Classic and Non-Prophetic Model Checking for Hybrid Petri Nets with Stochastic Firings

Carina Pilch
Westfälische Wilhelms-Universität
Münster, Germany
carina.pilch@uni-muenster.de

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

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

ABSTRACT

Nondeterminism occurs naturally in Petri nets whenever multiple events are enabled at the same time. Traditionally, it is resolved at specification time using probability weights and priorities. In this paper, we focus on model checking for hybrid Petri nets with an arbitrary but finite number of stochastic firings (HPnGs) while preserving the inherent nondeterminism as a first-class modelling and analysis feature. We present two algorithms to compute optimal *non-prophetic* and *prophetic* schedulers. The former can be applied to all HPnG models while the latter is only applicable if information on the firing times of general transitions is specifically encoded in the model. Both algorithms make use of recent work on the parametric location tree, which symbolically unfolds the state space of an HPnG. A running example illustrates the approach and confirms the feasibility of the presented algorithm.

CCS CONCEPTS

• **Theory of computation → Timed and hybrid models**.

KEYWORDS

Reachability analysis, Stochastic hybrid systems, Hybrid Petri nets with general transitions, Nondeterminism, Prophetic Schedulers

ACM Reference Format:
Carina Pilch, Arnd Hartmanns, and Anne Remke. 2020. Classic and Non-Prophetic Model Checking for Hybrid Petri Nets with Stochastic Firings. In *23rd ACM International Conference on Hybrid Systems: Computation and Control (HSCC '20), April 22–24, 2020, Sydney, NSW, Australia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3365365.3382198

1 INTRODUCTION

Petri nets offer a graphical notation for process models with a precise mathematical semantics. In the original definition [26], the order in which multiple concurrently enabled transitions fire is nondeterministic, i.e. any possible order must be considered in the analysis. Petri nets are thus well-suited for modelling the concurrent behaviour of distributed systems. Over time, variants have been introduced that cover quantitative aspects. In this paper, we focus on systems that exhibit *stochastic* behaviour, and that intermix discrete control with complex continuous dynamics (i.e. *hybrid* systems). Existing Petri net formalisms in this area, like generalised stochastic Petri nets (GSPN [1]), fluid stochastic Petri nets (FSPN [19]), and hybrid Petri nets [2], resolve their inherent nondeterminism either probabilistically by assigning weights to transitions, via a fixed execution order prescribed by transition priorities, or using a combination of both. However, any prescribed deterministic or probabilistic ordering between two transition firings is inappropriate to model concurrency (i.e. systems composed of independent computational or physical components), unknown (controllable or adversarial) environments and inputs, and (intentional) underspecification.

With this paper, we make nondeterminism a first-class feature in the modelling and analysis of concurrent stochastic hybrid systems with Petri nets: we allow hybrid Petri nets with general transitions (HPnGs [15]) to make nondeterministic choices between transition firings that are possible at the same point in time. In a nondeterministic system, reachability probabilities depend on how such choices are resolved, i.e. on *schedulers*. Probabilistic model checkers compute the maximum or minimum probability over *all possible* schedulers, the results being referred to as *optimal* probabilities. Specific scheduler classes of interest are, for example, partial-information, history-dependent, or randomised schedulers.

Stochasticity in HPnGs stems from *general transitions* which fire after a delay sampled from some specified general (usually continuous) probability distribution. HPnGs thus allow for non-memoryless distributions, and a crucial question is what information should be available to "all possible" schedulers in the first place. In particular, can they observe the time at which a stochastically-timed event *will happen* in the future? Complete-information schedulers for stochastic automata (SA [6], akin to HPnG without continuous places) can do so [4]. We call such schedulers *prophetic* [17]. They may be considered unrealistic, but constitute the most general class of schedulers for model checking. *Non-prophetic* schedulers, however, are more desirable in applications because making decisions based on the timing of future random events is, in most cases, impossible in reality. However, from a theoretical perspective, they are less general than prophetic schedulers.

In this paper, we present two algorithms to compute optimal reachability probabilities in HPnGs, one for prophetic and one for non-prophetic schedulers. Both extend recent results on the symbolic representation of the state space [20] of, and on the corresponding model checking algorithms [21] for, purely probabilistic HPnGs. The key idea of the existing work is to separate the analysis into computing a *parametric location tree* (PLT) to identify reachable goal states in the hybrid dynamics followed by an integration over the general probability distributions to compute reachability probabilities. The existing algorithms support an arbitrary but finite number of general transition firings. They thus work well to compute time-bounded reachability probabilities in HPnGs.

Our new algorithm for the non-prophetic case compares all nondeterministic choices in the PLT and picks the optimum in terms of the reachability probability. It thus finds optimal history-dependent deterministic schedulers. Our algorithm for prophetic schedulers, however, requires information on stochastic events to be explicitly encoded in the model: the delay until a general transition fires must be made explicit by adding a continuous place that stores an amount of fluid corresponding to the random variable modelling that delay. This is referred to as *pre-computation*. The prophetic algorithm then uses the distributions of the random variables to find the optimal decision boundaries over time, which might depend on the values and the firing order of the pre-computed random variables. This poses an optimisation problem over the support of the random variables that, to be solved in full generality, would require the availability of efficient optimisation solvers.

Overall, our algorithms complement a graphical formal modelling framework for a challenging yet highly practical [12, 13, 23] combination of quantitative aspects by an exhaustive analysis that delivers results with guaranteed coverage and accuracy. We have implemented the non-prophetic algorithm in full generality, and provide a proof of concept of the prophetic variant that supports at most one decision point per pair of concurrently enabled transitions. We use an electric vehicle decision-making case study as a running example to confirm the effectiveness of the non-prophetic algorithm and to illustrate the prophetic approach.

Related work. Like we add nondeterminism to (the analysis of) HPnGs, it has been reintroduced to GSPN via a transformation [7] to the nondeterministic Markovian formalism of Markov automata (MA) [8]. Due to the absence of hybrid dynamics plus the memoryless nature of GSPN, standard probabilistic model checking techniques for classic schedulers apply. No class of schedulers with limited information is equivalent to prophetic schedulers for SA [5]; this is in contrast to MA where e.g. untimed schedulers suffice for unbounded expected-time properties. Yet, even in settings where model checking for classic schedulers is tractable, scheduler classes with partial information are of interest as evidenced by work on distributed schedulers [14] for Markov decision processes (MDP) and the model of partially-observable MDP [3]. The verification of *unbounded* properties becomes undecidable in such settings [14, 25]; we in contrast rely on limited-information schedulers to make checking *time-bounded* properties *practical*. For HPnGs, the analysis of non-prophetic schedulers is conceptually and practically easier than that for prophetic schedulers, due to the characteristics of the PLT-based approach. Related Petri net formalisms like hybrid Petri nets [2], FSPN [19], and colored Petri nets [22], as well as their extension [10], do not allow nondeterminism as a model feature.

2 HYBRID PETRI NETS

HPnGs extend the formalism of hybrid Petri nets [2] with stochastic behaviour. We give a simplified definition of their syntax and semantics and illustrate them through our running example.

Definition 2.1. An HPnG is a 5-tuple $(\mathcal{P}, \mathcal{T}, \mathcal{A}, \mathbf{M}_0, \Phi)$ where Φ is a tuple $(\Phi_d^{\mathcal{T}}, \Phi_g^{\mathcal{T}}, \Phi_c^{\mathcal{T}})$ of parameter functions. The finite set $\mathcal{P} = \mathcal{P}^d \uplus \mathcal{P}^c$ contains the *discrete places* (drawn as circles) and the *continuous places* (double-outline circles). Any discrete place $P_i \in \mathcal{P}^d$ holds $m_i \in \mathbb{N}_0$ tokens. A continuous place $P_j \in \mathcal{P}^c$ holds an amount of fluid $x_j \in \mathbb{R}_0^+$. The initial number of tokens and the initial fluid levels are given by the *initial marking* $\mathbf{M}_0 = (\mathbf{m}_0, \mathbf{x}_0)$ with $\mathbf{m}_0 = \{m_1, ..., m_{|\mathcal{P}^d|}\}$ and $\mathbf{x}_0 = \{x_1, ..., x_{|\mathcal{P}^c|}\}$.

A *transition* connects zero or more input and output places of the same type. The finite set $\mathcal{T} = \mathcal{T}^I \uplus \mathcal{T}^D \uplus \mathcal{T}^G \uplus \mathcal{T}^C$ of transitions holds *immediate* (drawn as solid black bars), *deterministic* (grey filled), *general* (single-outline) and *continuous* (double-outline) transitions. Immediate transitions in $\mathcal{T}^I$ *fire* as soon as they are *enabled*; deterministic transitions in $\mathcal{T}^D$ fire after being enabled for a predefined time given by $\Phi_d^{\mathcal{T}} : \mathcal{T}^D \to \mathbb{R}^+$. The time to fire for a general transition follows a continuous probability distribution given by $\Phi_g^{\mathcal{T}} : \mathcal{T}^G \to (f : \mathbb{R}^+ \to [0, 1])$. A continuous transition in $\mathcal{T}^C$ has a continuous inflow and outflow of fluid and fires continuously if enabled. $\Phi_c^{\mathcal{T}} : \mathcal{T}^C \to \mathbb{R}^+$ defines the nominal firing rates of these transitions. The finite set $\mathcal{A} = \mathcal{A}^d \uplus \mathcal{A}^f \uplus \mathcal{A}^t \uplus \mathcal{A}^h$ consists of *discrete* (solid arrow), *continuous* (double-outline arrow), *test* (double arrow) and *inhibitor* (circle arrow) *arcs*, which connect places with transitions. Discrete places and non-continuous transitions can be connected via discrete arcs in $\mathcal{A}^d \subseteq (\mathcal{P}^d \times (\mathcal{T} \setminus \mathcal{T}^C)) \cup ((\mathcal{T} \setminus \mathcal{T}^C) \times \mathcal{P}^d)$. A continuous arc in $\mathcal{A}^f \subseteq (\mathcal{P}^c \times \mathcal{T}^C) \cup (\mathcal{T}^C \times \mathcal{P}^c)$ connects one continuous place to one continuous transition. Test arcs in $\mathcal{A}^t \subseteq \mathcal{P}^d \times \mathcal{T}$ and inhibitor arcs $\mathcal{A}^h \subseteq \mathcal{P}^d \times \mathcal{T}$ connect a discrete place to any transition.

A transition T is *enabled* in a marking iff all of its input places (i.e. places P with $(P, T) \in \mathcal{A}^d \cup \mathcal{A}^f$) as well as all discrete places connected to it by a test arc have a token or fluid available, and all discrete places connected by an inhibitor arc hold no token. When a non-continuous transition *fires*, one token is removed from every input place and one added to every output place (i.e. the places P with $(T, P) \in \mathcal{A}^d \cup \mathcal{A}^f$). At any point in time, the amount of time that a transition was enabled since its last firing is its *enabling time*. We refer to [15, 27] for the full formal rules and in particular for a detailed description of the behaviour of general transitions.

The *drift* of a place P describes its derivative and equals the sum of the firing rates of all enabled incoming continuous transitions (i.e. enabled transitions T with $(T, P) \in \mathcal{A}^f$) minus the sum of the firing rates of all enabled outgoing continuous transitions (i.e. enabled transitions T with $(P, T) \in \mathcal{A}^f$). If the fluid level of P is 0 and the supposed drift is negative, the drift is set to 0 and the rates of the incoming transitions get adapted accordingly (as defined by the *rate adaption* rules [15]). As we consider constant nominal fluid rates, we obtain piecewise-linear continuous behavior.

Definition 2.2. A *state* of an HPnG is a tuple $\Gamma = (\mathbf{m}, \mathbf{x}, \mathbf{c}, \mathbf{d}, \mathbf{g}, \mathbf{e})$ with discrete marking $\mathbf{m}$, continuous marking $\mathbf{x}$, and *enabling time* $\mathbf{c}$ for each deterministic transition. For each continuous place, *drift* $\mathbf{d}$ records the current change of fluid level per time unit. $\mathbf{g}$ holds the enabling times in $\mathbb{R}_0^+$ for general transitions, and $\mathbf{e}$ describes the Boolean *enabling status* of all transitions. The initial state is $\Gamma_0 \in S$ with S the set of all possible states and $\Gamma_0 = (\mathbf{m}_0, \mathbf{x}_0, \mathbf{0}, \mathbf{d}_0, \mathbf{0}, \mathbf{e}_0)$. The values of the $\mathbf{d}$ and $\mathbf{e}$ are uniquely determined by $\mathbf{m}$ and $\mathbf{x}$.

The discrete part of the state (the numbers of tokens $\mathbf{m}$, the drifts $\mathbf{d}$, and the enabling statuses $\mathbf{e}$) only changes when an *event* occurs:

Definition 2.3. An event $\Upsilon(\Gamma_i, \Gamma_{i+1}) = (\Delta\tau_i, \varepsilon_i)$ in an HPnG with $\varepsilon_i \in \mathcal{P}^c \cup \mathcal{T} \setminus \mathcal{T}^C \cup \mathcal{A}^t \cup \mathcal{A}^h$ is the change from state Γ_i to state Γ_{i+1}, where $\Delta\tau_i \in \mathbb{R}_0^+$ is the time after which the event takes place, starting from Γ_i, such that
(1) potentially $\Gamma_i.\mathbf{m} \neq \Gamma_{i+1}.\mathbf{m} \wedge \varepsilon_i \in \mathcal{T} \setminus \mathcal{T}^C$ (a transition fires), or
(2) $\Gamma_i.\mathbf{d} \neq \Gamma_{i+1}.\mathbf{d} \wedge \varepsilon_i \in \mathcal{P}^c$ (a boundary is reached), or
(3) potentially $\Gamma_i.\mathbf{e} \neq \Gamma_{i+1}.\mathbf{e} \wedge \varepsilon_i \in \mathcal{A}^t \cup \mathcal{A}^h$ (arc condition changes).
We denote by $\mathcal{E}(\Gamma_i) = \{\Upsilon(\Gamma_i, \Gamma_{i+1}) = (\Delta\tau_i, \varepsilon_i)\}$ the set of events which can occur in state Γ_i and by $\mathcal{E}^{min}(\Gamma_i) = \{\Upsilon_j(\Gamma_i, \Gamma_j) = (\Delta\tau_j, \varepsilon_j) \in \mathcal{E}(\Gamma_i) \mid \nexists \Upsilon_k(\Gamma_i, \Gamma_k) = (\Delta\tau_k, \varepsilon_k) \in \mathcal{E}(\Gamma_i) \colon \Delta\tau_k < \Delta\tau_j\}$ the subset of events with the minimum remaining time. Having $\Upsilon(\Gamma_i, \Gamma_{i+1}) \in \mathcal{E}^{min}(\Gamma_i)$, the continuous evolution between Γ_i and Γ_{i+1} results in the set of states $S(\Gamma_i, \Gamma_{i+1}) = \{\Gamma_j \in S \mid \Gamma_j.\mathbf{m} = \Gamma_i.\mathbf{m} \ \wedge\ \Gamma_j.\mathbf{d} = \Gamma_i.\mathbf{d} \ \wedge\ \Gamma_j.\mathbf{e} = \Gamma_i.\mathbf{e} \ \wedge\ \exists \Delta\tau_j \colon 0 \le \Delta\tau_j < \Delta\tau_i \ \wedge\ \Gamma_j.\mathbf{x} = \Gamma_i.\mathbf{x} + \Gamma_i.\mathbf{d} \cdot \Delta\tau_j \ \wedge\ \Gamma_j.\mathbf{c} = \Gamma_i.\mathbf{c} + \Delta\tau_j \ \wedge\ \Gamma_j.\mathbf{g} = \Gamma_i.\mathbf{g} + \Delta\tau_j\}$, as time evolves.

Definition 2.4. A finite *path* $\sigma = \Gamma_0 \xrightarrow{(\Delta\tau_0, \varepsilon_0)} \Gamma_1 \xrightarrow{(\Delta\tau_1, \varepsilon_1)} \dots \Gamma_n$ is a finite alternating sequence of states and events s.t. there exists an event $\Upsilon(\Gamma_i, \Gamma_{i+1}) = (\Delta\tau_i, \varepsilon_i) \in \mathcal{E}^{min}(\Gamma_i)$ for every $i \in \mathbb{N}, i < n$, and $\Delta\tau_i$ is the time spent in states from $S(\Gamma_i, \Gamma_{i+1})$ before the event occurs. The state at time t in σ is denoted $\gamma(\sigma, t) \in S(\Gamma_i, \Gamma_{i+1})$ with $i = \min\{j \in \mathbb{N} \colon t \le \sum_{0 \le l \le j} \Delta\tau_l\}$ and $\Delta\tau(\gamma(\sigma, t)) = t - \sum_{0 \le l \le i-1} \Delta\tau_l$. We denote the set of all finite paths which start from the initial state Γ_0 by $Paths(\Gamma_0)$ and further we define the subset of finite paths up to time t (starting from Γ_0) as $Paths(\Gamma_0, t) = \{\sigma = \Gamma_0 \xrightarrow{(\Delta\tau_0, \varepsilon_0)} \Gamma_1 \xrightarrow{(\Delta\tau_1, \varepsilon_1)} \dots \Gamma_k \xrightarrow{(\Delta\tau_k, \varepsilon_k)} \mid \sum_{i=0}^{k-1} \Delta\tau_i \le t < \sum_{i=0}^{k} \Delta\tau_i\}$, such that $\Gamma_i \in S$ is the state after the i-th event $\Upsilon(\Gamma_{i-1}, \Gamma_i)$.

Running example. Figure 1 shows an HPnG for the charging and discharging of an electric car. One can choose between taking this car or another petrol-powered car, while the (future) state of charge of the electric car (i.e. the distance it can drive) may be unknown. The state of charge is modelled by the unbounded continuous place $P^C_{\text{e-car}}$. It can be filled by the continuous transition T^C_{charge} and emptied by $T^C_{\text{discharge}}$, both with nominal rate one (not shown). In the initial marking, T^C_{charge} is enabled but $T^C_{\text{discharge}}$ is not, due to the connected test arcs. The token in P^D_{charging} that enables T^C_{charge} is moved to P^D_{driving} when the general transition T^G_{stop} fires. After its firing, $T^C_{\text{discharge}}$ and the general transition T^G_{distance} become enabled. The latter models the (random) distance to be driven. When it fires, the token in P^D_{driving} vanishes and discharging stops. The decision of which car to take is modelled by the immediate transitions T^I_{electric} and T^I_{petrol}, which fire immediately after the firing of T^G_{stop} (due to the inhibitor arcs). Since T^I_{electric} and T^I_{petrol} require the same token in P^D_{start} for firing, only one can fire thereby disabling the other. This is called a *transition conflict*.

The electric car is thus charged over night (up to an uncertain amount of charge) and the decision of which car to take being made in the morning, just before departure. Note that, even if the petrol car is chosen, we model the discharging process of the electric car to be able to model-check whether the best choice of car was made, after driving is complete. Alternatively, if we exclude the inhibitor arcs, both transitions are enabled initially, before charging starts, such that the decision cannot depend on their firing times. We

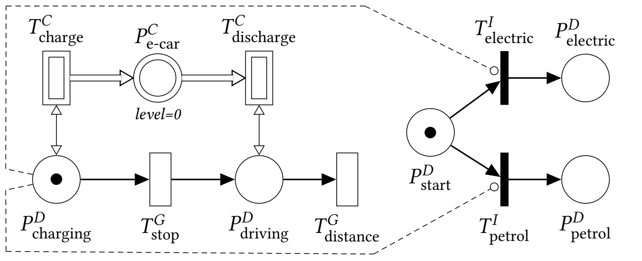


Figure 1: Running example HPnG

consider both of these model variants throughout this paper.

To the best of our knowledge, HPnGs are the only Petri net formalism which combines continuous dynamics and *general* distributions as required in this example. Alternatively, stochastic hybrid automata (SHA) [11] could have been used, however due to their generality they are more difficult to analyse.

2.1 Nondeterminism and Schedulers

Whenever multiple transitions can fire at the same point in time, there is naturally a nondeterministic choice between the transitions *in conflict*[1]. Previously, such nondeterminism was resolved using priorities and weights [15]. In this paper, we for the first time consider HPnGs without such a resolution. Instead, we properly include the nondeterminism in the HPnG analysis. As is standard in probabilistic model checking, we use the notion of *schedulers* to capture arbitrary fixed resolutions of nondeterminism:

Definition 2.5. Let $C^{\mathcal{T}}(\Gamma_n) \subset \mathcal{T} \setminus \mathcal{T}^C$ denote the set of transitions which are in conflict in a state Γ_n. A *discrete probability distribution* over $C^{\mathcal{T}}(\Gamma_n)$ is a function $\mu \colon C^{\mathcal{T}}(\Gamma_n) \to [0, 1]$, such that $support(\mu) = \{T \in C^{\mathcal{T}}(\Gamma_n) \mid \mu(T) > 0\}$ is countable and $\sum_{T \in support(\mu)} \mu(T) = 1$. We denote the set of probability distributions over $C^{\mathcal{T}}(\Gamma_n) \subset \mathcal{T}$ by $\mathrm{Dist}(C^{\mathcal{T}}(\Gamma_n))$.

Definition 2.6. A *scheduler* for an HPnG is a measurable function $\mathfrak{s} \colon Paths(\Gamma_0) \to \mathrm{Dist}(C^{\mathcal{T}}(\Gamma_n))$ that maps every finite path $\sigma = \Gamma_0 \xrightarrow{(\Delta\tau_0, \varepsilon_0)} \Gamma_1 \xrightarrow{(\Delta\tau_1, \varepsilon_1)} \dots \Gamma_n$, which ends in a conflict of transitions in Γ_n, to a discrete probability distribution over the transitions in $C^{\mathcal{T}}(\Gamma_n)$. Let $\mathfrak{S}^n$ denote the class of those schedulers.

This type of scheduler has *complete information* about the system's states, its decisions are *history-dependent*, and it is *randomised* since it returns a probability distribution. Randomised schedulers are in practice only relevant for the verification of multi-objective problems [9] and in certain certain constrained settings (see e.g. [18]); we thus restrict to *deterministic* schedulers that always pick one transition only in this paper. If we further limit the information given to a scheduler, we obtain restricted *scheduler classes*. For example, *memoryless* schedulers only see state Γ_n of a path, and *discrete-history* ones only see the discrete part of a state—the discrete marking, drifts, and enabling status, which only change at events—and the type of events but not their exact timings, i.e. they

[1]For simplicity of presentation, we assume absolutely continuous distributions, as the probability that two general transitions fire at the same time, or one fires at a specific time point, is then zero and conflicts involving general transitions can be neglected.

map a discrete path $(\Gamma_0.\mathbf{m}, \Gamma_0.\mathbf{d}, \Gamma_0.\mathbf{e}) \xrightarrow{\varepsilon_0} \dots \xrightarrow{\varepsilon_{n-1}} (\Gamma_n.\mathbf{m}, \Gamma_n.\mathbf{d}, \Gamma_n.\mathbf{e})$ to a discrete probability distribution. Let $\mathfrak{S}^n_{\mathrm{m}}$ denote the class of memoryless and $\mathfrak{S}^n_{\mathrm{d}}$ the class of discrete-history schedulers.

2.2 Prophetic and Non-Prophetic Scheduling

For stochastic automata (SA) [6], which are stochastic-timed systems with general distributions like HPnG but without continuous places, the landscape of scheduler classes has recently been mapped [5]. A general transition in an HPnG corresponds to a *clock* in an SA. A clock may be *reset*, at which point its *value* is set to 0 and its *expiration time* is sampled from the clock's probability distribution. Once a clock's value reaches its expiration time, it may enable an edge in the SA. Clock values correspond to enabling times in HPnG, and enabling an edge corresponds to firing a general transition. A notable difference in semantics is that a clock's (random) expiration time is determined the moment the clock is reset, while the states of an HPnG do not track predetermined firing times. The fact that expiration times are part of the states of an SA gives rise to *prophetic* scheduling [17] that takes these times into account to optimise decisions. In this way, prophetic schedulers can "see" the timing of "future" events. They are often considered unrealistically powerful since they can achieve higher (lower) maximal (minimal) probabilities than one expects if the timing of random events is assumed to be unpredictable.

Due to this difference in semantics, if we restrict to HPnG without continuous places, the schedulers of Definition 2.6 are clearly non-prophetic: they have no access to information about the exact future firing times of general transitions. However, in the full HPnG formalism, we can fill a continuous place with rate 1 during the enabling time of a general transition, as we do in our example. This transfers the firing time of a general transition into a continuous variable, but only after the firing. Yet the amount of fluid in the continuous place could subsequently be drained at rate −1 and an immediate transition be fired once empty. A scheduler can well observe the continuous marking and thus the time to fire for the immediate—but now in a sense stochastic—transition. We take the viewpoint that the choice of making the value of a random variable available via such an encoding or not is a conscious choice of the modeller, and schedulers that take such values into account should not be considered prophetic per se since they only make use of information that has been *explicitly* added to the model. Thus, we consider all the schedulers of Definition 2.6 to be non-prophetic.

In order to define prophetic schedulers for HPnG, we would modify the semantics to proceed like the semantics for SA: once a general transition becomes enabled for the first time since its last firing, the time to firing is drawn according to its associated probability distribution, and included in states. On this modified semantics, a scheduler according to Definition 2.6 would be prophetic and we denote this set of schedulers with $\mathfrak{S}^p$.

2.3 Time-Bounded Reachability

We are interested in the computation of time-bounded reachability, i.e. the probability of entering certain states with a total time delay of $\leq t_{max}$. Due to the presence of nondeterminism, there is no single such probability, but every scheduler applied to a nondeterministic HPnG induces a fully stochastic version of that HPnG.

Definition 2.7. Let $\Gamma_0 \in S$ be the initial state of a given HPnG $(\mathcal{P}, \mathcal{T}, \mathcal{A}, \mathbf{M}_0, \Phi)$. Let $\mathfrak{s} \in \mathfrak{S}^n \cup \mathfrak{S}^p$ be a scheduler and let $Prob(\sigma, \mathfrak{s})$ denote the probability for a single finite path $\sigma \in Paths(\Gamma_0, t_{max})$, starting in Γ_0, in which all transition conflicts are resolved by $\mathfrak{s}$. We define the probability that a set of states $S^{goal} \subseteq S$ is reached within a finite time bound $t_{max} \in \mathbb{R}^+_0$ as

$$p(S^{goal}, \mathfrak{s}, t_{max}) = \int_{Paths(\Gamma_0, t_{max})} \mathbf{1}_{S^{goal}}(\sigma, t_{max}) \cdot Prob(\sigma, \mathfrak{s})\, d\sigma, \quad (1)$$

with

$$\mathbf{1}_{S^{goal}}(\sigma, t_{max}) = \begin{cases} 1 & \text{if } \exists\, \Gamma \in S^{goal}, \tau \in [0, t_{max}] \colon \gamma(\sigma, \tau) = \Gamma, \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

For the definition of the probability $Prob(\sigma, \mathfrak{s})$, we refer to [27], where the definition of $Prob(\sigma)$ in Equation 2 has to be changed as follows: The probability for transition T to fire (first case) is replaced by the probability $\mathfrak{s}(\sigma)(T)$ given by the scheduler.

If we fix a class of schedulers $\mathfrak{S} \in \{\mathfrak{S}^n, \mathfrak{S}^p\}$, we obtain a range of probabilities $[p^{\mathfrak{S}}_{min}(S^{goal}, t_{max}), p^{\mathfrak{S}}_{max}(S^{goal}, t_{max})]$, where

$$p^{\mathfrak{S}}_{min}(S^{goal}, t_{max}) = \inf_{\mathfrak{s} \in \mathfrak{S}} p(S^{goal}, \mathfrak{s}, t_{max}) \quad \text{and}$$
$$p^{\mathfrak{S}}_{max}(S^{goal}, t_{max}) = \sup_{\mathfrak{s} \in \mathfrak{S}} p(S^{goal}, \mathfrak{s}, t_{max})$$

are the infimum, respectively the supremum, over the probabilities induced by all schedulers in that class. We refer to these optimal probabilities as minimum and maximum. The set of states S^{goal} is said to be *reachable* before time bound t_{max} if the minimum probability $p^{\mathfrak{S}}_{min}(S^{goal}, t_{max})$ is larger than zero.

2.4 The Parametric Location Tree

The computation of reachability in HPnGs is facilitated by the *parametric location tree* (PLT), which describes the evolution of an HPnG over time but abstracts from concrete probabilities. It was initially proposed for one general transition firing in [15] and recently extended for an arbitrary but finite number of such firings in the case that nondeterminism is not present in the model [20].

Every node in the PLT symbolically represents a set of states with the same discrete marking. The marking changes due to a *source event*, leading to a child node. The evolution of the continuous variables in the model might depend on the firing times of the general transition, hence different firing times lead to different successor nodes in the PLT. Every firing time of a general transition is a random variable that is distributed according to its continuous probability distribution. We denote these random variables by $s_1, s_2, \dots, s_n$ in the order they have been instantiated, where at any point in time n equals the number of general transition firings which have already occurred plus the number of general transitions which are currently enabled [21]. We prohibit cycles of transition firings, which potentially take no time, i.e. cycles of only immediate and general transitions, to prevent Zeno-behavior.

Consequently, the PLT symbolically describes the stochastic behaviour of an HPnG in terms of the random variables $s_1, s_2, \dots, s_n$ and its deterministic evolution by *linear functions* in those random variables that corresponds to past firings. This allows a symbolic representation that is independent of the probability distributions. The main idea of parametric analysis is to collect, for each random variable s_i, the values that lead to the same (discrete) evolution over time into intervals S_i. We call these intervals *potential domains*. The potential domains of any parent node are split into intervals for its child nodes, which are disjoint in case that no transition conflict arises. It thus follows that potential domains in a location only consist of a single interval per random variable [20].

Definition 2.8. Let $(\mathcal{P}, \mathcal{T}, \mathcal{A}, \mathbf{M}_0, \Phi)$ be an HPnG and $t_{max} \in \mathbb{R}_0^+$ a time-bound. The parametric location tree (PLT) of the HPnG is a tree $(\mathbf{V}, \mathbf{E}, v_{\Lambda_0})$, with $\mathbf{V}$ the set of nodes and $\mathbf{E}$ the set of edges. Every node $v_i \in \mathbf{V}$ is defined by a *parametric location* $\Lambda = (\text{ID}, t, \Gamma, \text{S})$ with unique identifier Λ.ID. The *entry time* $\Lambda.t$ is either a time point in $\mathbb{R}_0^+$ or a linear function of the random variables $s_1, s_2, \ldots, s_i$ present in the HPnG, i.e. $\Lambda.t = a_0 + a_1 \cdot s_1 + a_2 \cdot s_2 + \cdots + a_i \cdot s_i$ whereas $\forall j, 0 \leq j \leq i\colon a_j \in \mathbb{R}$ holds. $\Lambda.\Gamma \in S$ is a state of the HPnG.

The vector Λ.S collects the *potential domains*, where every $S_i \in \text{S}$ has lower and upper boundaries $S_i.l \leq S_i.u$. Both $S_i.l$ and $S_i.u$ are linear functions of the random variables, i.e. $S_i.l = b_0 + b_1 \cdot s_1 + b_2 \cdot s_2 + \cdots + b_{i-1} \cdot s_{i-1}$ and $S_i.u = c_0 + c_1 \cdot s_1 + c_2 \cdot s_2 + \cdots + c_{i-1} \cdot s_{i-1}$, having $\forall j, 0 \leq j \leq (i-1)\colon b_j, c_j \in \mathbb{R}$, such that $S_i \in \Lambda.\text{S}$ collects all values of s_i for which $\Lambda.\Gamma$ is reachable.

The node v_{Λ_0} is the root node of the PLT with $\Lambda_0.\Gamma = \Gamma_0$ and $\Lambda_0.\mathbf{S} = \mathbf{S}^\infty$ the set of unlimited domains for those random variables s_i that correspond to the first firing of initially enabled general transitions, i.e. $S_i^\infty.l = 0$ and $S_i^\infty.u = \infty$. An edge $e_i = (v_{\Lambda_j}, v_{\Lambda_k}) \in \mathbf{E}$ exists for $v_{\Lambda_j}, v_{\Lambda_k} \in \mathbf{V}$ if an event $\Upsilon(\Lambda_j.\Gamma, \Lambda_k.\Gamma)$ exists, such that any sequence of locations in the PLT from the root to a leaf node represents a path in $Paths(\Gamma_0, t_{max})$ of the HPnG.

Example (continued). Figure 2a shows the structure of the PLT for our example without inhibitor arcs. The nondeterministic choice is at the root location Λ_1, leading to Λ_2 when choosing T^I_{electric} and to Λ_3 otherwise. In both sub-trees, the next event is the firing of T^G_{stop} into location Λ_4 resp. Λ_5. From there, either T^G_{distance} fires next or $P^C_{\text{e-car}}$ becomes empty before T^G_{distance} fires. Table 2b summarises the values stored per location, with x the continuous variable that represents the amount of fluid in place $P^C_{\text{e-car}}$ (on the entry time point to the location) and $\dot{x}$ its drift. t is the entry time $\Lambda_i.t$ into a location Λ. s_0 resp. s_1 denotes the random variable of T^G_{stop} resp. T^G_{distance} and S_0 and S_1 the corresponding potential domains.

The potential domains in Λ.S of a location Λ can be represented by half spaces and the intersection of those half spaces constitutes an n-dimensional polytope $\mathbf{P}_{\Lambda.\text{S}}$ where $n = |\Lambda.\text{S}|$ (i.e. the number of past general transition firings up to Λ plus the number of general transitions enabled in Λ.) [21]. Λ is reachable within time bound t_{max} if $\Lambda.t < t_{max}$. Hence, in case the entry time $\Lambda.t$ is a linear function on random variables, Λ is reachable if there exist values for those random variables within Λ.S such that the function is smaller than t_{max}. For computing time-bounded reachability, the PLT is constructed up to time bound t_{max} and the potential domains are restricted to those values that satisfy the time bound requirement. The probability to reach Λ within time bound t_{max} then equals the probability that the value of every random variable s_i for $0 \leq i \leq n$ lies within its potential domain S_i in Λ.S:

$$Prob(\Lambda.\text{S}) = \int_{\Lambda.S_1.l}^{\Lambda.S_1.u} \int_{\Lambda.S_2.l}^{\Lambda.S_2.u} \ldots \int_{\Lambda.S_n.l}^{\Lambda.S_n.u} \prod_{i=1}^{n} g(s_i)\, ds_n \ldots ds_2\, ds_1, \quad (3)$$

where $g(s_i)$ is the probability density function for s_i. In practice, we compute these probabilities by numerical integration techniques in the same way as described in [20]. Note that Eq. 3 differs from the computation of the *transient probabilities* in [20]. The worst-case complexity of building the PLT is in $O(n^2 \cdot e)$ with n the number of random variables and e the number of events. For details on how a PLT is constructed, we refer to [20].

3 CHECKING NON-PROPHETIC SCHEDULERS

Our approach for computing time-bounded reachability probabilities for nondeterministic HPnG is a modification of the existing PLT-based one for the case where nondeterminism is resolved via priorities and weights [20, 21]. Since we keep conflicts as nondeterministic, the "conflict probability" of [20] is not needed. We recursively construct the PLT of a (non-Zeno) HPnG as in [20] for an arbitrary but finite number of general transition firings. The construction stops if a predefined model time bound t_{max} is reached or if no further events can occur; thus the number of parametric locations is finite. We now describe how to compute probabilities based on the PLT and give an algorithm to obtain optimal probabilities in the presence of nondeterminism.

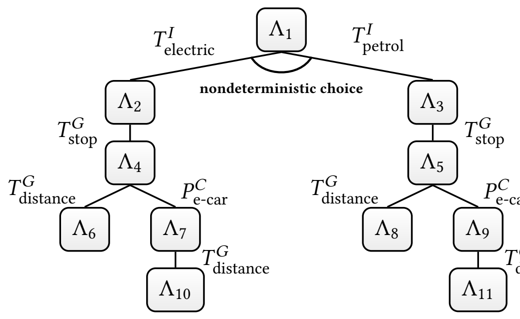
3.1 Computing Optimal Probabilities

Goal locations may be below nondeterministic choices resulting from conflicts in the PLT. The firing of every transition that participates in the conflict leads to a different successor location and thus a different subtree. To find the overall optimal probability of reaching any goal location starting from the root, we traverse the PLT bottom-up and compare the reachability probabilities of the subtrees in every location in which a conflict arises. We then take the maximum (minimum) of these probabilities to finally determine the overall maximum (minimum) probability. The collection of these decisions characterises a scheduler.

Recall from Section 2.1 that a discrete-history scheduler only sees the discrete markings, drifts, enabling status and the event types, but not their timings. Furthermore, a non-prophetic scheduler neither has information on future events nor can it take base decisions on the values of continuous variables. Since the (discrete) history of a location (i.e. the sequence of locations in the PLT back to the root) is unique, a discrete-history non-prophetic scheduler always takes the same decision per location. Hence, a scheduler of this class will always pick the same sub-tree in the PLT and consequently, the maximum probability (starting from that location) cannot be larger than the maximum over the single probabilities of the outgoing conflicting sub-trees (plus the sum over additional non-conflicting child locations):

Let Λ_0 be the location of a PLT's root node and for any location Λ_i, let $C(\Lambda_i)$ be the (possibly empty) set of conflicting child locations and $N(\Lambda_i)$ the set of non-conflicting child locations. The maximum probability of the optimal discrete-history non-prophetic scheduler in $\mathfrak{S}_d^n$ to reach any goal location in Loc^* is $p_{\max}(\Lambda_0)$ with

$$p_{\max}(\Lambda_i) = \begin{cases} Prob(\Lambda_i.\text{S}) & \text{if } \Lambda_i \in Loc^*, \\ \max_{\Lambda_{i+1} \in C(\Lambda_i)} \; p_{\max}(\Lambda_{i+1}) & \\ + \sum_{\Lambda_{i+1} \in N(\Lambda_i)} \; p_{\max}(\Lambda_{i+1}) & \text{otherwise.} \end{cases} \quad (4)$$



(a) Basic structure

	$\dot{x}$	x	1 token each in:	t	$[S_0.l, S_0.u]$	$[S_1.l, S_1.u]$
Λ_1	1	0	$P^D_{\text{charging}}, P^D_{\text{start}}$	0	$[0, \infty]$	disabled
Λ_2	1	0	$P^D_{\text{charging}}, P^D_{\text{electric}}$	0	$[0, \infty]$	disabled
Λ_3	1	0	$P^D_{\text{charging}}, P^D_{\text{petrol}}$	0	$[0, \infty]$	disabled
Λ_4	-1	s_0	$P^D_{\text{driving}}, P^D_{\text{electric}}$	s_0	$[0, \infty]$	$[0, \infty]$
Λ_5	-1	s_0	$P^D_{\text{driving}}, P^D_{\text{petrol}}$	s_0	$[0, \infty]$	$[0, \infty]$
Λ_6	0	$s_0 - s_1$	P^D_{electric}	$s_0 + s_1$	$[0, \infty]$	$[0, s_0]$
Λ_7	0	0	$P^D_{\text{driving}}, P^D_{\text{electric}}$	$2 \cdot s_0$	$[0, \infty]$	$[s_0, \infty]$
Λ_8	0	$s_0 - s_1$	P^D_{petrol}	$s_0 + s_1$	$[0, \infty]$	$[0, s_0]$
Λ_9	0	0	$P^D_{\text{driving}}, P^D_{\text{petrol}}$	$2 \cdot s_0$	$[0, \infty]$	$[s_0, \infty]$
Λ_{10}	0	0	P^D_{electric}	$s_0 + s_1$	$[0, \infty]$	$[s_0, \infty]$
Λ_{11}	0	0	P^D_{petrol}	$s_0 + s_1$	$[0, \infty]$	$[s_0, \infty]$

(b) Content of Λ_i

Figure 2: Parametric location tree for the running example (without inhibitor arcs)

Algorithm 1 presents pseudocode for the maximum computation, given a PLT. To compute the minimum probability, perform a less-than comparison in line 12. Function `solveNondetNonProph` is initially called with Λ being the root location and Loc^* being the set of goal locations. At first, function `isGoalLoc` checks if the parent location Λ is included in Loc^*, and if so, function `computeProb` computes the reachability probability of Λ as described in Sect. 2.3 (lines 3–5). Otherwise, all child locations of Λ whose source events are conflicting transitions are collected into the set *conflictingLocs* (line 6). If *conflictingLocs* is not empty (lines 8–22), all locations in that set are traversed (lines 10–21). For each of the locations Λ_c in *conflictingLocs*, the main function is called recursively, returning the maximum reachability probability for the subtree of Λ_c (line 11). If Λ_c is the first considered location or if the computed probability is greater than the currently stored value of *optimalProb*, a new optimal decision was found. In this case, the set of optimal locations and *optimalProb* are updated to the computed maximum probability (lines 13–15). If the result is equal to the current value of *optimalProb*, Λ_c is added to the existing list *optimalLocs* (lines 17–20), as multiple scheduler decisions lead to the maximum probability. After *conflictingLocs* is completely processed, the main function is again recursively called for each location in *nonConflictingLocs* and their probabilities are added to *optimalProb* (lines 23–28): Locations with source events that are not in conflict have different entry times. Hence, depending on the values of the random variables, *either* one of the non-conflicting events *or* the conflict occurs. Probabilities for non-conflicting nodes can be summed up as they depend on the disjoint potential domains. Finally, *optimalProb* is returned together with *optimalLocs* (line 29).

The PLT is recursively processed by post-order traversal and probabilities are only computed for goal locations. We obtain a worst-case complexity for Algorithm 1 of $O(l + g \cdot f_i(n))$ with l the number of locations, $g \leq l$ the number of goal locations and $f_i(n)$ the complexity of the integration for n random variables. The algorithm terminates as the tree is finite.

Optimality. Alg. 1 computes $p_{\max}(\Lambda_i)$, as in Eq. 4. If Λ_i is a goal location, $Prob(\Lambda_i.\mathrm{S})$ is determined according to Equation 3 in lines 3–4. Otherwise, lines 6–7 determine $C(\Lambda)$ as *conflictingLocs* and $N(\Lambda)$ as *nonConflictingLocs*. Line 11 computes $p_{\max}(\Lambda_c)$ for every

Algorithm 1 `solveNondetNonProph`(Λ, Loc^*)

```
1:  optimalProb = 0;
2:  optimalLocs = new list;
3:  if (isGoalLoc($\Lambda$, $Loc^*$)) then
4:     optimalProb = computeProb($\Lambda$);
5:  else
6:     conflictingLocs = determineConflictingLocations($\Lambda$);
7:     nonConflictingLocs = childLocations($\Lambda$) - conflictingLocs;
8:     if (size(conflictingLocs) > 0) then
9:        first = true;
10:       for ($\Lambda_c$ : conflictingLocs) do
11:          (currentProb, currentLocs) = solveNondetNonProph($\Lambda_c$, $Loc^*$);
12:          if (first or currentProb > optimalProb) then
13:             optimalProb = currentProb;
14:             optimalLocs = currentLocs;
15:             optimalLocs.insert($\Lambda_c$);
16:             first = false;
17:          else if (currentProb == optimalProb) then
18:             optimalLocs.insert(currentLocs);
19:             optimalLocs.insert($\Lambda_c$);
20:          end if
21:       end for
22:    end if
23:    for ($\Lambda_c$ : nonConflictingLocs) do
24:       (currentProb, currentLocs) = solveNondetNonProph($\Lambda_c$, $Loc^*$);
25:       optimalProb = optimalProb + currentProb;
26:       optimalLocs.insert(currentLocs);
27:    end for
28: end if
29: return (optimalProb, optimalLocs);
```

$\Lambda_c \in C(\Lambda)$. The maximum over all results is determined by the comparison in line 12 and stored as *optimalProb*. In lines 23–27, the sum of $p_{\max}(\Lambda_c)$ over all $\Lambda_c \in N(\Lambda)$ is computed and added to $p_{\max}(\Lambda)$.

Schedulers, discrete case. If we restrict to HPnG without continuous places, the locations in the PLT contain no information about future firing times, even for already enabled general transitions. The history of reaching a location w.r.t. discrete events (but not their exact timing) is however unambiguously encoded in the PLT (as the sequence of locations from the root to the considered location). Different locations may represent identical sets of states, however with a different history. A scheduler thus can take different decisions for identical states dependent on the history.

Concurrently enabled general transitions in a (parent) location lead to a conditioning in the potential domains of the corresponding random variables in the child locations. This allows a scheduler to take the relative values of the random variables into account when taking its decision in any location, where at least one of the general transition has fired, but the other ones have not fired yet. However, in such a location, this is anyway observable as part of the discrete history (through the firing order of the general transitions encoded in the structure of the PLT). Hence, our algorithm computes optimal probabilities for, and finds schedulers from, the class of *discrete-history non-prophetic* schedulers.

Schedulers, continuous case. Considering the full HPnG formalism, recall from Sect. 2.2 that stochastic firing times can be "stored" in continuous places, but we do not consider this as allowing prophetic scheduling. Locations in the PLT store the fluid levels of such places, but only *symbolically* as linear functions of the random values describing *previous* firings of general transitions. Exact values are still not available to schedulers in our algorithm. For these reasons, and since continuous information is only revealed in a symbolic fashion as induced by discrete events, the class of schedulers we consider in our algorithm remains that of discrete-history non-prophetic schedulers even for general HPnGs.

3.2 Non-Prophetic Scheduling in the Example

Let us come back to our running example, again without inhibitor arcs. We prefer taking the electric car but want to avoid the case where the available charge is insufficient for the (random) driving distance. Accordingly, we want to reach a state where

(a) the electric car is taken and the charge is sufficient (i.e. the battery is not empty when discharging stops), or
(b) the petrol-powered car is taken and the charge of the electric car *would not have been* sufficient.

So, formally, we want to reach a marking where $(m_{\text{electric}} = 1 \wedge m_{\text{driving}} = 0 \wedge c_{\text{e-car}} > 0) \vee (m_{\text{petrol}} = 1 \wedge m_{\text{driving}} = 1 \wedge c_{\text{e-car}} = 0)$ holds. In the PLT as presented in Figure 2a, the corresponding goal locations are Λ_6 and Λ_9. Considering the variant without inhibitor arcs results in T^I_{electric} and T^I_{petrol} being initially enabled and in conflict. The optimal *non-prophetic* decision can depend on the *distributions* of the two random variables for T^G_{stop} and T^G_{distance}, but not on their (future) *firing times.*

We list max. reachability probabilities (fourth column) and the corresponding non-prophetic schedulers decisions (fifth column) for 4 scenarios with different probability distributions and $t_{max} = 48$ in Table 1. The distributions for both general transitions are given in the second and third column. Let 1 time unit in the model be 1 h. It took approx. 13 ms to build the PLT[2]. We obtained probabilities via Monte Carlo integration [24] for the multi-dimensional integrals; we include its statistical error in the last column.

In scenario 1a, the expected charging time is significantly larger than the expected value of the distance, both with low variance σ^2. Consequently, T^I_{electric} is the better choice in most cases and leads to a maximum probability p_{max} close to 1. Larger variances in 1b lead to a lower probability as it is now more likely that the distance is larger than the state of charge allows. The distribution parameters for both transitions are equal in 1c and 1d, resulting in a probability of 0.5 for each immediate transition to reach a goal state. The deviation of the computed maximum probability from 0.5 is due to the numerical integration, but lies within the error estimate. Overall, these results show that the resulting probabilities depend on the probability distributions assigned to the general transitions. A non-prophetic scheduler cannot optimise further.

[2]All computations were done on a VirtualBox running Ubuntu 18.04 with 2x2.7 GHz Intel Core i5 processor and 4 GB of RAM; C++17 Code compiled with GCC 7.4.0.

4 CHECKING PROPHETIC SCHEDULERS

Non-prophetic schedulers may be more "realistic" and desirable in practice, yet they are a somewhat arbitrarily restricted class at least in SA where expiration times are part of states. We now describe a way towards model checking HPnG w.r.t. classic prophetic schedulers: we use continuous places to store firing times and make the continuous marking of the current state available to schedulers. Recall that we call a random variable *pre-computed* if the delay until the corresponding general transition fires is made explicit via the amount of fluid in a continuous place. For models in which at least one random variable is pre-computed, a scheduler that observes the fluid level of the corresponding continuous place can make prophetic decisions for that variable and thereby perform better than a discrete-history non-prophetic scheduler. Models can be adapted to pre-compute all random variables as follows:

(1) Add one continuous place P^C for every general transition T^G.
(2) Fill P^C with drift 1 as long as T^G is enabled.
(3) After T^G has fired, drain P^C with drift -1.
(4) Replace the firing of T^G by the event of P^C reaching fluid level 0.

Step 4 may require additional changes throughout the model depending on the interplay of components in the original system; for time-bounded properties, it may be necessary to pause the overall system evolution during pre-computation.

Example (continued). In our running example, the firing time of T^G_{stop} is stored in continuous place $P^C_{\text{e-car}}$, by nature of the model, and this place is drained after firing via $T^C_{\text{discharge}}$ with rate one. Yet to make the value of the random variable available to a prophetic scheduler, the decision of which car to take needs to be delayed to after T^G_{stop} has fired. This is achieved by the dashed inhibitor arcs in Figure 1, which we consider to be included from now on.

The values of the pre-computed random variables are stored in the states of the adapted model. However, Algorithm 1 for non-prophetic schedulers does not take this information into account. Therefore, we now present a different approach for prophetic schedulers, where decision can depend on the random variables.

4.1 Determining Split Hyperplanes

The optimal prophetic decision might depend on the order of random variables and their values. The former may not and the latter definitely is not represented in the PLT unless random variables are pre-computed. However, if they are, then taking the continuous values into account when maximising or minimising reachability probabilities leads to an optimisation problem over the potential domains of the pre-computed random variables. Solving this optimisation problem results in a partitioning of the potential domains into subsets for which different decisions are optimal. The partitions are characterised by *split hyperplanes* separating the space of all random variables into more restricted domains by half space intersection. Those half spaces represent the scheduler decisions,

Table 1: Case study results for non-prophetic schedulers

scenario	T^G_{stop}	T^G_{distance}	optimal scheduler decision	p_{max}	error estimate	run time
1a)	folded normal ($\mu = 12\,\text{h}, \sigma^2 = 1\,\text{h}$)	folded normal ($\mu = 8\,\text{h}, \sigma^2 = 1\,\text{h}$)	T^I_{electric}	0.998277	$\pm 4.33E^{-3}$	2.238s
1b)	folded normal ($\mu = 12\,\text{h}, \sigma^2 = 16\,\text{h}$)	folded normal ($\mu = 8\,\text{h}, \sigma^2 = 16\,\text{h}$)	T^I_{electric}	0.758743	$\pm 1.23E^{-4}$	1.171s
1c)	folded normal ($\mu = 8\,\text{h}, \sigma^2 = 1\,\text{h}$)	folded normal ($\mu = 8\,\text{h}, \sigma^2 = 1\,\text{h}$)	$T^I_{\text{electric}}, T^I_{\text{petrol}}$	0.500274	$\pm 8.23E^{-4}$	2.342s
1d)	uniform ($7\,\text{h}, 9\,\text{h}$)	uniform ($7\,\text{h}, 9\,\text{h}$)	$T^I_{\text{electric}}, T^I_{\text{petrol}}$	0.499716	$\pm 2.08E^{-3}$	0.640s

and the split hyperplanes can only depend on the values of the pre-computed random variables (known to the scheduler). The optimal scheduler in $\mathfrak{S}^p$ then relates to the optimal split hyperplanes that maximise (minimise) the reachability probabilities, which are computed over the restricted potential domains (recall Equation 3).

Let Λ denote a location in the PLT with a conflict between transitions $T_j, j \in \{1, ..., k\}$. The firing of T_j leads to location Λ_j. We traverse the k sub-trees with roots $\Lambda_1 ... \Lambda_k$ using depth-first search and identify all locations that represent states in the set of goal states. Let Loc^* denote the set of those goal locations. For $n = |\Lambda.\mathrm{S}|$, let m with $1 \leq m \leq n$ be the number of pre-computed random variables (visible to the prophetic scheduler at $\Lambda.t$). Recall from Section 2.4 that the potential domains $\Lambda_i.\mathrm{S}$ can geometrically be represented by half spaces, whose intersection constitutes a convex polytope $\mathbf{P}_{\Lambda_i.\mathrm{S}}$. Our goal is to find additional hyperplanes

$$H_i = \{(s_1, ..., s_n) \in \mathbb{R}^n \mid a_1 s_1 + ... + a_m s_m = b\},$$

that only depend on the m pre-computed random variables. For each pair of goal locations reached via different choices, up to two hyperplanes might be required to separate the polytopes, to maximize (minimize) the total probability over these domains[3].

Let $\mathbf{H}$ denote the set of those hyperplanes, i.e. $|\mathbf{H}| \leq |Loc^*|$. Any hyperplane $H_i \in \mathbf{H}$ separates the n-dimensional space into the two (open) half spaces, which we denote by

$$\mathbf{h}_i^{\sim} = \{(s_1, ..., s_n) \in \mathbb{R}^n \mid a_1 s_1 + ... + a_m s_m \sim b\},$$

for $\sim \in \{\leq, \geq\}$. Let $Loc^*_j \subset Loc^*$ denote all goal locations in the subtree of Λ_j. For each location $\Lambda_c \in Loc^*_j$, the polytope $\mathbf{P}_{\Lambda_c.\mathrm{S}}$ has to be intersected for each hyperplane H_i with one of the half spaces $\mathbf{h}_i^{\leq}$ and $\mathbf{h}_i^{\geq}$. This leads to a restriction of $\Lambda_c.\mathrm{S}$, which we denote by $\Lambda_c.\mathrm{S}'$. We formalise the optimisation problem for maximising the probability to reach a goal location from Loc^* from a nondeterministic conflict in Λ as:

$$\begin{aligned}
&\max Prob(Loc^*_1, ..., Loc^*_k, \mathbf{H}) = \textstyle\sum_{j=1}^{k} \sum_{\Lambda_c \in Loc^*_j} Prob(\Lambda_c.\mathrm{S}') \\
&\text{subject to constraints} \\
&\forall j \in \{1, ..., k\}\colon \forall l \in \{1, ..., k\}\colon l \neq j\colon \forall (\Lambda_c, \Lambda_d) \in Loc^*_j \times Loc^*_l\colon \\
&\exists\, H_r, H_s \in \mathbf{H} \land \exists \bowtie, \star, \diamond, \triangledown \in \{\leq, \geq\} \text{ with } \bowtie \neq \diamond \land \star \neq \triangledown\colon \\
&\quad \mathbf{P}_{\Lambda_c.\mathrm{S}'} \text{ equals } \mathbf{P}_{\Lambda_c.\mathrm{S}} \text{ intersected by the half spaces } \mathbf{h}_r^{\bowtie} \text{ and } \mathbf{h}_s^{\star} \\
&\land\, \mathbf{P}_{\Lambda_d.\mathrm{S}'} \text{ equals } \mathbf{P}_{\Lambda_d.\mathrm{S}} \text{ intersected by the half spaces } \mathbf{h}_r^{\diamond} \text{ and } \mathbf{h}_s^{\triangledown}.
\end{aligned} \tag{5}$$

The formulation for the minimum probability is analogous. The optimal scheduler decision results from checking in which of the half spaces the values of the pre-computed random variables lie.

[3]Since the polytopes are convex, one hyperplane is in general sufficient to separate two polytopes according to the hyperplane separation theorem. However, due to the restriction to hyperplanes which can only depend on the m pre-computed random variables, two hyperplanes might be necessary if $m < n$.

4.2 Algorithm Outline

Algorithm 2 shows the pseudocode for computing max. probabilities under prophetic scheduling taking pre-computed continuous values into account. Function `solveNondetProph` is initially called with Λ being the root location, Loc^* the set of goal locations, and m the number of pre-computed random variables. Compared to Algorithm 1, we handle locations with conflicts differently.

Again, if the parent location Λ is a goal location, we compute its reachability probability (lines 3–5). Otherwise, we traverse every sub-tree in a pairwise manner, using depth-first search, and collect all goal locations which are the firstly reached in a path from the root. Those locations are saved separately per nondeterministic choice (lines 8–12). For every pair of such nondeterministic choices, we traverse the Cartesian product over the resulting two sets of goal states, such that we consider pairs of goal locations Λ_c and Λ_d (which are reached from different nondeterministic choices; lines 13–25). To separate the potential domains $\Lambda_c.\mathrm{S}$ and $\Lambda_d.\mathrm{S}$, we then solve the optimisation problem of Equation 5 for Λ_c and Λ_d in line 18, returning the split hyperplanes h_1 and h_2. In addition to the computation of the hyperplanes, we also have to determine which decision is taken for which resulting half space, which can be easily realised by comparing both possibilities. Therefore, we let the optimisation solver return the corresponding Booleans dir_1 and dir_2, too. The polytopes $\mathbf{P}_{\Lambda_c.\mathrm{S}}$ and $\mathbf{P}_{\Lambda_d.\mathrm{S}}$ are intersected by the half spaces h_1 and h_2 (lines 19–22), and the latter are stored in *splits* (lines 23–24). Since the potential domains are updated, but no probabilities are computed, the recursive call is done for all (conflicting and non-conflicting) child locations in lines 29–33. Note that this is different to Algorithm 1. Finally, the optimal probability as well as the *splits* list are returned in line 35.

Let c denote the number of choices in a single nondeterministic conflict and g_j the number of goal locations, i.e. polytopes to consider, in a sub-tree related to the choice of location Λ_j. Since nondeterministic choices are considered in a pairwise manner and further pairs of goal locations are considered via the Cartesian product, the number of split hyperplanes is in $O(c^2 \cdot g^2_{max})$ with $g_{max} = \max_{j \in \{1,...,c\}}\{\, g_j \,\}$. The number of free variables in the optimisation problem for a single conflict is in $O(c^2 \cdot g^2_{max} \cdot m)$. Let l denote the total number of locations and $g \leq l$ the number of goal locations. As $c \leq l$ and $g_{max} \leq g$ holds for every conflict, we denote the complexity of solving the optimisation problem by $f_o(l, g, m)$. The PLT is again recursively processed by post-order traversal and again reachability is only computed for goal locations. Thus, the worst-case runtime for Algorithm 2 is in

$$O(g \cdot f_i(n) + l^2 \cdot g^2 \cdot f_o(l, g, m)),$$

where $f_i(n)$ is the complexity of integration for n random variables.

Algorithm 2 solveNondetProph(Λ, Loc^*, m)

```
optimalProb = 0;
splits = new list;
if (isGoalLoc($\Lambda$, $Loc^*$)) then
   optimalProb = optimalProb + computeProb($\Lambda$);
else
   conflictingLocs = determineConflictingLocations($\Lambda$);
   if (size(conflictingLocs) > 0) then
      goalLocs = new list;
      goalLocs.insert(getGoalLocations(conflictingLocs[0]));
      for (j=0, j < size(conflictingLocs)-1, j++) do
         for (k = j + 1, k < size(conflictingLocs), k++) do
            goalLocs.insert(getGoalLocations(conflictingLocs[k]));
            for each pair ($\Lambda_c$, $\Lambda_d$) $\in$ (goalLocs[j] $\times$ goalLocs[k]) do
               $h_1$ = new array[m+1];
               $h_2$ = new array[m+1];
               $dir_1$ = false;
               $dir_2$ = false;
               ($h_1$, $dir_1$, $h_2$, $dir_2$) = solveOptimisationProblem($\Lambda_c$, $\Lambda_d$);
               $\mathbf{P}_{\Lambda_c}$.S = $\mathbf{P}_{\Lambda_c}$.S.intersect(Halfspace($h_1$, $dir_1$));
               $\mathbf{P}_{\Lambda_c}$.S = $\mathbf{P}_{\Lambda_c}$.S.intersect(Halfspace($h_2$, $dir_2$));
               $\mathbf{P}_{\Lambda_d}$.S = $\mathbf{P}_{\Lambda_d}$.S.intersect(Halfspace($h_1$, $dir_1$));
               $\mathbf{P}_{\Lambda_d}$.S = $\mathbf{P}_{\Lambda_d}$.S.intersect(Halfspace($h_2$, $dir_2$));
               splits.insert($\Lambda_j$, $\Lambda_k$, $h_1$, $dir_1$);
               splits.insert($\Lambda_j$, $\Lambda_k$, $h_2$, $dir_2$);
            end for
         end for
      end for
   end if
   for ($\Lambda_c$ : childLocations($\Lambda$)) do
      (currentProb, currentSplits) = solveNondetProph($\Lambda_c$, $Loc^*$, $m$);
      optimalProb = optimalProb + currentProb;
      splits.insert(currentSplits);
   end for
end if
return (optimalProb, splits);
```

The main challenge in implementing Algorithm 2 is to find an appropriate solver for an optimisation problem of such size, i.e. the implementation of the function solveOptimisationProblem. We have implemented the algorithm for the case of only two random variables and a single split line being sufficient. Our implementation uses the ALGLIB library (available at alglib.net) for optimisation. In addition to our case study on the running example, we have computed prophetic schedulers for two HPnG encodings of models from [5]. The results we obtain match the expected results.

Optimality. Every pair of restricted polytopes of goal locations reachable from different scheduler decisions must have empty intersection to obtain a valid scheduler. Algorithm 2 iterates every pair of such polytopes and separates them by split hyperplanes. The final set of hyperplanes yields the optimal scheduler. The probability computed by Algorithm 2 depends on the number of pre-computed random variables, as this constitutes the amount of information available. If all random variables are pre-computed, the scheduler has complete information and the algorithm computes the best possible result. If only some random variables are pre-computed, the computed result is optimal w.r.t. the available information, as the set of possible split hyperplanes is restricted. The resulting probability is the same for pre-computing the random variables in parallel or in sequence. However, the former is computationally more efficient as the split hyperplanes are already encoded in the PLT.

Table 2: Results for prophetic scheduling, partial adaptation

sc.	split value a	decision for $s_{\text{stop}} > a$	p_{max}	error	run time
2a	8.59669 h	T^I_{electric}	0.997904	$\pm 2.23E^{-4}$	765.645s
2b	7.99520 h	T^I_{electric}	0.816482	$\pm 3.54E^{-6}$	157.588s
2c	8.00360 h	T^I_{electric}	0.749933	$\pm 5.43E^{-5}$	100.076s
2d	7.99893 h	T^I_{electric}	0.750096	$\pm 2.02E^{-4}$	335.100s

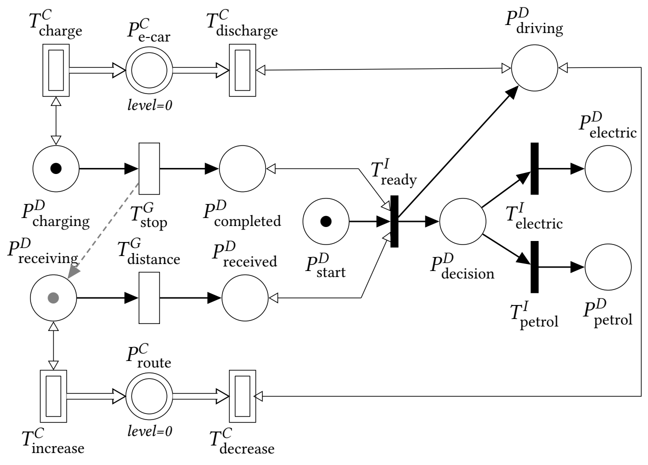


Figure 3: Model adaptation for the running example

4.3 Prophetic Scheduling in the Example

The non-prophetic case in our running example corresponds to the owner of the cars having to decide which one to take before the electric car starts charging (for a random amount of time). We now present optimal reachability probabilities obtained by prophetic schedulers (using the example with the inhibitor arcs).

Pre-computation of charge only. We first consider the prophetic case where we only know the firing time of T^G_{stop}, but not of T^G_{distance}. In this setting, the decision between cars is made once the electric car is charged (and the charge can be read from its displays), but before the driving route is known. Recall that the firing time of T^G_{stop} is already pre-computed via the fluid level of $P^C_{\text{e-car}}$.

The resulting maximum probabilities and schedulers are summarised in Table 2, using the same four scenarios as in Table 1. The second column presents the *split value a*, such that the prophetic schedulers choose the electric car if the pre-computed state of charge s_{stop} exceeds a (third column) to achieve the maximum reachability probability (fourth column).

The resulting split line $H_a = \{(s_{\text{stop}}, s_{\text{distance}}) \in \mathbb{R}^2 \mid s_{\text{stop}} = a\}$, depends on the distribution of T^G_{distance}. For all scenarios, the split values lie around 8, which is the mean of the distribution of T^G_{distance} and thus its expected firing time. The probability of scenario 2a only slightly deviates from scenario 1a (due to the integration error). Always choosing T^I_{electric} already yields a probability close to one, as the firing time of T^G_{stop} is most likely above eight due to the low variance of the distribution. Since in this case slightly changing the split value only results in small changes in the resulting probability, the computation of a is less accurate. The remaining scenarios

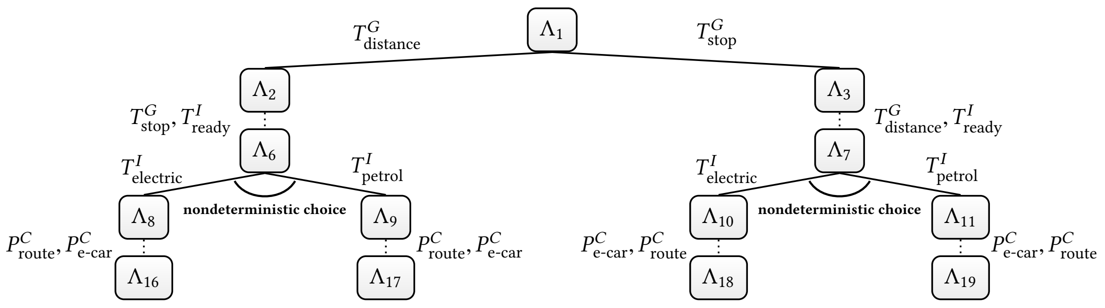


Figure 4: Basic structure of the PLT for the adapted model with pre-computing all random variables in parallel

in Table 2 demonstrate how prophetic scheduling can increase p_{max} for the desired property by taking decisions depending on the pre-computed firing time of T^G_{stop}. Comparing the run times of Tables 1 and 2 shows that the computation of prophetic schedulers in general takes longer. The reason is that optimisation requires a large number of integrations for different potential split values a until the optimal value is found.

Pre-computation of charge and distance. The example can further be adapted to also pre-compute T^G_{distance}. A prophetic scheduler then knows both firing times before taking its decision. This would apply if the owner of the cars also knows the route to be driven before leaving. The firing times can be pre-computed either in parallel or in sequence. Figure 3 shows the adapted model for both options. If the gray-dashed arc from T^G_{stop} to $P^D_{\text{receiving}}$ is excluded and initially there is a token in $P^D_{\text{receiving}}$, both general transitions are pre-computed in parallel. Including the arc and removing the tokens results in enabling T^G_{distance} only when T^G_{stop} has been pre-computed. We added continuous place P^C_{route} and two continuous transitions to pre-compute T^G_{distance}. The nondeterministic decision is delayed by the immediate transition T^I_{ready}, which only fires after the pre-computation. The outflow transitions $T^C_{\text{discharge}}$ and T^C_{decrease} become enabled after T^I_{ready} has fired.

Table 3 shows the maximum probabilities computed by both non-prophetic (discrete-history) and prophetic schedulers (as implemented for a single split line) for the adapted model with pre-computing the random variables in parallel. The results are approximately one in all scenarios. This is because the *discrete* event of either one of the pre-computations finishing first is encoded in the PLT, of which an abstraction is shown in Figure 4. The tree branches into two subtrees encoding the *relation* of the firing times of the general transitions. This results in two locations (here Λ_6 and Λ_7) with a nondeterministic choice. As described in Sect. 3.1, even a non-prophetic scheduler can then base its decision on the ordering of the firing times. In the example, choosing T^I_{electric} is optimal if T^G_{distance} has fired first, and otherwise T^I_{petrol} is optimal.

Table 3: Results for prophetic scheduling, full adaptation

	non-prophetic			prophetic		
sc.	p_{max}	error	time	p_{max}	error	time
3a)	0.999798	$\pm 9.07E^{-3}$	2.214s	1.000000	$\pm 4.41E^{-4}$	706.791s
3b)	0.998012	$\pm 1.16E^{-2}$	2.408s	0.9986220	$\pm 1.93E^{-4}$	318.626s
3c)	1.000000	$\pm 1.57E^{-2}$	4.480s	0.999869	$\pm 5.69E^{-4}$	892.401s
3d)	0.999432	$\pm 1.90E^{-2}$	1.094s	0.999438	$\pm 1.08E^{-3}$	439.169s

If both general transitions are pre-computed in sequence, no information on their relative size is encoded in the (discrete) PLT and the prophetic scheduler should perform better than the non-prophetic one due to the extra optimisation step. The non-prophetic results for this example match those of Table 1. Computing an optimal prophetic scheduler for this case yields an optimisation problem with multiple free variables, which our implementation does not support due to a lack of efficient solvers. Yet pre-computing all random variables of the model is a special case in which the scheduler has complete information on the future evolution and the required split hyperplanes can be derived from the potential domains of the goal locations (via union and intersection). For our example, this leads to $p_{\max} \approx 1$ in all scenarios, but requires less computation time than computing split hyperplanes.

5 CONCLUSION

We expanded the reach of model checking for HPnGs to cope with nondeterminism that naturally arises from transition conflicts. Our new algorithm computes optimal probabilities ranging over all possible resolutions of nondeterminism as captured by *discrete-history non-prophetic* schedulers. Non-prophetic scheduling is highly desirable for stochastic timed systems with non-memoryless probability distributions. Nevertheless, we investigated how to extend the algorithm to check the theoretically more appropriate class of *prophetic* schedulers. This is possible by exploiting the ability of HPnG to capture continuous dynamics, but requires efficient optimisation solvers. We provide pseudocode and a proof-of-concept implementation for this problem. Overall, the case study and its evaluation with our implementation show that our approach is feasible, for both types of schedulers. The example also highlights the flexibility of HPnG in allowing not only fully non-prophetic and fully prophetic scheduling, but also a flexible selection between the two, as appropriate for every random variable. In contrast to previous approaches to checking nondeterministic hybrid systems with general probability distributions that deliver safe over- or underapproximations of probabilities, such as the one for (the more general model of) SHA of [16], our algorithm provides exact results (up to the statistical error of the Monte Carlo integration).

Data availability. The implementation and test files are available as part of the HPnmG tool at github.com/jannikhuels/hpnmg, including the full PLTs of figs. 2a and 4.

ACKNOWLEDGMENTS

This work was partly supported by NWO VENI grant 639.021.754.

REFERENCES

[1] Marco Ajmone Marsan, Gianfranco Balbo, Gianni Conte, Susanna Donatelli, and Giuliana Franceschinis. 1994. *Modelling with Generalized Stochastic Petri Nets* (1 ed.). John Wiley & Sons, New York, NY.

[2] Hassane Alla and René David. 1998. Continuous and hybrid Petri nets. *Journal of Circuits, Systems, and Computers* 8, 1 (1998), 159–188.

[3] Karl J. Åström. 1965. Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications* 10, 1 (1965), 174–205.

[4] Mario Bravetti and Pedro R. D'Argenio. 2004. Tutte le Algebre Insieme: Concepts, Discussions and Relations of Stochastic Process Algebras with General Distributions. In *Validation of Stochastic Systems*, Christel Baier, Boudewijn Haverkort, Holger Hermanns, Joost-Pieter Katoen, and Markus Siegle (Eds.). LNCS, Vol. 2925. Springer, Berlin, Heidelberg, 44–88.

[5] Pedro R. D'Argenio, Marcus Gerhold, Arnd Hartmanns, and Sean Sedwards. 2018. A Hierarchy of Scheduler Classes for Stochastic Automata. In *International Conference on Foundations of Software Science and Computation Structures, FoSSaCS 2018 (LNCS)*, Vol. 10803. Springer, Cham, 384–402.

[6] Pedro R. D'Argenio and Joost-Pieter Katoen. 2005. A theory of stochastic systems part I: Stochastic automata. *Information and Computation* 203, 1 (2005), 1–38.

[7] Christian Eisentraut, Holger Hermanns, Joost-Pieter Katoen, and Lijun Zhang. 2013. A Semantics for Every GSPN. In *International Conference on Application and Theory of Petri Nets and Concurrency, PETRI NETS 2013 (LNCS)*, Vol. 7927. Springer, Berlin, Heidelberg, 90–109.

[8] Christian Eisentraut, Holger Hermanns, and Lijun Zhang. 2010. On Probabilistic Automata in Continuous Time. In *25th Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 342–351.

[9] Kousha Etessami, Marta Z. Kwiatkowska, Moshe Y. Vardi, and Mihalis Yannakakis. 2008. Multi-Objective Model Checking of Markov Decision Processes. *Logical Methods in Computer Science* 4, 4 (2008).

[10] Mariken Everdij and Henk Blom. 2008. Enhancing hybrid state Petri nets with the analysis power of stochastic hybrid processes. In *9th International Workshop on Discrete Event Systems, WODES 2008*. IEEE, 400–405.

[11] Martin Fränzle, E. Moritz Hahn, Holger Hermanns, Nicolás Wolovick, and Lijun Zhang. 2011. Measurability and Safety Verification for Stochastic Hybrid Systems. In *14th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2011*. ACM, New York, NY, 43–52.

[12] Hamed Ghasemieh, Boudewijn Haverkort, Marijn Jongerden, and Anne Remke. 2015. Energy Resilience Modelling for Smart Houses. In *45th IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2015*. IEEE, 275–286.

[13] Hamed Ghasemieh, Anne Remke, and Boudewijn Haverkort. 2016. Survivability analysis of a sewage treatment facility using hybrid Petri nets. *Performance Evaluation* 97 (2016), 36–56.

[14] Sergio Giro, Pedro R. D'Argenio, and Luis M. F. Fioriti. 2014. Distributed probabilistic input/output automata: Expressiveness, (un)decidability and algorithms. *Theoretical Computer Science* 538 (2014), 84–102.

[15] Marco Gribaudo and Anne Remke. 2016. Hybrid Petri nets with general one-shot transitions. *Performance Evaluation* 105 (2016), 22–50.

[16] E. Moritz Hahn, Arnd Hartmanns, Holger Hermanns, and Joost-Pieter Katoen. 2013. A Compositional Modelling and Analysis Framework for Stochastic Hybrid Systems. *Formal Methods in System Design* 43, 2 (2013), 191–232.

[17] Arnd Hartmanns, Holger Hermanns, and Jan Krcál. 2016. Schedulers are no Prophets. In *Semantics, Logics, and Calculi*, Christian W. Probst, Chris Hankin, and René R. Hansen (Eds.). LNCS, Vol. 9560. Springer, Cham, 214–235.

[18] Arie Hordijk and Lodewijk C. M. Kallenberg. 1984. Constrained Undiscounted Stochastic Dynamic Programming. *Mathematics of Operations Research* 9, 2 (1984), 276–289.

[19] Graham Horton, Vidyadhar G. Kulkarni, David M. Nicol, and Kishor S. Trivedi. 1998. Fluid stochastic Petri nets: Theory, applications, and solution techniques. *European Journal of Operational Research* 105, 1 (1998), 184–201.

[20] Jannik Hüls, Carina Pilch, Patricia Schinke, Joanna Delicaris, and Anne Remke. 2019. State-Space Construction of Hybrid Petri Nets with Multiple Stochastic Firings. In *16th International Conference on Quantitative Evaluation of Systems, QEST 2019 (LNCS)*, Vol. 11785. Springer, Cham, 182–199.

[21] Jannik Hüls and Anne Remke. 2019. Model Checking HPnGs in Multiple Dimensions: Representing State Sets as Convex Polytopes. In *19th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Objects, Components, and Systems, FORTE 2019 (LNCS)*, Vol. 11535. Springer, Cham, 148–166.

[22] Kurt Jensen and Lars M. Kristensen. 2009. *Coloured Petri nets: Modelling and Validation of Concurrent Systems*. Springer, Berlin, Heidelberg.

[23] Marijn Jongerden, Jannik Hüls, Anne Remke, and Boudewijn Haverkort. 2016. Does Your Domestic Photovoltaic Energy System Survive Grid Outages? *Energies* 9 (2016), 736–744.

[24] G. Peter Lepage. 1978. A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics* 27, 2 (1978), 192–203.

[25] Omid Madani, Steve Hanks, and Anne Condon. 1999. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *16th National Conference on Artificial Intelligence and 11th Conference on Innovative Applications of Artificial Intelligence, AAAI '99/IAAI '99*. AAAI, Orlando, FL, 541–548.

[26] Carl A. Petri. 1962. *Kommunikation mit Automaten*. PhD Thesis. Universität Hamburg, Germany.

[27] 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, 475–486.