Model Checking HPnGs in Multiple Dimensions: Representing State Sets as Convex Polytopes

Jannik Hüls, Anne Remke

Model checking HPnGs in multiple dimensions: representing state sets as convex polytopes

Jannik Hüls[1] and Anne Remke[1]

Westfälische Wilhelms-Universität, Münster, Germany
{jannik.huels; anne.remke}@uni-muenster.de

Abstract. Hybrid Petri Nets with general transitions (HPnG) include general transitions that fire after a randomly distributed amount of time. Stochastic Time Logic (STL) expresses properties that can be model checked using a symbolic representation for sets of states as convex polytopes. Model checking then performs geometric operations on convex polytopes. The implementation of previous approaches was restricted to two stochastic firings. This paper instead proposes model checking algorithms for HPnGs with an arbitrary but finite number of stochastic firings and features an implementation based on the library HyPro.

1 Introduction

Hybrid systems combine continuous and discrete behavior and are used to model and verify safety-critical systems. Different approaches exist for the reachability analysis of Hybrid automata, e.g., flowpipe construction for different state-space representations [23, 21, 12]. Hybrid Petri nets form a subclass of Hybrid automata [2] and have further been extended to *Hybrid Petri nets with general transitions* (*HPnGs*) in [14], that fire stochastically after a random delay. They form a subclass of stochastic hybrid systems with piece-wise linear continuous behaviour without resets and a probabilistic resolution of discrete non-determinism. Albeit these restrictions, they have been applied successfully to critical infrastructures, like water and power distribution [8, 18]. Several approaches for Hybrid automata extended with discrete probability distributions exist [31, 20, 28, 29]. More general stochastic Hybrid systems often require a higher level of abstraction [19, 1]. Related Petri net approaches are also restricted e.g., w.r.t. the number of continuous variables [15] or to Markovian jumps [5].

Stochastic Time Logic (STL) closely resembles MITL [3] or the *temporal layer* of STL/PSL [22] and is used to specify properties of HPnGs. Their piece-wise linear evolution of continuous variables allows to partition the state space into convex polytopes (so-called regions) with similar characteristics [9]. The idea of a polyhedra based representation of the state space has been explored before for model checking HPnGs [11], for (flowpipe) approximations [6, 7] and to abstract uncountable-state stochastic processes [27, 26]. Our approach explicitly includes the stochastic behaviour over time into the state representation; every stochastic firing adds a dimension to the state space. Model checking then identifies all realizations of the random variables, which satisfy a given STL formula. The satisfaction set of (the conjunction of) atomic properties is a single convex polytope, and negation requires a translation into a convex representation. A previous approach using Nef polyhedra was restricted to models with two stochastic firings [13] due to the restricted implementation of hyperplane arrangement in the corresponding CGAL library. Recently, we proposed the translation of the nodes of the *parametric location tree* (PLT) [17] into a geometric and symbolic system representation. This construction allows to circumvent the problem of hyperplane arrangement while still providing a geometric state set representation. Given the PLT of an HPnG model, this paper presents model checking for STL properties in HPnGs with an *arbitrary but finite number of stochastic firings*. For each STL operator an algorithm is introduced, based on geometric operations on symbolic state set representations. Our implementation relies on the library HyPro [25], which offers efficient implementations for operations on convex polytopes [32] in higher dimensions. Being aware of other implementations [4],[30], we like HyPro's convenient interfaces and conversion functions.

Model checking recursively follows the parse tree of the formula. Per region a convex representation of its satisfying parts is returned. A simple but scalable example is used to showcase the feasibility of the approach. Note that the resulting satisfaction sets implicitly contain the stochastic evolution and allow to compute the probability that a HPnG satisfies a specific STL by integrating over the density of each random variable.

Organisation: Section 2 discusses the modeling formalism, Section 3 illustrates the state-space generation using HyPro. Section 4 describes the logic STL, for which Section 5 introduces the region-based model checking approach.

2 Hybrid Petri nets with general transitions

HPnGs are defined according to [14] with the extension to *multiple* stochastic firings that fire after a randomly distributed amount of time as in [10].

Their key components are: Discrete or continuous *Places* which contain a number of tokens or an amount of fluid. A *marking* $\mathbf{M} = (\mathbf{m}, \mathbf{x})$ combines the discrete marking $\mathbf{m} = (m_1, ..., m_{n_d})$, and the continuous marking $\mathbf{x} = (x_1, ..., x_{n_c})$, for n_d discrete and n_c continuous places. The number of tokens in the i-th discrete place is denoted m_i and x_i contains the value of the i-th continuous place.

Transitions change the content of places upon firing. Discrete transitions (general, deterministic and immediate) change the number of tokens in discrete places. Transitions may only fire if all enabling criteria are met. Deterministic transitions fire after being enabled for a constant predefined amount time. Immediate transitions fire after zero time. The random firing delay of a general transition is distributed according to an arbitrary continuous probability distribution. Continuous transitions change the fluid level of connected input and output places with a constant nominal rate. [14]. *Arcs* connect places and transitions and define via weights and priorities how their content changes, when a transition fires. Guard arcs enable transitions based on the discrete or continuous marking of connected places.

Definition 1. *An HPnG is defined as a tuple $(\mathcal{P}, \mathcal{T}, \mathcal{A}, \mathbf{M}_0, \Phi)$. $\mathcal{P}$ is the set of places, $\mathcal{T}$ the set of transitions and $\mathcal{A}$ the set of arcs. The initial marking is denoted as $\mathbf{M}_0$ and the tuple of mappings Φ, further defines the model evolution. The finite set $\mathcal{P} = \mathcal{P}^d \cup \mathcal{P}^c$ combines discrete and continuous places. The finite set of transitions $\mathcal{T} = \mathcal{T}^I \cup \mathcal{T}^D \cup \mathcal{T}^G \cup \mathcal{T}^F$ holds immediate, deterministic, general and continuous transitions, $\mathcal{T} \setminus \mathcal{T}^F$ holds the set of discrete transitions. The set $\mathcal{A}$ is divided into three subsets: (i) The set of discrete arcs $\mathcal{A}^d \subseteq ((\mathcal{P}^d \times \mathcal{T}^D) \cup (\mathcal{T}^D \times \mathcal{P}^d))$ connects discrete places and transitions. (ii) The set of continuous arcs $\mathcal{A}^f \subseteq ((\mathcal{P}^c \times \mathcal{T}^F) \cup (\mathcal{T}^F \times \mathcal{P}^c))$ connects continuous places and transitions. (iii) The set of guard arcs $\mathcal{A}^t \subseteq (((\mathcal{T}^D \cup \mathcal{T}^I \cup \mathcal{T}^G) \times (\mathcal{P}^d \cup \mathcal{P}^c)) \cup (\mathcal{T}^C \times \mathcal{P}^d))$ connects discrete and continuous places to all kind of transitions. The initial marking $\mathbf{M}_0 = (\mathbf{m}_0, \mathbf{x}_0)$ denotes the initial number of tokens and fluid levels in the places. Parameter functions defining the specifics of enabling and model evolution are collected in:*

$$\Phi = (\Phi_b^{\mathcal{P}}, \Phi_p^{\mathcal{T}}, \Phi_d^{\mathcal{T}}, \Phi_{st}^{\mathcal{T}}, \Phi_g^{\mathcal{T}}, \Phi_n^{\mathcal{A}}, \Phi_u^{\mathcal{A}}, \Phi_s^{\mathcal{A}})$$

Example 1. Figure 1 shows a HPnG, which models a buffer (`B`) as continuous place (double circle) with a varying number of continuous input transitions and one continuous output transition (double rectangle). The buffer has a max. capacity of `C` = 100, starts with $L = 10$ and can be filled using producer pumps $\mathtt{I_1}...\mathtt{I_n}$ and is drained by one demand pump D, each with nominal rate $\mathtt{r_i} \cup \mathtt{r_d} = 5$. The general transitions (rectangle) $\mathtt{G_1}, ..., \mathtt{G_n}, \mathtt{G_d}$ disable the pumps connected via guard arcs (two arrowheads) with weight 1 to the discrete places (circle).

Enabling rules Every continuous place has an upper boundary defined in $\Phi_b^{\mathcal{P}} : \mathcal{P}^c \to \mathbb{R}^+ \cup \infty$. (The lower boundary is always zero.) $\Phi_{st}^{\mathcal{T}} : \mathcal{T}^C \to \mathbb{R}^+$ defines the constant nominal flow rate for each continuous transition. $\Phi_g^{\mathcal{T}} : \mathcal{T}^G \to CDF$ assigns a unique cumulative distribution function (CDF) to each general transition, which does not depend on the number of firings. $\Phi_p^{\mathcal{T}} : \mathcal{T}^D \cup \mathcal{T}^C \cup \mathcal{T}^G \to \mathbb{N}_{>0}$ defines a priority for each type of transition. Using $\Phi_u^{\mathcal{A}} : \mathcal{A}^t \to \{\lhd, \mathbb{R}\}$ with $\lhd = \{\geq, <\}$ assigns a comparison operator and a weight to each guard arc. $\Phi_n^{\mathcal{A}} : \mathcal{A}^d \to \mathbb{R}^+$ determines the number of tokens moved when the transition fires and $\Phi_s^{\mathcal{A}} : \mathcal{A}^C \to \mathbb{R}^+$ defines a share for conflicting continuous transitions.

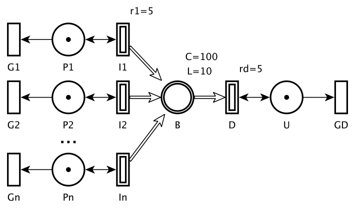

Fig. 1. Scalable HPnG model.

For transition T we define the set of input places $\mathcal{I}_{\mathcal{P}}(T)$, the set of output places $\mathcal{O}_{\mathcal{P}}(T)$ and the set of places connected via guard arcs $\mathcal{G}_{\mathcal{P}}(T)$. Let $\mathcal{P}_i^d \in P^d$ denote the i-th discrete place and $\mathcal{P}_i^c \in P^c$ the i-th continuous place, respectively. A discrete transition $T_j \in \mathcal{T} \setminus \mathcal{T}^{\mathcal{C}}$ is enabled if the following conditions hold: (i) Discrete guard arcs satisfy: $\forall P_i^d \in \mathcal{P}^d \cap \mathcal{G}_{\mathcal{P}}(T_j), (\lhd, q) = \Phi_u^A(\langle T_j, P_i^d \rangle) : m_i \lhd q$, (ii) Continuous guard arcs satisfy: $\forall P_i^c \in \mathcal{P}^c \cap \mathcal{G}_{\mathcal{P}}(T_j), (\lhd, q) = \Phi_u^A(\langle T_j, P^c \rangle) : x_i \lhd q$, (iii) Connected input places satisfy: $\forall P_i^d \in \mathcal{I}_{\mathcal{P}}(T_j) : m_i \geq \Phi_n^A(\langle P_i^d, T_j \rangle)$.

Continuous transitions may only be connected to discrete places via guard arcs. The following needs to hold for a continuous transition $T_j^F \in \mathcal{T}^F$ to be enabled: (i) Discrete guard arcs satisfy: $\forall P_i^d \in \mathcal{P}^d \cap \mathcal{G}_{\mathcal{P}}(T_j^F), (\lhd, q) = \Phi_u^A(\langle T_j^F, P_i^d \rangle) : m_i \lhd q$, and (ii) connected input places hold fluid: $\forall P_i^c \in \mathcal{I}_{\mathcal{P}}(T_j^F) : x_i > 0$.

Model evolution Discrete transitions are associated with clocks. Let c_j be the clock associated with transition $T_j \in \mathcal{T} \setminus \mathcal{T}^F$. If transition T_j is enabled c_j evolves with $\delta c_j / \delta t = 1$, otherwise $\delta c_j / \delta t = 0$. A deterministic transition fires when c_j reaches the transitions firing time defined by $\Phi_d^{\mathcal{T}} : \mathcal{T} \setminus \mathcal{T}^F \to \mathbb{R}^+$. A firing of a discrete transition changes the corresponding marking according to the weights specified in $\Phi_n^{\mathcal{A}} : \mathcal{A} \to \mathcal{R}^+$. A general transition may fire at any point in time, if enabled, and changes the discrete marking similarly to the discrete transitions. The probability that a general transition fires at the scheduled firing time of a discrete transition is zero, hence, model evolution needs to consider all enabled general transitions firing before or after the next scheduled deterministic event.

For a continuous transition $P_i^c \in \mathcal{P}^c$ we define the set of input transitions $\mathcal{I}_{\mathcal{T}}(P_i^c)$, the set of output places $\mathcal{O}_{\mathcal{T}}(P_i^c)$. An enabled continuous transition fires with its nominal rate. If a continuous place is at either boundary *rate adaptation* is performed to connected continuous transitions. This decreases the inflow to match the outflow if the place is full (or the other way around). Let $r(T_j^F)$ be the actual rate of the continuous transition $T_j^F \in \mathcal{T}^F$ after adaptation. The in-flow $f_{in}(P_i^c)$ of $P_i^c \in \mathcal{P}^c$ is defined as $f_{in}(P_i^c) = \sum_{T_j^F \in \mathcal{I}_{\mathcal{T}}(P_i^c)} r(T_j^F)$, i.e., the sum of all incoming rates. The out-flow is the defined as $f_{out}(P_i^c) = \sum_{T_j^F \in \mathcal{O}_{\mathcal{T}}(P_i^c)} r(T_j^F)$. A continuous place evolves with *drift* $d(P_i^c) = f_{in}(P_i^c) - f_{out}(P_i^c)$.

Example 2. The enabling of the continuous transitions in Figure 1 depends on the marking of the discrete places. Initially, each discrete place contains one token, which satisfies the condition of the guard arc connecting them to `Ii` and `D`, respectively, hence enabling them. Each general transition `Gi` is initially enabled and when firing after a random delay, it disables input pump `Ii`. The initial marking of place `B` is 10, and its drift depends on the number of enabled producer pumps (and the enabling of `D`): $d = \sum_{1 \le i \le n} r_i - r_d$. Since all r_i and r_d have the same rate 5, the initial drift is $d = 5(n-1)$.

3 State space representation using HyPro

The state of an HPnG contains all information required by the time-bounded analysis, as well as model checking an HPnG. It is defined as $\Gamma = (\mathbf{m}, \mathbf{x}, \mathbf{c}, \mathbf{d}, \mathbf{g})$, where $\mathbf{m}$ and $\mathbf{x}$ are the discrete and continuous marking, respectively, and $\mathbf{c} = (c_1, ..., c_{|T^D|})$ is the vector of discrete clocks, $\mathbf{d} = (d_1, ... d_{|T^C|})$ contains the drift of each continuous place. Furthermore, $\mathbf{g} = (g_1, ... g_{|T^G|})$ indicates the time each general transition has been enabled. The state space $S = \{\Gamma = (\mathbf{m}, \mathbf{x}, \mathbf{c}, \mathbf{d}, \mathbf{g})\}$ contains all reachable HPnG states w.r.t. the initial state $\Gamma_0 = (\mathbf{m}_0, \mathbf{x}_0, \mathbf{0}, \mathbf{d}_0, \mathbf{0})$. The continuous marking $\mathbf{x}$ changes with derivative $\mathbf{d}$. The discrete clocks $\mathbf{c}$

and the enabling time of general transitions $\mathbf{g}$ change with derivative 1 for all enabled transitions. The discrete marking $\mathbf{m}$ and the drift of the continuous places $\mathbf{d}$ change with *events*: (i) A continuous place reaching its lower or upper boundary. (ii) A continuous place reaches the weight of a connected guard arc. (iii) An enabled discrete transition fires. Events do not move time forward.

Although the state of the system Γ changes continuously with time, its bounded evolution up to some maximum time $\tau_{\max}$ can be described symbolically using a *parametric location tree* (PLT) [14] [17]. Nodes are so-called parametric locations and symbolically represent all states, whose continuous marking only differ due to the evolution of time. The occurrence of events results in branching to new locations. A location is defined as a tuple $\Lambda = (t_e, \Gamma, \mathbf{S}, p)$. At time t_e the system enters the parametric location and the state Γ follows the Definition presented before. The potential domain $\mathbf{S}$ provides the bounds for each general transition firing. The real number p is a probability assigned to each location in case of a conflict. The number of random variables present in the system n corresponds to the number of stochastic firings that occurred plus the number of general transitions that are currently enabled but have not fired before $\tau_{\max}$. All random variables are collected in the vector $\mathbf{s} = (s_0, ..., s_n)$ and the domain $\mathbf{S} = ([l_0, r_0], ..., [l_n, r_n])$ contains all possible values for the random variables per parametric location. We define $\mathbf{s} \in \mathbf{S}$ iff $s_i \in [l_i, r_i]$ for all $0 \leq i \leq n$.

The PLT is generated using a depth-first search by extending all parametric locations until $\tau_{\max}$. We start from the initial parametric location, which extends the initial state Γ_0 by $t_e = 0$, $p = 1$ and $\mathbf{S}_0 = ([0, \tau_{\max}], \ldots, [0, \tau_{\max}])$. In each location the time until the next event is computed relatively to the entry time t_e of that location. Note that the number of events that occur at the *next minimum event time* $\tau_{\min}$ is finite [14], and for each possible event e, a new parametric location is created with a marking adapted according to the causing event. Additionally, each enabled general transition may fire before that point in time. Hence, additional successors are scheduled for each enabled general transition and the potential domains have to be set accordingly. The next minimum event time is unique before the first stochastic firing. After a single stochastic firing s_i, the entry time of a location, the clocks, the continuous marking and the potential domains may linearly depend on the value of the corresponding random variable s_i. The case of multiple general transition firings leads to multi-dimensional linear equations and the domains $\mathbf{S}$ may linearly depend on the vector of random variables $\mathbf{s}$. For each successor, the procedure is called recursively and the domains are adapted to ensure the order of events. The intervals denote the values of $\mathbf{s}$, for which the causing event is the minimum next event.

Geometric representation of locations We propose model checking algorithms for HPnGs that combine the tree-based approach of parametric locations with the geometric representation of stochastic time diagrams (STD). The implementation of the presented algorithm allows the analysis of HPnGs with multiple general transition firings. For each parametric location, we construct a $n+1$-dimensional geometric representation, one dimension for each random variable and one for time. This corresponds to a region in an STD, as defined in [9].

For a given time t and a valuation of vector $\mathbf{s}$, $\Gamma(\mathbf{s},t)$ defines a specific system state. For all system states in a region, the initial marking $\Gamma(\mathbf{s},t).\mathbf{m}$ and the drift $\Gamma(\mathbf{s},t).\mathbf{d}$ do not change. As shown in [9], the amount of fluid and the clock valuations are linear equations of $\mathbf{s}$ and t.

Definition 2. *A region R is a maximal connected set of $(\mathbf{s},t)$ points, for which:*

$$\forall(\mathbf{s}_1,t_1),(\mathbf{s}_2,t_2)\in R \begin{cases}\Gamma(\mathbf{s}_1,t_1).\mathbf{m}=\Gamma(\mathbf{s}_2,t_2).\mathbf{m},\\ \Gamma(\mathbf{s}_1,t_1).\mathbf{d}=\Gamma(\mathbf{s}_2,t_2).\mathbf{d}.\end{cases}$$

The boundaries between regions, which represent the occurrence of an event, are also characterized by linear functions of $\mathbf{s}$ and t and represent a multi dimensional *hyperplane*. We denote the hyperplane between regions R and R' that corresponds to event e as $H^e_{R,R'}$. Using halfspace intersection, convex polytopes are created as geometric representation of regions [9].

Time evolution Starting from a tuple $(\mathbf{s},t)$ the time evolution is deterministic within a region, such that a time step τ is defined through the forward time closure as $\mathcal{T}^+_R(\mathbf{s},t)=\{(\mathbf{s},t')\mid(\mathbf{s},t')\in R\wedge t'\geq t\}$. The occurrence of an event e does not advance time, but may lead to branching between locations, e.g. in case multiple events are scheduled at the same time. Hence, a discrete step caused by event e to other regions R' is defined for all tuples that lie on the hyperplane $H^e_{R,R'}$, for $R\neq R'$. The discrete successors of $(\mathbf{s},t)$ are then defined as $\mathcal{D}^+(\mathbf{s},t)=\{R'\mid\exists e.(\mathbf{s},t)\in H^e_{R,R'}\}$. For a fixed valuation $\mathbf{s}$, a finite path σ, starting at time t_0 is denoted as $\sigma(\mathbf{s},t_0)$ and defined as alternating sequence $((\mathbf{s},t_0)\in R_0)\xrightarrow{\tau_1}((\mathbf{s},t_1)\in R_0)\xrightarrow{e_1}((\mathbf{s},t_1)\in R_1)\xrightarrow{\tau_2}\ldots\xrightarrow{\tau_n}((\mathbf{s},t_n)\in R_{n-1})\xrightarrow{e_n}((\mathbf{s},t_n)\in R_n)$, such that $(\mathbf{s},t_i)\in\mathcal{T}^+_{R_{i-1}}(\mathbf{s},t_{i-1})$ and $R_i\in\mathcal{D}^+(\mathbf{s},t_i)$ and $t_i=t_0+\sum_{j\geq1}^{i}\tau_j$ for all $0\leq i\leq n$. A state is on path σ if it is in the forward time closure of a region in step i of σ:

$$(\mathbf{s},t)\in\sigma \text{ iff } \exists R.\exists i.(\mathbf{s},t)\in\mathcal{T}^+_R(\mathbf{s},t_i). \tag{1}$$

For a definition of the resulting probability space, we refer to [24]. Note that Zeno-behaviour is excluded by prohibiting cycles which potentially take no time, i.e. cycles of only immediate and general transitions. This together with the restriction to time-bounded reachability analysis ensures that a path is always finite. The exclusion of cycles of general transition firings is mostly technical, as the probability of infinitely many firings in finite time is zero with continuous distributions.

The use of HyPro The algorithm presented in [17] to transform locations into the graphical representation of regions heavily relies on the `C++` library HyPro [25]. Amongst other data structures, HyPro contains an implementation for convex polytopes [32] as well as a wrapper class to the well-known Parma polyhedra library (PPL) [4]. We use the so-called $\mathcal{H}$-representation for convex polytopes, where H is defined as the intersection of a finite set of halfspaces. Note that previous implementations [10, 13] used Nef-polyhedra by CGAL, but were limited to three dimensions. While Nef-polyhedra are closed under the set operations union, intersection and set difference, they are not necessarily convex. Convex polytopes are only closed under intersection. Section 5 will present model checking algorithms that only deal with convex state set representations.

Note that HyPro is restricted to closed convex polytopes, that also need to be bounded. This leads to difficulties when performing operations that are not closed w.r.t. this representation, as e.g. negation. The restriction to bounded polytopes however naturally fits our analysis and model checking approach, as the state space is bounded by the maximum time of analysis.

Example 3. The running example focuses on the core complexity of the HPnG formalism, i.e. the number of general transition firings. As shown in [8, 18], analyzing larger models with few random variables is not prohibitive. The random variables modeling pump failures compete and hence increase model complexity. In general, the resulting state space has $n + 2$ dimensions.

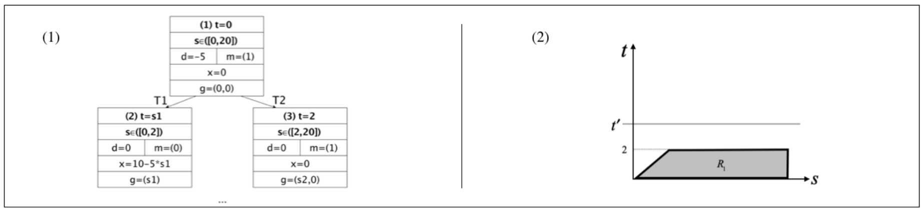

Fig. 2. A PLT and the region R_1 of the root location.

Figure 2 shows the root location and the first level of child locations of the PLT for $n = 0$. PLTs for all settings are available online[1]. The root location has the entry time $t = 0$ and neither the potential domain is restricted nor any clock has evolved. Since no input pump is present in this system, only the demand pump is enabled, initially. The initial drift is $d = -5$ and given the initial level $L = 10$ of place B, it takes 2 time units until B is empty. Hence, two events may occur, i.e. either the demand is disabled first or B is empty. The entry times of the child locations thus are $t = s_1$ and $t = 2$, respectively. Their potential domains are restricted s.t. the locations are only valid for $s \in [0, 2]$, if the general transition fires before the place is empty, and $s \in [2, \tau_{max}]$ otherwise.

The geometric representation of the root location is shown in the right part of Figure 2. The region R_1 is created by halfspace intersection: Every region is first restricted by the entry time of the location and τ_{max} and the potential domain of the random variables. Hence for the root location the halfspaces defined by $t \geq 0$ and $t \leq \tau_{max}$ as well as $s \geq 0$ and $s \geq \tau_{max}$ are intersected. Intersecting the halfspaces defined by the entry time of the respective locations, i.e. $t \geq 0$, $t \leq s_1$ and $t \leq 2$, then creates R_1.

[1] https://uni-muenster.sciebo.de/s/A3mNHLclM8233T5

4 Stochastic Time Logic

A logic for expressing properties of interest for HPnGs at a certain time was introduced in [11] and denoted as Stochastic Time Logic (STL). This paper concentrates on computing those subsets of the domain $\mathbf{S}$ for which Φ holds at a given point in time. An STL formula Φ is built according to Equation 2:

$$\Phi ::= \ x_P \geq c \mid m_P = a \mid \ \neg\Phi \ \mid \Phi \wedge \Phi \ \mid \Phi\mathcal{U}^{[0,T]}\Phi, \tag{2}$$

where $x_P \geq c$ and $m_P = a$ are continuous and discrete atomic properties and $T \in \mathbb{R}^+$ a time bound. The satisfaction relation from [11] is adapted, to cover branching between locations:

$$\begin{aligned}
&\Gamma(\mathbf{s},t) \models x_P \geq c && \text{iff } \Gamma(\mathbf{s},t).x_P \geq c, && (3)\\
&\Gamma(\mathbf{s},t) \models m_P = a && \text{iff } \Gamma(\mathbf{s},t).m_P = a, && (4)\\
&\Gamma(\mathbf{s},t) \models \neg\Phi && \text{iff } \Gamma(\mathbf{s},t) \not\models \Phi, && (5)\\
&\Gamma(\mathbf{s},t) \models \Phi_1 \wedge \Phi_2 && \text{iff } \Gamma(\mathbf{s},t) \models \Phi_1 \wedge \Gamma(\mathbf{s},t) \models \Phi_2, && (6)\\
&\Gamma(\mathbf{s},t) \models \Phi_1\mathcal{U}^{[0,T]}\Phi_2 && \text{iff } \exists\sigma(\mathbf{s},t).\exists\tau \in [t,t+T].\Gamma(\mathbf{s},\tau) \models \Phi_2 \wedge (\mathbf{s},\tau) \in \sigma(\mathbf{s},t) && \\
& && \quad \wedge (\forall\tau' \in [t,\tau].\Gamma(\mathbf{s},\tau') \models \Phi_1 \wedge (\mathbf{s},\tau') \in \sigma(\mathbf{s},t)). && (7)
\end{aligned}$$

The *until* operator holds if a path σ starting in $(\mathbf{s},t)$ exists, such that a point in time $\tau \geq t$ exists, for which Φ_2 holds and that for all time points in $\tau' \in [t,\tau]$ the formula Φ_1 holds and all corresponding states $(\mathbf{s},\tau')$ lie on σ, according to Equation 1. We define the satisfaction set for time t', denoted $Sat^{t'}$ and the satisfaction set for a region Sat^R as follows:

$$Sat^{t'}(\Phi) = \{\mathbf{s} \in \mathbf{S} \mid \Gamma(\mathbf{s},t') \models \Phi\} \text{ and } Sat^R(\Phi) = \{(\mathbf{s},t) \in R \mid \Gamma(\mathbf{s},t) \models \Phi\}. \tag{8}$$

The former satisfaction set contains all possible stochastic firing times, such that their time evolution from t' on satisfies a given STL formula Φ. These subsets of the domain of all random variables present in the system are also called *validity intervals*. The latter satisfaction set contains all points $(\mathbf{s},t) \in R$, such that $\Gamma(\mathbf{s},t)$ satisfies Φ. Note that Φ can also be wrapped into a probability operator $\varphi ::= P_{\bowtie p}(\Phi)$, where $\bowtie \ \in \{<,>,\leq,\geq\}$ is a comparison operator and $p \in [0,1]$ a probability bound. This expresses that for a given point in time, the probability that a formula Φ holds matches the threshold p. This probability can be computed from the resulting satisfaction sets, which implicitly includes information about the stochastic behaviour. This computation is however not covered in this paper. We also exclude the nesting of multiple until operators.

Example 4. All states with a disabled consumer pump are identified by $\Phi_1 := (m_U = 0)$ and $\Phi_2 := \neg(x_B \leq 2)$ ensures that the buffer does not have less than two units of fluid. Checking whether the buffer is emptied within 4 time units, while the output pump stays on, is formulated as $\Phi_3 := (m_U = 1)U^{[0,4]}(x_B \leq 0)$.

5 Model Checking STL

This section presents STL model checking algorithms for HPnGs. To obtain $Sat^{t'}(\Phi)$, first all regions the model can be in at time t' are identified. Geometrically, these regions all have a non-empty intersection with hyperplane $H_{t'}$. Then the general model checking function is called per candidate region for the overall STL formula and returns a satisfaction set $Sat^{R}(\Phi)$ per region R. Following the recursive definition of STL formulas, operator-specific algorithms are called to compute satisfaction sets along the parse tree of the STL formula. Model checking discrete and continuous atomic formulae as well as their conjunction solely relies on the intersection of regions with halfspaces. However, model checking negation and the time-bounded until operator requires the set operations complement, set difference and union, which may result in non-convex polytopes. We use sets of convex polytopes instead of performing the operation union to ensure that the model checking algorithms are closed w.r.t. the state representation.

5.1 Model checking algorithms per operator

Model checking STL is performed along the parse tree of the formula for all regions in which the system can be at time t'. Negation and conjunction are independent of time t', and return the set of convex polytopes $Sat^{R}(\Phi)$. Because of the relative definition of the time bound $[0,T]$, the algorithm for until is executed only for t', and returns the satisfaction set w.r.t. time t' and region R:

$$Sat^{t',R}(\Phi_1 U^{[0,T]}\Phi_2) = \{\mathbf{s} \in \mathbf{S} \mid (\mathbf{s},t') \in R \wedge \Gamma(\mathbf{s},t') \vDash \Phi_1 U^{[0,T]}\Phi_2\}. \tag{9}$$

Note that the interplay between the different kind of satisfaction sets is explained in Section 5.2. Recall that polytopes are represented as the intersection of a finite number of halfspaces. We create a polyope representation P by intersecting m_P halfspaces:

$$P = \bigcap_{i=1}^{m_P} h_{i,P}, \text{ where } h_{i,P} = \{x \in \mathbb{R}^d | c_i^T \cdot x \leq d_i\}. \tag{10}$$

Restricting polytopes to their intersection with R allows defining satisfaction sets per region as the finite union of n_Φ non-necessarily disjoint polytopes P_i^R:

$$P^R = P \cap R, \text{ and } Sat^R(\Phi) = \bigcup_{j=1}^{n_\Phi} P_j^R = \{(\mathbf{s},t) \in R \mid \Gamma(\mathbf{s},t) \vDash \Phi\}. \tag{11}$$

Atomic formula Model checking discrete and continuous atomic formula is shown in [17] and [11]. For completeness, we present the algorithm per region, as shown in Listing 1. It takes as input a specific region R and a discrete or continuous atomic property Φ and outputs a satisfaction set, which contains all states in R that satisfy Φ. For a continuous atomic formula Φ, the region has to be intersected with the halfspace representing the continuous level $h_{x_P \geq c}$. (Line 3) A discrete formula is satisfied in the entire region or not at all, hence, a test whether the considered regions meets the marking specified by the formula Φ is sufficient. (Line 4)

Theorem 1. *The satisfaction set $Sat^R(\Phi)$ w.r.t. a region R is empty or a convex polytope in case Φ is an atomic property.*

Proof. Let $\Phi := m_p = a$. A discrete atomic property is satisfied in the whole region or not at all. In either case, $Sat^R(\Phi)$ is a convex polytope. Let $\Phi := x_p \geq c$. A continuous atomic formula may only be satisfied in part of the region. The boundary c implies a halfspace $h_{x_P \geq c}$ which after intersection with R again results in a convex polytope as convex polytopes are closed under intersection.

Negation According to the semantics of STL and as implemented in [13], negation is defined as set difference. Convex polytopes are not closed under set difference, hence, the satisfaction set of $\Phi = \neg\Phi_1$ is in general not a convex polytope. We obtain a representation in terms of sets of convex polytopes, as follows. The complement of a convex polytope with respect to a region $\mathbf{R}$ is a not necessarily convex polytope P_C and can be computed as the union of the inverted halfspaces which define P. However, inverting halfspaces results in turning a non-strict comparison operator in the halfspace definition into a strict comparison. This results in an open polytope, which HyPro currently does not support. Hence, we define a non-disjunct complement w.r.t. region R:

$$P^R_{\tilde{C}} = (\bigcup_{i=1}^{m_P} h^{\sim}_{i,P}) \cap R \text{ and } h^{\sim}_{i,P} = \begin{cases} \{\mathbf{x} \in \mathbb{R}^d | c_i^T \cdot X \geq d_i\} \text{ iff } h_{i,P} = \{\mathbf{x} \in \mathbb{R}^d | c_i^T \cdot x \leq d_i\}, \\ \{\mathbf{x} \in \mathbb{R}^d | c_i^T \cdot X \leq d_i\} \text{ iff } h_{i,P} = \{\mathbf{x} \in \mathbb{R}^d | c_i^T \cdot x \geq d_i\}. \end{cases} \tag{12}$$

such that $P^R_{\tilde{C}} \cap P^R \neq \emptyset$ and results exactly in the facets of P^R. Note that this definition also results in non-disjunct satisfaction sets and imprecise borders of the validity intervals that are computed after model checking each region. This is currently circumvented by additionally storing in a separate vector whether a halfspace is open or closed. In case the satisfaction set already consists of more than one polytope, the negation of the respective formula requires building the complement over a set of polytopes.

Listing 2 illustrates the general algorithm for the negation of an STL formula with respect to region R. After instantiating the satisfaction set for $\neg\Phi$ (Line 1), the general model checking routine is called for Φ (Line 2). The resulting satisfaction set consists of a set of convex polytopes, each created by the intersection of halfspaces. Hence, when computing the satisfaction set of $\neg\Phi$, for each polytope all creating halfspaces need to be inverted (Line 5–6) and collected.

Listing 1 Satisfaction set $Sat^R(\Phi)$ for atomic formula Φ and a region R.

```
1: $Sat^R(\Phi) \leftarrow \emptyset$
2: **if** isContinuous($\Phi$) **then**
3:     $Sat^R(\Phi) \leftarrow R \cap h_{x_p \geq c}$
4: **if** isDiscrete($\Phi$) $\wedge$ $\Gamma.m = \Phi.m$ **then**
5:     $Sat^R(\Phi) \leftarrow R$
6: **return** $Sat^R(\Phi)$
```

Listing 2 Satisfaction set $Sat^R(\Phi)$ of a negated formula $Sat(\neg\Phi)$ for region R.

```
 1: $Sat^R(\Phi) \leftarrow$ modelcheck($R$, $\Phi$)
 2: for all $P_j \in Sat^R(\Phi)$ do
 3:     $H^{\sim}_{P_J} \leftarrow \varnothing$                    ▷ Set of convex polytopes that fulfill $\neg\Phi$ in $R$.
 4:     for all $h_{i,P_j} \in P_j$ do                  ▷ For all halfspaces defining $P$.
 5:         $H^{\sim}_{P_J} \leftarrow H^{\sim}_{P_J} \cup h_{i,P_j}$.invert()
 6: for all $x \in \times(H^{\sim}_{P_j})$ do   ▷ For all elements in the cross product over all sets $H^{\sim}_{P_j}$.
 7:     $P \leftarrow x_1 \cap R$
 8:     for $j \leftarrow 0; j > 1; j++$ do              ▷ For all halfspaces in the cross product.
 9:         $P \leftarrow P \cap x_j$
10:     $Sat^R(\Phi) \leftarrow Sat^R(\Phi) \cup P$
11: return $Sat^R(\Phi)$
```

Listing 3 Satisfaction set computation of $\Phi_1 \wedge \Phi_2$ for a region R.

```
1: $Sat^R(\Phi) \leftarrow \varnothing$
2: $Sat(\Phi_1) \leftarrow$ modelcheck($R$, $\Phi_1$)
3: $Sat(\Phi_2) \leftarrow$ modelcheck($R$, $\Phi_2$)
4: for all $P_i \in Sat(\Phi_1)$ do                         ▷ For all polytopes in $Sat(\Phi_1)$
5:     for all $P_j \in Sat(\Phi_2)$ do                     ▷ For all polytopes in $Sat(\Phi_2)$
6:         $Sat^R(\Phi) \leftarrow Sat^R(\Phi) \cup (P_i \cap P_j)$
7: return $Sat^R(\Phi)$
```

Then, the cross product over all these sets of inverted halfspaces per polytope is required (Line 7), whereas each entry in the resulting tuple indicates an inverted halfspace from a specific polytope. For each element of the cross product, a new polytope is constructed by successively intersecting the halfspaces that correspond to each entry of the tuple with the region (Lines 8–10), ensuring the intersection of all possible combinations of inverted halfspaces per polytope.

The resulting polytope, representing a part of the region satisfying $\neg\Phi$, is then added to the list of (not necessarily disjoint) convex polytopes, forming the satisfaction set of $\neg\Phi$ (Line 11), which then is returned (Line 12).

Conjunction In case of a conjunction $\Phi = \Phi_1 \wedge \Phi_2$ both satisfaction sets with respect to a specific region R, namely $Sat^R(\Phi_1)$ and $Sat^R(\Phi_2)$, are required and intersected to compute the satisfaction set $Sat(\Phi)$. Listing 3 illustrates the algorithm using the representation as sets of convex polytopes. First, the model checking algorithm is called recursively for Φ_1 and Φ_2. Then, each of the polytopes in $Sat(\Phi_1)$ is intersected with each of the polytopes in $Sat(\Phi_2)$ (Line 4–5) and the result (if non-empty) is added to the resulting satisfaction set. (Line 6)

Theorem 2. *The satisfaction set $Sat^R(\Phi)$ w.r.t. a region R, for an STL formula Φ that consists of negation and conjunction only, is a set of not necessarily disjunct convex polytopes.*

Proof. We prove the above Theorem by structural induction over the parse tree of the formula Φ, using the notation introduced throughout this section.
Inductive hypothesis: Suppose the theorem holds for arbitrary sub-formulas Φ_1 and Φ_2, which only consist of negation and conjunction.
Inductive case 1: For an atomic formula Φ follows directly from Theorem 1 that $Sat(\Phi)$ contains at most one convex polytope.
Inductive case 2: In the following we distinguish between a formula Φ, where the highest binding operator is a conjunction or a negation. Let $\Phi = \Phi_1 \wedge \Phi_2$ be a conjunction. Using the constructor case, it follows that both satisfaction sets $Sat(\Phi_1)$ and $Sat(\Phi_2)$ are sets of convex polytopes, which can be rewritten according to Equation 11. When intersecting those unions of polytopes, applying the distributive law yields again the union (of a union) of convex polytopes, as the intersection of two convex polytopes P_i^R and P_j^R will always be convex again.

$$Sat^R(\Phi_1 \cap \Phi_2) = Sat^R(\Phi_1) \cap Sat^R(\Phi_2) = \bigcup_{i=1}^{n_{\Phi_1}} P_i^R \cap \bigcup_{j=1}^{n_{\Phi_2}} P_j^R = \bigcup_{i=1}^{n_{\Phi_1}} \bigcup_{j=1}^{n_{\Phi_2}} (P_i^R \cap P_j^R). \tag{13}$$

Let $\Phi = \neg\Phi_1$ be a negation. According to Equation 11, let $Sat(\Phi_1) = Sat^R(\Phi_1) = \bigcup_{j=1}^{n_{\Phi_1}} P_j^R$ be a finite set of convex polytopes which all satisfy Φ_1 in R. Then, it follows that:

$$Sat^R(\neg\Phi_1) := R \setminus Sat^R(\Phi_1) = R \setminus (\bigcup_{j=1}^{n_{\Phi_1}} P_j^R) = R \cap \neg(\bigcup_{j=1}^{n_{\Phi_1}} P_j^R) = R \cap \bigcap_{j=1}^{n_{\Phi_1}} P_{\tilde{C},j}^R. \tag{14}$$

Note that the above definition also results in non-disjunct sets $Sat^R(\Phi) \cap Sat^R(\neg\Phi) \neq \varnothing$. Using Equation 12, we can rewrite Equation 14 as the intersection of R with the intersection over all polytopes in $Sat(\Phi_1)$ over the union of all inverted halfspaces per polytope:

$$Sat^R(\neg\Phi_1) = \left(\bigcap_{j=1}^{n_{\Phi_1}} \bigcup_{i=1}^{m_P} h_{i,P}^{\sim}\right) \cap R = \left(\bigcup_{x \in X} \bigcap_{j=1}^{m_P} h_{i,P}^{\sim}\right) \cap R = \bigcup_{x \in X} \left(\bigcap_{j=1}^{m_P} h_{i,P}^{\sim} \cap R\right), \tag{15}$$

where the Cartesian product over all sets H_{P_i} of defining halfspaces for polytopes P_i in $Sat(\Phi_1)$ is defined as $X = \times_{i=1}^{n_{\Phi_1}} H_{P_i}$, forall $P_i^R \in Sat^R(\Phi_1)$.

The second equality follows from the distributive law for families of sets. The last equality results in a set of convex polytopes, restricted to region R. Together, both cases show that the satisfaction set of an STL formula, which does not contain the until operator, can be expressed as a set of convex polytopes.

Time-bounded Until The time-bounded until operator $\Phi := \Phi_1 U^{[0,T]} \Phi_2$ describes a property for paths within the time interval $[0, T]$ relative to time t', hence the algorithm potentially calls all regions that can be reached from the initially called region within the time interval $[t', t'+T]$. Geometrically, these are all regions that lie between the halfspaces h_t and $h_{t+T}^{\sim}$. Recall from Section 3 that each location is reached through a so-called source event. In the geometric representation, this corresponds to a halfspace h_R. Also, a region can be left through its other facets, which mark the entrance into the children of that region in the PLT.

Listing 4 check_until: satisfaction set $Sat^{t',R}(\Phi)$ w.r.t. region R, time hyperplane H and remaining time halfspace $h_{t'+T}$ for $\Phi = \Phi_1 U^{[0,T]} \Phi_2$.

```
 1: set<interval> $I1, I2, I3 \leftarrow \varnothing$        $\triangleright$ Intervals validating $\Phi$ in the respective region.
 2: $Sat^R(\Phi_1) \leftarrow$ modelcheck$(R, \Phi_1)$
 3: $Sat^R(\Phi_2) \leftarrow$ modelcheck$(R, \Phi_2)$
 4: $R \leftarrow R \cap h_{t'+T}$
 5: $I1 \leftarrow$ project$(Sat^R(\Phi_2) \cap H)$        $\triangleright$ Intervals validating $\Phi_2$ immediately.
 6: $C \leftarrow$ project$(Sat^R(\Phi_1) \cap H)$        $\triangleright$ Candidate intervals validating $\Phi_1$ immediately.
 7: $C_{\Phi_1 \cap \Phi_2} \leftarrow$ project$(Sat^R(\Phi_1) \cap Sat^R(\Phi_2))$
 8: $C_F \leftarrow$ project$(Sat^R(\Phi_1) \cap Sat^R(\neg\Phi_1 \wedge \neg\Phi_2))$        $\triangleright$ Projection of boundary states.
 9: $I2 \leftarrow C \setminus (C_F \cap C_{\Phi_1 \cap \Phi_2})$        $\triangleright$ Set of intervals validating $\Phi_1 U \Phi_2$ in $R$.
10: $C_N \leftarrow C \setminus C_F \setminus I2$        $\triangleright$ Removing the non-convex parts of $Sat^R(\Phi_1)$
11: **if** $C_N != \varnothing$ **then**        $\triangleright$ Call function for children intersecting remaining candidates.
12:     **for all** $R_C \in R.children() : H^e_{R,R_c} \cap box(C_N, R) != \varnothing$ **do**
13:         $I3 = I3 \cup (C_N \cap$ check_until$(R_c, \Phi_1, \Phi_2, h_{t+T}, H^e_{R,R_c})))$
14: **return** $I1 \cup I2 \cup I3$
```

According to Equation 9, a state in region R satisfies Φ if (i) it immediately satisfies Φ_2 or if (ii) a state Φ_2 is reached inside R only via Φ_1-states or if (iii) a Φ_2-state outside region R is reached, also only via Φ_1-states. The first two cases can be determined per region and the third case recursively model checks each child until the property is satisfied or time $t' + T$ is reached. As the time bound T is relative to time t', computing the satisfaction of an until operator within a region not only depends on $Sat(\Phi_1)$ and $Sat(\Phi_2)$, but also on their relative distance with respect to time, which may vary within the region. To simplify the matter, we compute $Sat^{t',R}(\Phi_1 U^{[0,T]} \Phi_2)$, (c.f., Equation 9) which corresponds to fixing the t-component for all tuples $(\mathbf{s}, t)$ in a region to time t'.

Listing 4 describes the model checking process and first reduces the identified region to the part that lies before the end of the time interval $[0, T]$, (Line 4) which is reached at time $t' + T$. The remainder of the algorithm operates on families of multi-dimensional intervals which satisfy different combinations of Φ_1 and Φ_2. They are obtained by projecting convex polytopes that are subsets of region R onto the domain of the random variables $\mathbf{S}$. We say that an interval I *validates* a formula Φ at time t, if $\Gamma(\mathbf{s}, t) \models \Phi \forall \mathbf{s} \in I$. All intervals that immediately validate the until property are obtained by intersecting the satisfaction set of Φ_2 with the hyperplane H and then projecting the results onto the $\mathbf{S}$-space (Line 5), as indicated by the function `project`. Initially, `check_until` is called for region R in which the model can be at time t', the hyperplane H is instantiated as $t = t'$. Hence, $I1$ identifies all points in the region that satisfy Φ_2 at time t'.

The algorithm then proceeds to identify those intervals that validate $\Phi_1 U^{[0,T]} \Phi_2$ within the region R. This corresponds to case (ii) in the above explanation. First, a family C of candidate intervals is computed by projecting those states that satisfy Φ_1 when entering the region. This is done by intersecting the satisfaction set of Φ_1 again with hyperplane H (Line 6). Another family of candidate intervals $C_{\Phi_1 \cap \Phi_2}$ is formed by the projection of goal states, i.e., all $(\mathbf{s}, t) \in R$ which satisfy $\Phi_1 \cap \Phi_2$ (Line 7). However, only those candidates $\mathbf{s}$ which, from time t' on, continuously fulfill Φ_1 on their path to a goal state belong to the satisfaction set. Hence, we need to identify boundaries between $Sat^R(\Phi_1)$ and $Sat^R(\neg\Phi_1)$. C_F contains the projection of those facets that do not fulfill Φ_2 (Line 8) and can be computed as the projection of the intersection of $Sat^R(\Phi_1) \cap Sat^R(\neg\Phi_1)$. According to Equation 14, the above intersection returns precisely the points on the boundary between both satisfaction sets, due to the non-disjunct definition of complement.

$I2$ then is computed (Line 9), i.e., the family of intervals validating the until formula within region R by only taking those candidates $\mathbf{s}$ whose time evolution $(\mathbf{s}, t)$ for $t \geq t'$ continuously satisfies Φ_1 and finally reaches a Φ_2-state within region R before $t = t' + T$. To compute the family of intervals which validate the until formula by reaching a Φ_2-state in another region, first, the family of intervals whose time evolution continuously satisfies Φ_1 and which do not reach a Φ_2-state within region R is computed (Line 10). If this collection is non-empty (Line 11), the model checking algorithm is called for each child of R, restricting the potential domain to the box defined by C_N and the states that can be reached only via Φ_1-states in R (Line 12–13). For each child, we collect those candidates which reach a Φ_2-state before time $t' + T$ within that child and intersect them with the candidates whose time evolution continuously satisfies Φ_1 and does not satisfy Φ_2 within the current region. Calling algorithm `check_until` (Line 13) instantiates hyperplane H as source event for each child location to account for different possible entrance times of child locations. The algorithm returns the family of intervals of the $\mathbf{S}$ domain (Line 14), which validate the until formula in that region from time t' on.

The approach is illustrated for three regions in Figure 3. The model checking algorithm is called for an until formula $\Phi_1 U^{[0,T]} \Phi_2$ and for region R_1, where H is initiated as hyperplane $H_{t'} = h_{t'} \cap h^{\sim}_{t'}$. The intersection of $H_{t'}$ and $Sat^R(\Phi_2)$ is empty, hence $I1$, containing those states that immediately satisfy the until formula, is empty. The candidate intervals are computed by projecting the intersection of $H_{t'}$ and $Sat^R(\Phi_1)$ and indicated in Figure 3.

Φ_2 does not hold in Region R_1, hence $C_{\Phi_1 \cap \Phi_2}$ and $I2$ are empty. In the next step the facets between the polytopes representing $Sat^R(\Phi_1)$ and $Sat^R(\Phi_2)$ are projected as C_F (also shown below). They represent those parts of the domain whose time evolution after t' not continuously satisfies Φ_1. Hence, they have to be subtracted from the candidate intervals, yielding C_N. Since the latter is not empty, model checking is called recursively for the children that have a non-

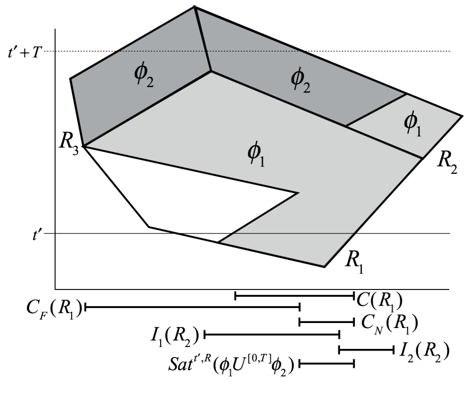

Fig. 3. Intervals for time-bounded until.

empty intersection with C_N. In this example, the function is only called for R_2, where H is initiated as hyperplane representing the occurrence time of its source event. First, $I1$ is the projection of H with $Sat^R(\Phi_2)$, as indicated below the state space. Then $I2$ is obtained as the intersection of the projection of all candidates with the projection of the facets between the polytopes representing $Sat^R(\Phi_1)$ and $Sat^R(\Phi_2)$. Note that the computation of `check_until` for region R_2 requires calling the function again for the child of R_2 which lies above the polytope representing $Sat^R(\Phi_1)$ (not illustrated in the figure). For this part of the candidates a Φ_2-state cannot be reached in R_2, nor is the end of the time bound reached. The result of the function call for R_2 is intersected with the intervals in C_N of region R_1 and returned as $Sat^{t'}(\Phi)$.

5.2 Computing $Sat^{t'}(\Phi)$ for nested formula and complexity

We have shown that model checking atomic and compound formulas Φ generally results in a set of convex polytopes, containing all tuples $(\mathbf{s}, t)$ that satisfy Φ. Model checking an until operator, however returns a set of intervals, i.e. all $\mathbf{s} \in \mathbf{S}$, which validate the formula at time t'. To enable the conjunction of an until operator with another arbitrary STL formula, its satisfaction set needs to be lifted back into the region, by adding time t' to all elements $\mathbf{s} \in Sat^{t'}(\Phi_1 U^{[0,T]} \Phi_2)$. This results in one convex polytope per interval. Convexity results directly from the use of intersection, set difference and projection.

The general routine `modelcheck(`R, Φ`)` recursively calls the operator-specific functions, as introduced above, along the parse tree of the formula. To compute the overall satisfaction set $Sat^{t'}(\Phi)$, the satisfaction sets of all candidate regions $Sat^R(\Phi)$ have to be intersected with the hyperplane representing time $H_{t'}$. The results are projected onto the $\mathbf{S}$-space and the resulting validity intervals are combined for all candidate regions. If the STL formula Φ is wrapped inside a probability operator, multi-dimensional integration is performed over the resulting set $Sat^{t'}(\Phi)$ using the density function of each random variable combined with the branching probabilities. The computation of the PLT and the multi-dimensional integration are explained in [16].

The complexity of the overall model checking routine depends on the number of regions in the PLT $|R|$ and the number of operators in the STL formula $|L|$. Negation requires geometric operations on the cross product of polytopes, which is cubic in the number of halfspaces ($\mathcal{O}(|H_P|^3)$). Model checking Until relies on a series of geometric operations, where polytope inversion has the worst case complexity (similar to negation) and accesses at most $|R|$ children. The worst case complexity of the overall model checking routine is then $\mathcal{O}(|H_P|^3 \times |R|^2 \times |L|)$, as it might be called for all regions. The dimensionality of halfspaces and regions influences the the complexity of the geometric operations.

Model checking nested formula might result in a large list of convex polytopes, caused by negating non-atomic formulas (c.f. Section 5). This effect can be reduced by rewriting the propositional parts of an STL formula in disjunctive normal form. When negation is applied directly to atomic properties, it does not increase the number of convex polytopes in the representation.

			$\Phi_1 := (m_U = 0)$, $t' = 4$		$\Phi_2 := \neg(x_B \leq 2)$, $t' = 4$		Φ_3, $t' = 0$	
n	$\lvert\mathbf{L}\rvert$	$\lvert\mathbf{C_R}\rvert$	$\lvert Sat^{t'}(\Phi_1)\rvert$	t_c [ms]	$\lvert Sat^{t'}(\Phi_2)\rvert$	t_c [ms]	$\lvert Sat^{t'}(\Phi_3)\rvert$	t_c[ms]
1	9	7	4	3	4	112	5	208
2	31	20	7	12	14	2327	3	3200
3	139	97	68	61	63	37337	4	69218
4	667	456	327	306	320	897511	7	2055254
5	3683	2338	1797	2100	1961	26790200	N/A	N/A

Table 1. Results for model checking Φ_1, Φ_2, and $\Phi_3 := (m_U = 1)\mathcal{U}^{[0,T]}(x_B \leq 0)$.

Example 5. Table 1 shows results for checking Φ_1 and Φ_2 at time $t' = 4$ and Φ_3 at time $t' = 0$, for a varying number of input pumps. The computations have been performed on a MacBook Pro with 2.5 GHz i7 and 16 GByte RAM. The number of locations generated before $\tau_{max} = 20$ is indicated by $|L|$. Per formula, we provide the number of candidate locations ($|C_R|$), the number of intervals stored in the satisfaction set ($|Sat^{t'}|$), and the respective computation times. The number of candidate regions grows considerably with the number of random variables present in the system. The computation times are much larger for the until formula, $n = 4$ required 34 minutes and $n = 5$ could not be solved. The large computation times for model checking an until formula are due to the required geometric operations within a region and the recursive call for child regions. In contrast model checking Φ_1 only requires to check the discrete marking, with is done in constant time per region. Checking Φ_2 is more time consuming, due to negation, but results for $n = 5$ can be obtained. The number of candidates is slightly smaller for checking Φ_3 at $t' = 0$, as less branching has taken place. Due to space limitations, this number is not included in the table.

6 Conclusions

We proposed model checking algorithms for STL operators that can be used to check properties of HPnGs with an arbitrary but finite number of stochastic firings, working only on convex state set representations. While the current paper does not provide a framework to compute the probability that an STL formula holds, the current results in terms of validity intervals can be used to synthesize parameters for the timing of general transitions, which validate a specific formula. To the best of our knowledge, we present a model checking approach for a type of Hybrid Petri nets, that is neither restricted in the number of continuous variables, nor in the number of stochastic firings. Future work will present an algorithm to compute the complete satisfaction set $Sat^R(\Phi)$ for the until operator and compare computational complexities and efficiency of both approaches, as well as an algorithm to evaluate the probability operator, taking into account branching probabilities between locations. Furthermore, we plan to conduct a large-scale case study, to evaluate the efficiency of the current implementation. The transformation of a PLT to hybrid automata is being investigated.

References

1. A. Abate, J.-P. Katoen, J. Lygeros, and M. Prandini. Approximate Model Checking of Stochastic Hybrid Systems. *European Journal of Control*, 6:624–641, 2010.
2. H. Alla and R. David. Continuous and hybrid petri nets. *Journal of Circuits, Systems, and Computers*, 8(01):159–188, 1998.
3. R. Alur, T. Feder, and T.A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.
4. R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.
5. M.H.C Everdij and H.A.P. Blom. Piecewise deterministic Markov processes represented by dynamically coloured Petri nets. *Stochastics*, 77(1):1–29, 2005.
6. G. Frehse, Zhi Han, and B. Krogh. Assume-guarantee reasoning for hybrid I/O-automata by over-approximation of continuous interaction. In *43rd IEEE Conf. on Decision and Control*, pages 479–484, 2004.
7. G. Frehse, R. Kateja, and C. Le Guernic. Flowpipe approximation and clustering in space-time. In *16th Int. conf. on Hybrid systems: computation and control*, pages 203–212. ACM, 2013.
8. H. Ghasemieh, A. Remke, and B. Haverkort. Survivability analysis of a sewage treatment facility using hybrid petri nets. In *Performance Evaluation*, volume 97, pages 36–56. Elsevier, 2016.
9. H. Ghasemieh, A. Remke, B. Haverkort, and M. Gribaudo. Region-based analysis of hybrid petri nets with a single general one-shot transition. In *Int. Conf. on Formal Modeling and Analysis of Timed Systems*, pages 139–154. Springer, 2012.
10. H. Ghasemieh, A. Remke, and B. R. Haverkort. Hybrid petri nets with multiple stochastic transition firings. In *8th Int. Conf. on Performance Evaluation Methodologies and Tools, 2014*, pages 217–224. ICST, 2014.
11. H. Ghasemieh, A. Remke, and B.R. Haverkort. Survivability Evaluation of Fluid Critical Infrastructures Using Hybrid Petri Nets. In *IEEE 19th Pacific Rim Int. Symp. on Dependable Computing*, pages 152–161. IEEE, 2013.
12. A. Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems: Computation and Control*, volume 3414 of *LNCS*, pages 291–305. Springer, 2005.
13. A. Godde and A. Remke. Model Checking the STL Time-Bounded Until on Hybrid Petri Nets Using Nef Polyhedra. In *European Workshop on Performance Engineering*, pages 101–116. Springer, 2017.
14. M. Gribaudo and A. Remke. Hybrid petri nets with general one-shot transitions. *Performance Evaluation*, 105:22–50, 2016.
15. G. Horton, V.G. Kulkarni, D.M. Nicol, and K.S. Trivedi. Fluid stochastic Petri nets: Theory, applications, and solution techniques. *Journal of Operational Research*, 105(1):184–201, 1998.
16. J. Hüls, C. Pilch, P. Schinke, J. Delicaris, and A. Remke. State-space construction of Hybrid Petri nets with multiple stochastic firings. Technical report, Westfälische Wilhelms-Universität Münster, 2018. Available at https://uni-muenster.sciebo.de/s/BMwdh25rHgmDvb6.
17. J. Hüls, S. Schupp, A. Remke, and E. Ábrahám. Analyzing Hybrid Petri nets with multiple stochastic firings using HyPro. In *11th Int. Conf. on Performance Evaluation Methodologies and Tools*, 2017.

18. M. R. Jongerden, J. Hüls, A. Remke, and B. R. Haverkort. Does your domestic photovoltaic energy system survive grid outages? *Energies*, 9(9):736, 2016.
19. A. A. Julius. Approximate Abstraction of Stochastic Hybrid Automata . In *Int. Conf. on Hybrid Systems: Computation and Control*, pages 318–332. Springer, 2006.
20. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1):101 – 150, 2002.
21. C. Le Guernic and A. Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250–262, 2010.
22. O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.
23. R. E. Moore, R. B. Kearfott, and M. J. Cloud. *Introduction to interval analysis*. SIAM, 2009.
24. C. Pilch and A. Remke. Statistical Model Checking for hybrid Petri nets with multiple general transitions. In *47th Int. Conf. on Dependable Systems and Networks*, pages 475–486. IEEE, 2017.
25. S. Schupp, E. Ábrahám, I. B. Makhlouf, and S. Kowalewski. Hypro: A c++ library of state set representations for hybrid systems reachability analysis. In *9th Int. Symp. NASA Formal Methods*, pages 288–294. Springer, 2017.
26. S. Esmaeil Zadeh Soudjani and A. Abate. Adaptive and sequential gridding procedures for the abstraction and verification of stochastic processes. *SIAM Journal on Applied Dynamical Systems*, 12(2):921–956, 2013.
27. S. Esmaeil Zadeh Soudjani, C. Gevaerts, and A. Abate. FAUST: Formal Abstractions of Uncountable-STate STochastic Processes. In *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, pages 272–286. Springer, 2015.
28. J. Sproston. Decidable model checking of probabilistic hybrid automata. In *Int. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 31–45. Springer, 2000.
29. T. Teige and M. Fränzle. Constraint-based analysis of probabilistic hybrid systems. *IFAC Proceedings Volumes*, 42(17):162 – 167, 2009.
30. The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.10 edition, 2017.
31. L. Zhang, Z. She, S. Ratschan, H. Hermanns, and E. M. Hahn. Safety verification for probabilistic hybrid systems. *European Journal of Control*, 18(6):572 – 587, 2012.
32. G. M. Ziegler. *Lectures on polytopes*, volume 152. Springer Science & Business Media, 2012.