decision and Control (CDC), pp. 5338–5343. IEEE, Nice, France (2019). https://doi.org/10.1109/CDC40024.2019.9028919
23. Hasanbeig, M., Abate, A., Kroening, D.: Cautious reinforcement learning with logical constraints. In: Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, pp. 483–491 (2020)
24. Heymann, M., Feng Lin, Meyer, G., Resmerita, S.: Analysis of Zeno behaviors in a class of hybrid systems. IEEE Trans. Autom. Control **50**(3), 376–383 (2005). https://doi.org/10.1109/TAC.2005.843874
25. Hüls, J., Remke, A.: Model checking hpngs in multiple dimensions: Representing state sets as convex polytopes. In: Formal Techniques for Distributed Objects, Components, and Systems—39th IFIP WG 6.1 International Conference, FORTE 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17–21, 2019, Proceedings. LNCS, vol. 11535, pp. 148–166. Springer (2019). https://doi.org/10.1007/978-3-030-21759-4_9
26. Hüls, J., Niehaus, H., Remke, A.: hpnmg: AC++ tool for model checking hybrid Petri nets with general transitions. In: NASA Formal Methods, LNCS, vol. 12229,

pp. 369–378. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-55754-6_22
27. Hüls, J., Pilch, C., Schinke, P., Niehaus, H., Delicaris, J., Remke, A.: State-space construction of hybrid Petri nets with multiple stochastic firings. ACM Trans. Model. Comput. Simul. **31**(3), 1–37 (2021). https://doi.org/10.1145/3449353
28. Jegourel, C., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Sedwards, S.: Importance sampling for stochastic timed automata. In: Dependable Software Engineering: Theories, Tools, and Applications, LNCS, vol. 9984, pp. 163–178. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-47677-3_11
29. Laurenti, L., Lahijanian, M., Abate, A., Cardelli, L., Kwiatkowska, M.: Formal and efficient synthesis for continuous-time linear stochastic hybrid processes. IEEE Trans. Autom. Control **66**(1), 17–32 (2021). https://doi.org/10.1109/TAC.2020.2975028. Jan
30. Legay, A., Sedwards, S., Traonouez, L.M.: Scalable verification of Markov decision processes. In: Software Engineering and Formal Methods, LNCS, vol. 8938, pp. 350–362. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-15201-1_23
31. Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. **8**(1), 3–30 (1998). https://doi.org/10.1145/272991.272995
32. Niehage, M., Hartmanns, A., Remke, A.: Learning optimal decisions for stochastic hybrid systems. In: Proceedings of the 19th ACM-IEEE International Conference on Formal Methods and Models for System Design, pp. 44–55. ACM, Virtual Event China (2021). https://doi.org/10.1145/3487212.3487339
33. Niehage, M., Pilch, C., Remke, A.: 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, pp. 88–95. ACM, Tsukuba Japan (2020). https://doi.org/10.1145/3388831.3388842
34. Niehage, M., Remke, A.: Learning that grid-convenience does not hurt resilience in the presence of uncertainty. In: Formal Modeling and Analysis of Timed Systems—20th International Conference, FORMATS 2022, Warsaw, Poland, September 13–15, 2022, Proceedings. LNCS, vol. 13465, pp. 298–306. Springer (2022). https://doi.org/10.1007/978-3-031-15839-1_17
35. Pilch, C., Edenfeld, F., Remke, A.: 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, pp. 186–191. ACM Press, Venice, Italy (2017). https://doi.org/10.1145/3150928.3150956
36. Pilch, C., Hartmanns, A., Remke, A.: 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, pp. 1–11. ACM, Sydney New South Wales Australia (2020). https://doi.org/10.1145/3365365.3382198
37. Pilch, C., Krause, M., Remke, A., Ábrahám, E.: A transformation of Hybrid Petri nets with stochastic firings into a subclass of stochastic hybrid automata. In: NASA Formal Methods, LNCS, vol. 12229, pp. 381–400. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-55754-6_23
38. Pilch, C., Niehage, M., Remke, A.: 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), pp. 157–169. IEEE, Milwaukee, WI (2018). https://doi.org/10.1109/MASCOTS.2018.00024

39. Pilch, C., Remke, A.: 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), pp. 475–486. IEEE, Denver, CO, USA (2017). https://doi.org/10.1109/DSN.2017.41
40. Pilch, C., Schupp, S., Remke, A.: Optimizing reachability probabilities for a restricted class of stochastic hybrid automata via Flowpipe-construction. In: Quantitative Evaluation of Systems, LNCS, vol. 12846, pp. 435–456. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-85172-9_23
41. Sadigh, D., Kim, E.S., Coogan, S., Sastry, S.S., Seshia, S.A.: A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In: 53rd IEEE Conference on Decision and Control, pp. 1091–1096. IEEE, Los Angeles, CA, USA (2014). https://doi.org/10.1109/CDC.2014.7039527
42. Shmarov, F., Zuliani, P.: ProbReach: verified probabilistic delta-reachability for stochastic hybrid systems. In: Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, pp. 134–139. ACM, Seattle Washington (Apr 2015). https://doi.org/10.1145/2728606.2728625
43. Shmarov, F., Zuliani, P.: Probabilistic hybrid systems verification via SMT and Monte Carlo techniques. In: Hardware and Software: Verification and Testing, LNCS, vol. 10028, pp. 152–168. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-49052-6_10
44. da Silva, C., Schupp, S., Remke, A.: Optimizing reachability probabilities for a restricted class of stochastic hybrid automata via flowpipe-construction. ACM Trans. Model. Comput. Simul. (2023). https://doi.org/10.1145/3607197
45. Zimmermann, A., Maciel, P.: Importance function derivation for RESTART simulations of Petri nets. In: 9th International Workshop on Rare Event Simulation (2012)