Analyzing Hybrid Petri nets with multiple stochastic firings using HyPro

Jannik Hüls
Westfälische Wilhelms-Universität Münster
Germany
jannik.huels@wwu.de

Stefan Schupp
RWTH Aachen University
Germany
stefan.schupp@informatik.rwth-aachen.de

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

Erika Ábrahám
RWTH Aachen University
Germany
abraham@informatik.rwth-aachen.de

ABSTRACT

Two approaches have been presented for the analysis of HPnGs, (i) a tree-based approach and (ii) a state-space representation based on computational geometry. We present a translation of the tree-based representation into a geometric representation using the C++ library HyPro, which has been developed for the analysis of hybrid automata. This allows the representation and efficient and accurate analysis of HPnGs with multiple stochastic firings.

KEYWORDS

Hybrid Petri nets, symbolic analysis, convex polyhedra, HyPro

ACM Reference format:
Jannik Hüls, Stefan Schupp, Anne Remke, and Erika Ábrahám. 2017. Analyzing Hybrid Petri nets with multiple stochastic firings using HyPro. In *Proceedings of 11th EAI International Conference on Performance Evaluation Methodologies and Tools, Venice, Italy, December 5–7, 2017 (VALUETOOLS 2017),* 8 pages.
https://doi.org/10.1145/3150928.3150938

1 INTRODUCTION

Hybrid systems are systems that combine continuous and discrete behavior. Motivated by their frequent appearance and safety-critical nature, a variety of modeling languages and (exact or approximate) analysis methods for hybrid systems have been proposed in computer science and control theory. *Hybrid automata* are one of the most popular modeling formalism, shown to be suitable for modeling industrial applications as well as academic problems. For the *reachability analysis* of hybrid automata and subclasses thereof, different approaches based on theorem proving, bounded model checking or flowpipe construction have been proposed and implemented in numerous tools and libraries (see Section 3).

VALUETOOLS 2017, December 5–7, 2017, Venice, Italy

ACM ISBN 978-1-4503-6346-4/17/12…$15.00
https://doi.org/10.1145/3150928.3150938

Hybrid Petri nets [1] are another modeling formalism for hybrid systems, which extend the notation of Petri nets by continuous behavior. *Hybrid Petri nets with general transitions* (*HPnGs*) further extend hybrid Petri nets by adding so-called *general transitions*, that fire after being enabled for a random amount of time. The firing of such a general transition is called a *stochastic firing*. As proposed in [15], they provide a high-level formalism and have recently been used to model critical infrastructures like water sewage facilities [11], smart homes [18] and electric vehicle charging systems [17].

There are two forward symbolic reachability approaches for HPnGs. They both represent time-bounded system executions symbolically in the form of a tree, where the nodes represent state sets and the paths represent system executions. The first method named *parametric reachability analysis* [16] generates a tree called *parametric location tree* (*PLT*), whereas the second method computes a *stochastic time diagram* [10].

However, the two methods use different representations for the state sets in the tree nodes: the first method uses logical specifications and the latter one uses convex polyhedra as a geometric representation. Both representations are symbolic, but they have different strengths and weaknesses: the logical representation can be computed fast but it is expensive to compute intersections (to determine the subset that satisfies a certain property), whereas it is time-consuming to compute the second representation but then the intersection computation can be done fast.

In its original form, the parametric reachability analysis approach is restricted to one general transition firing. Stochastic time diagrams do not have this restriction, but for each general transition firing they introduce an extra dimension in the state space. Furthermore, their generation relies on two well-kown geometrical problems, namely hyperplane arrangement and halfspace intersection. To the best of our knowledge no implementation for hyperplane arrangement exist for more than three dimensions, hence, current approaches for analyzing HPnGs [12, 14] are restricted to three dimensions.

To overcome the limitation in the number of general transition firings this paper presents a transformation which generates a geometric representation for each node of the parametric location tree and fully relies on halfspace intersection, which is dual to the convex-hull problem and can be solved e.g. using quick hull [3] also for higher dimensions. This combines the computational speed of the parametric reachability analysis with the exact analysis based

on STDs. We use the C++ library HyPro to create a region for each parametric location. HyPro allows creating the convex polyhedron representing a region also for higher dimensions using convex hull algorithms. This allows the analysis of Hybrid Petri nets with a finite number of multiple stochastic firings.

The contribution of this paper is the construction of the state space of HPnGs with an arbitrary but finite number of stochastic firings using HyPro. This is done by first constructing the parametric location tree up to a maximum time of analysis and by then building the geometric representation for each location in the tree. Note that we extended the existing algorithm for parametric reachability analysis for multiple stochastic firings. In case multiple general transitions are present or / and may fire multiple times, we obtain competing random variables which results in a tree with a larger node degree. Furthermore, we explain how subsets of the cross product of the domains of all random variables present in the system can be found that fulfill certain properties of the model at a given time t.

This paper is further organized as follows. Section 2 recalls the HPnG formalism and the model definition. Section 3 describes the library HyPro used for the analysis. Section 4 extends the parametric location tree for multiple stochastic firings and Section 5 introduces the translation of parametric locations into regions represented as $\mathcal{H}$-polyhedra. The paper concludes in Section 6.

2 HYBRID PETRI NETS WITH MULTIPLE GENERAL TRANSITION FIRINGS

Hybrid Petri nets with multiple general transition firings model systems with *hybrid*, i.e., mixed discrete-continuous behavior. The formalism extends hybrid Petri nets [1] with so-called *general transitions*, that fire according to an arbitrary but fixed probability distribution. A HPnG consists of three main model components, as shown in Figure 1.

The modeling formalism of HPnGs is discussed in Section 2.1 and a small model is introduced as running example in Section 2.2.

2.1 Model definition

Places can be discrete or continuous. Discrete places are represented by single circles and continuous places by double circles. The number of tokens in a discrete place and the amount of fluid in a continuous place constitute the marking of the Petri net. **Transitions** change the marking of the Petri net. Discrete transitions (general, deterministic and immediate) trigger a change of the discrete marking upon firing. Deterministic transitions fire after being enabled for a constant predefined amount time. General and immediate transitions can both be considered as special types of deterministic transitions. As mentioned above, general transitions fire according to an arbitrary probability distribution. Immediate transitions can be considered as deterministic transitions that fire after being enabled for zero time. Continuous transitions change the fluid level of connected input and output places with a constant nominal rate, which may however be adapted, when one of the connected places reaches its upper or lower boundary, i.e. so-called rate adaptation. **Arcs** connect places and transitions. They define the change of marking that is triggered by the firing of a transition,

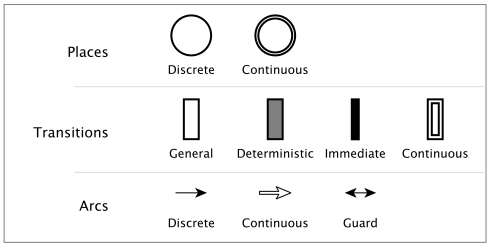

Figure 1: Primitives of the hybrid Petri nets with multiple general transitions formalism.

by defining input and output places and assigning their corresponding weight and priority. Guard arcs enable transitions based on the discrete or continuous marking of connected places.

Following the definition in [12, 16] we define hybrid Petri nets with multiple stochastic firings.

Definition 2.1 (HPnG). A 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 as Φ which defines the behavior of the hybrid Petri net.

The finite set $\mathcal{P} = \mathcal{P}^D \cup \mathcal{P}^C$ consists of discrete and continuous places. The finite set of transitions $\mathcal{T} = \mathcal{T}^I \cup \mathcal{T}^D \cup \mathcal{T}^G \cup \mathcal{T}^C$ holds immediate, deterministic, general and continuous transitions. The set $\mathcal{A}$ is divided into three subsets: (i) The set of discrete arcs $\mathcal{A}^D \subseteq ((\mathcal{P}^D \times \mathcal{T}^D) \cup (\mathcal{T}^D \times \mathcal{P}^D))$ connecting discrete places and transitions. (ii) The set of continuous arcs $\mathcal{A}^C \subseteq ((\mathcal{P}^C \times \mathcal{T}^C) \cup (\mathcal{T}^C \times \mathcal{P}^C))$ connecting continuous places and transitions. (iii) The set of guard arcs $\mathcal{A}^G \subseteq ((\mathcal{T}^D \times (\mathcal{P}^D \cup \mathcal{P}^C)) \cup (\mathcal{T}^C \times \mathcal{P}^D))$ connecting discrete transitions to either discrete and continuous places; and connecting continuous transitions only to discrete places.

The initial marking $\mathbf{M}_0 = (\mathbf{m}_0, \mathbf{x}_0)$ denotes the initial number of tokens and fluid levels of the places. The tuple

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

contains eight parameter functions that further define the behavior of the Hybrid Petri net. Each function is used to define the model in the following. However we refer to [10] for further details of the mapping.

Marking. A discrete place $P_i^D \in \mathcal{P}^D$ contains a non-negative number of *tokens* denoted as $m_i \in \mathbb{N}$ and a continuous place $P_j^C \in \mathcal{P}^C$ has a of non-negative *level* of fluid $x_j \in \mathbb{R}_{\geq 0}$. The content of all places is called *marking* and combines the discrete marking $\mathbf{m} = (m_1, ..., m_{|\mathcal{P}^D|})$ and the continuous marking $\mathbf{x} = (x_1, ..., x_{|\mathcal{P}^C|})$ into $\mathbf{M} = (\mathbf{m}, \mathbf{x})$. We use $m(\mathcal{P}_i^D)$ and $x(\mathcal{P}_i^C)$ to denote the number of tokens and the fluid level of the i-th discrete and continuous place, respectively.

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 cumulative distribution function (CDF) to each general transition. $\Phi_u^{\mathcal{A}} : \mathcal{A}^G \to \{\triangleleft, \mathbb{R}\}$ with $\triangleleft = \{\geq, <\}$ assigns a comparison operator and a real

number to each guard arc. This mapping implicitly encapsulates the weight of the guard arc determining the amount of fluid or tokens needed to enable the connected transition. $\Phi_n^{\mathcal{A}} : \mathcal{A}^D \to \mathbb{R}^+$ assigns an integer to each discrete input or output arc to determine the number of tokens moved when the transition fires. A transition may only fire if it is enabled.

We define the set $\mathcal{I}_T(T)$ as the set of all input places of transition T. Accordingly $\mathcal{O}_T(T)$ defines the set of all output places of transition T. The set $\mathcal{G}_T(T)$ denotes the set of places connected to transition T via guard arcs. A discrete transition $T^d \in \mathcal{T}^D$ is said to be enabled if one of the following conditions hold:

(i) Discrete guard arc reaching its weight q:

$$\forall P^d \in \mathcal{P}^D \cap \mathcal{G}(T^d), (\triangleleft, q) = \Phi_u^A(\langle T^D, P^d \rangle) : m(P^d) \triangleleft q,$$

(ii) Continuous guard arc reaching its weight q:

$$\forall P^c \in \mathcal{P}^C \cap \mathcal{G}(T^d), (\triangleleft, q) = \Phi_u^A(\langle T^D, P^d c \rangle) : x(P^c) \triangleleft q,$$

(iii) Connected input place reaching the minimum amount of tokens:

$$\forall P^d \in \mathcal{I}(T^d) : m(P^d) \geq \Phi_n^A(\langle P^d, T^d \rangle).$$

Continuous transitions are only allowed to be connected to discrete places via guard arcs. Hence there are only two conditions to test if a continuous transition $T^c \in \mathcal{T}^C$ is enabled:

(i) Discrete guard arc reaching its weight q:

$$\forall P^d \in \mathcal{P}^D \cap \mathcal{G}(T^c), (\triangleleft, q) = \Phi_u^A(\langle T^c, P^d \rangle) : m(P^d) \triangleleft q,$$

(iii) Connected input place having any level of fluid:

$$\forall P^c \in \mathcal{I}(T^c) : x(P^d) > 0.$$

System evolution. Discrete transitions are associated with clocks. Let c_i be the clock associated with transition $T_i^d \in \mathcal{T}^D$. If transition T_i^d is enabled c_i evolves with $dc_i/dt = 1$, otherwise $dc_i/dt = 0$. A deterministic transition fires when c_i reaches the transitions firing time defined by $\Phi_d^{\mathcal{T}} : \mathcal{T}^D \to \mathbb{R}^+$. When a discrete transition fires, the corresponding marking changes according to the weights of the input and output arcs, as specified in $\Phi_n^{\mathcal{A}} : \mathcal{A} \to \mathcal{R}^+$.

A continuous transition that is enabled fires according to the nominal in- and out-flow rates. The rates of transitions that are connected to a place that is either at its upper or lower boundary have to be adapted using *rate adaptation*. Then the inflow is decreased to match the outflow or the outflow is reduced to match the inflow changing the actual fluid rate of a transition. Let $r(T^c)$ be the actual rate of the continuous transition $T^c \in \mathcal{T}^C$. The in-flow $f_{in}(P^c)$ of a place $P^c \in \mathcal{P}^C$ is defined as $f_{in}(P^c) = \sum_{T_c \in \mathcal{I}_P(P^c)} r(T^c)$, i.e., the sum of rates of all the incoming transitions. Similarly the out-flow is the defined as $f_{out}(P^c) = \sum_{T_c \in \mathcal{O}_P(P^c)} r(T^c)$. A continuous place then evolves with the *drift* $d(P^c) = f_{in}(P^c) - f_{out}(P^c)$. Moreover we have $dx(P^c)/dt = d(P^c)$. We refer to [15] for the advanced concepts of *concession*, *shares* and *conflict resolution*, and do not explicitly define them as they are not needed to understand the introduced analysis concept.

State. The concept of a state of an HPnG summarizes all information necessary for the analysis of an HPnG.

Definition 2.2 (State). The state of a HPnG is defined as a tuple $\Gamma = (\mathbf{m}, \mathbf{x}, \mathbf{c}, \mathbf{d}, \mathbf{g})$ where

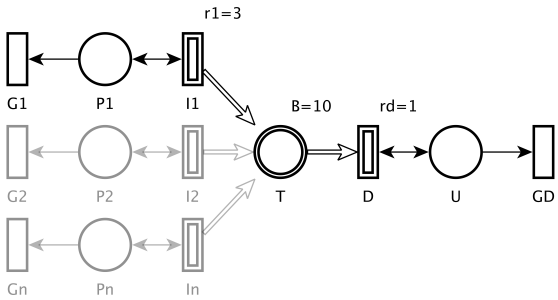

Figure 2: Example of a reservoir model with a scalable number of general transitions.

- $\mathbf{m}$ and $\mathbf{x}$ are the markings of the discrete and continuous places respectively,
- $\mathbf{c} = (c_1, ..., c_{|T^D|})$ is the vector of discrete clocks,
- $\mathbf{d} = (d_1, ...d_{|T^C|})$ contains the drift of each continuous place,
- $\mathbf{g} = (g_1, ...g_{|T^G|})$ indicates the enabling time of each general transition.

The initial state is defined as $\Gamma_0 \in S$ with $\Gamma_0 = (\mathbf{m}_0, \mathbf{x}_0, \mathbf{0}, \mathbf{d}_0, \mathbf{0})$. It is the starting point for the analysis method described in Section 4 and Section 5. The state space $S = \{\Gamma = (\mathbf{m}, \mathbf{x}, \mathbf{c}, \mathbf{d}, \mathbf{g}, \mathbf{e})\}$ contains all reachable states of an HPnG. States are separated by *events*, i.e. an event leads to a state change. Three types of events can occur:

(i) A continuous place reaching its lower or upper boundary.
(ii) A continuous place reaches the weight of a connected guard arc.
(iii) The firing of an enabled discrete transition.

The concept of events is later used to construct the *parametric location tree* (PLT) and to explain the relation to *stochastic time diagrams* (STD).

2.2 Running example

We will illustrate the algorithms presented in this paper using a small example, that is inspired by [15], but is scalable in the number of general transitions and their respective firings. Figure 2 shows the HPnG model of a tank, represented by the continuous place `T` with a maximum capacity of `B` = 10 units. It is initially empty and can be filled using possibly n different producer pumps, represented by continuous transitions $\mathtt{I}_1 ... \mathtt{I}_n$. Each of these pumps has a different nominal rate $\mathtt{r}_i$. The general transitions $\mathtt{G}_1, ..., \mathtt{G}_n$ model the failure of the corresponding producer. The continuous transition `D` models the consumer pump with nominal rate $\mathtt{r}_d = 1$. In contrast to [15] a general transition $\mathtt{G}_D$ is used to control the consumer pump. Note that the number of pumps and general transitions is scalable in order to increase the model complexity.

3 HYPRO

The algorithm presented in Section 4 heavily relies on the `C++` library HyPro [22], that is dedicated to the safety verification of linear hybrid automata. Originally designed to support fast and easy implementation of flowpipe construction based reachability

analysis methods for hybrid systems [19], HyPro incorporates datastructures and utility functions required as well as implementations for the most commonly used state set representations for hybrid systems reachability analysis. Among representations such as boxes [21], support functions [20] or zonotopes [13], HyPro includes an implementation of convex polytopes [24] as well as a wrapper class to the well-known Parma polyhedra library (PPL) [2]. All implemented state set representations are unified under a common, convenient interface and are convertible into each other (guaranteeing over-approximation). Furthermore, all representations are templated in the used number type, allowing to utilize either native floating point arithmetic or rational number types such as provided by the GNU multiple precision arithmetic library (GMP).

When utilizing the proprietary convex polytopes implemented in HyPro, the user can choose between two representations: Either a polytope P (a bounded polyhedron) is represented as the intersection of a finite set of halfspaces stored as a pair of a matrix A and a vector b such that $P = \{x \mid A \cdot x \leq b\}$. This representation is referred to as $\mathcal{H}$-representation and the resulting polytope is called $\mathcal{H}$-polytope. Equivalently, a convex polytope can be represented as the convex hull of a finite set of vertices v_i, i.e. $P = \{x \mid x = \Sigma_i^n \lambda_i \cdot v_i, \Sigma_i^n \lambda_i = 1\}$. This representation is known as $\mathcal{V}$-representation. Note that while the $\mathcal{H}$-representation almost naturally supports polyhedra (unbounded polytopes), the $\mathcal{V}$-representation requires an extension: a general polyhedron P in $\mathcal{V}$-representation is a tuple (V, C), where V is a set of vertices v_i and C is a cone. The resulting polyhedron $P = cHull(V) \oplus C$ is the Minkowski sum of a cone and the convex hull of a finite set of vertices.

Computational effort varies for set operations depending on the utilized representation: e.g. an affine transformation is easier to compute in $\mathcal{V}$-representation (polynomial time) while the intersection of two convex polytopes requires less computational effort in $\mathcal{H}$-representation (linear). However, both representations can be converted into each other via convex hull or vertex enumeration algorithms in worst case exponential time (average: polynomial). For further details about convex polytopes we refer the reader to [24].

Our implementation is based on $\mathcal{H}$-polytopes of HyPro (see Section 4.2), where previous implementations [12, 14] utilized CGAL, which implements Nef-polyhedra but is limited to three dimensions. Nef polyhedra are closed under Boolean set operations such as union, intersection and set difference and thus are not necessarily convex. The need for Nef polyhedra is rooted in verifying until operators, which implies non-convexity as set difference is required during the verification. In contrast to this, HyPro does not limit the number of dimensions and provides most of the required set operations required for our analysis of HPnGs.We are aware that there exist various implementations of convex polyhedra, e.g. PPL [2], polymake [8], CGAL [23], cdd [7] which can be used as well but do not provide intuitive, convenient interfaces and conversion functions. Furthermore we can exploit other features of HyPro for future work regarding analysis methods originating in flowpipe construction.

4 EXTENDED DATASTRUCTURE

The *parametric location tree* (PLT) was introduced in [15] for the analysis of HPnGs with a single general one-shot transition. We extend the formalism to support multiple general transition firings and translate the resulting locations into a geometric representation, as explained in Section 5. Note that each stochastic firing corresponds to one random variable that is distributed according to the CDF of the general transition that fired. The value of that random variable then corresponds to the time the general transition has been enabled before said firing.

Section 4.1 adapts the definition of a PLT to multiple stochastic firings. The PLT for the running example is constructed in Section 4.2.

4.1 Parametric Location Tree

Although the state of the system changes continuously, its bounded evolution up to some maximum time $\tau_{\max}$ can be described in a symbolic way using a *tree-based structure*. Each node of the tree represents a parametric location, which is defined as a tuple $\Lambda = (t, \mathbf{m}, \mathbf{x}, \mathbf{d}, \mathbf{c}, \mathbf{g}, \mathbf{S}, p)$. The time at which the system enters the parametric location is denoted t and $\mathbf{m}$, $\mathbf{x}$, $\mathbf{d}$, $\mathbf{c}$ and $\mathbf{g}$ follow Definition 2.2.

Each random variable may be concurrently enabled and their enabling time may depend on each other. Vector $\mathbf{g}$ collects the enabling time for each general transition present in the model. Hence, the size of vector $\mathbf{g}$ is $|\mathcal{T}^G|$ and the entry g_i is reset to zero when transition T_i^G fires. The potential domain $\mathbf{S}$ provides the bounds for each general transition firing per parametric location.

For each parametric location the range of possible values is summarized in the potential domain $\mathbf{S} = ([l_0, r_0], ..., [l_n, r_n])$. The size of the vector $\mathbf{S}$ is equivalent to the number of random variables present in the system. A new random variable is instantiated each time a general transition is enabled for the first time or after a previous firing. Hence, in total the number of random variables n equals the number of stochastic firings that occurred plus the number of general transitions that have been enabled but have not fired before $\tau_{\max}$. The vector $\mathbf{s} = (s_0, ..., s_n)$ collects all random variables representing said stochastic firings, and we define

$$\mathbf{s} \in \mathbf{S} \text{ iff } s_i \in [l_i, r_i] \quad \forall 0 \leq i \leq n.$$

Time t, vector $\mathbf{x}$, $\mathbf{c}$ and $\mathbf{S}$ are all linear functions of the firing times of the general transitions $\mathbf{s}$. To simplify the presentation we will not show their dependency explicitly. Finally, p is the probability of choosing a particular location if two or more events are scheduled at the same time, as discussed in Section 2.1.

Tree generation. The PLT is generated using a depth-first search by extending all parametric locations until $\tau_{\max}$ is reached. We start from the initial parametric location

$$\Lambda_0 = (\mathbf{0}, \mathbf{m_0}, \mathbf{x_0}, \mathbf{d_0}, \mathbf{c_0}, \mathbf{g_0}, \mathbf{S}_0, \mathbf{0}),$$

where $\mathbf{S}_0$ is a n-dimensional vector $([0, \tau_{\max}], \ldots, [0, \tau_{\max}])$, specifying all considered values of $\mathbf{s}$ in this computation. The initial marking $M_0 = (\mathbf{m}_0, \mathbf{x}_0)$ contains the initial number of tokens $\mathbf{m}(\mathcal{P}_i^D)$ of each discrete place and the initial amount of fluid $\mathbf{x}(\mathcal{P}_i^C)$ of each continuous place respectively. $\mathbf{d}_0$ denotes the initial drift of

each continuous place. Note that the initial clock values $\mathbf{c_0}$ and the general transition enabling times $\mathbf{g}_0$ are initially set to $\mathbf{0}$.

In each location the time until the next event $\tau_{\min}$ is computed relative to the entry time t of the respective location. The computation of the next location considers all possible events that occur at the next minimum event time. A new parametric location is reached with each possible event happening at that time, after which the procedure is called recursively. Instead of the deterministic events taking place at time $t + \tau_{\min}$, each enabled general transition may fire before that point in time. Hence, an additional successor needs to be scheduled for each enabled general transition and the potential domains have to be set accordingly.

The *next minimum event time* $\tau_{\min}$ is unique before the first stochastic firing. In that case, the new minimum time to fire is the minimum of the remaining times to fire of all enabled deterministic transitions with highest priority. Then all fluid places $P^c \in \mathcal{P}^C$ with non-zero drift have to be checked, whether they reach their upper or lower boundary or enable a guard arc connected to it before $\tau_{\min}$ has elapsed. The time to reach a threshold $0 \leq \gamma \leq \Phi_b^P(P^c)$ is given by $\tau_{\min} = (\gamma - \mathbf{x}[P^c])/\mathbf{d}[P^c])$, where $\mathbf{x}[P^c]$ is the continuous marking and $\mathbf{d}[P^c]$ the corresponding drift. For the exact computation of the next event, we refer to [9].

After a single stochastic firing s_i, the time in which a location is entered, the clocks, the continuous marking and the potential domains may linearly depend on the value of the corresponding random variable s_i. Since an arbitrary but finite number of random variables is present in the system and the computation of the next minimum event (for all event types) may depend on all transition firing times, it is based on polynomials and not scalars.

Note that for the case of multiple general transition firings, we obtain a vector of $\mathbf{s}$, which leads to multi-dimensional linear equations. Again the potential domains $\mathbf{S}$ linearly depend on $\mathbf{s}$. Recall that $\mathbf{S}$ is a vector of intervals. We define a corresponding vector $\mathbf{l}$ that collects all left interval bounds and a vector $\mathbf{r}$ that collects the right interval bounds, respectively. Let $f_i(\mathbf{s})$ be the linear equation describing the firing time of an event. Then $m_i = \{\mathbf{s} \in \mathcal{R}_+^n | f_1(\mathbf{s}) \leq \ldots \leq f_n(\mathbf{s})\}$ contains a set of vectors $\mathbf{s}$ denoting the boundaries for which event f_i occurs next. Each left interval bound $l_j \in \mathbf{l}$ and each right interval bound $r_j \in \mathbf{r}$ for the event f_i is defined as $l_j = min\{s_i | \mathbf{s} \in m_i\}$ and $r_j = max\{s_i | \mathbf{s} \in m_i\}$ and computed using HyPro.

After finding the next minimum event, a new parametric location is being generated with a marking that changed according to the event that triggered the change. The marking is updated and the procedure is repeated recursively on the newly created parametric location.

4.2 Running example

Figure 3 shows the PLT for the running example using $\tau_{\max} = 10$. In general four different events are possible: On the one hand,the model allows the general transitions $\mathbf{G_1}$ and $\mathbf{G_D}$ to each fire once, represented by labels T_1 and T_2, resulting in a model with two random variables. On the other hand the continuous place $\mathbf{T}$ can reach its upper boundary (T_3) or its lower bound (T_4).

Starting in the root node (1) the minimum next event happens at time $t = 5$ when the continuous place T reaches its upper boundary, represented by node (4). However, both general transitions G_1 and

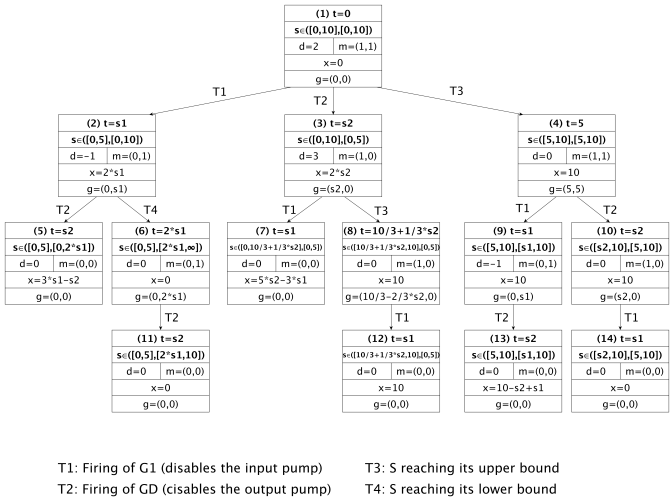

Figure 3: The Parametric Location Tree created from the running example given in Section 2.

G_D are enabled and possibly fire before time 5, leading to parametric locations (2) and (3), respectively. For each of the three successors of the root the procedure is called recursively and the bounds on $\mathbf{s}$ are adapted for each parametric location to ensure the order of events. The latter is done as follows:

Recall that before a general transition has fired, the intervals are scalars and do not depend on $\mathbf{s}$, after the firing of one or more general transitions, the bounds of $\mathbf{s}$ may linearly depend on those firing times. Hence, as shown in Figure 3, (4) is defined for $\mathbf{s} \in ([5, 10], [5, 10])$, allowing both general transitions to take values larger or equal to 5. The siblings (2) and (3) are defined for $\mathbf{s} \in ([0, 5], [0, 10])$ and $\mathbf{s} \in ([0, 10], [0, 5])$, respectively. These intervals are computed using halfspace intersection and vertex enumeration in HyPro.

Note that events T_1 and T_2 lead to a similar system evolution in this example, with the only difference that one will possibly lead to an empty tank (represented by parametric location (6)) and the other to a possibly full one (as in (8)).

In node (4) either event T_1 or T_2 occurs next, since no deterministic event is scheduled. Event T_1 leads to node (9) for values of $\mathbf{s} \in ([5, 10], [s_1, 10])$ and T_2 leads to location (10) for $\mathbf{s} \in ([s_2, 10], [5, 10])$. If T_1 happens first, i.e. the production stops and place T possibly reaches its lower bound before the demand stops. As we have chosen $\tau_{\max} = 10$, this event is not considered in the current execution. Hence, node (9) has only one successor, i.e., node (13). In node (10) only T_1 can happen, as the place is at its upper boundary and the output pump is already disabled.

5 ANALYSIS OF HPNGS USING HYPRO

We propose an analysis method for HPnGs that combines the tree-based approach of parametric reachability analysis with the geometric representation of stochastic time diagrams. Note that the implementation of the presented algorithm allows the analysis of multiple general transition firings. Using the library HyPro [22], we

construct a geometric representation for each parametric location, which corresponds to a region in an STD.

The concept of regions is repeated in Section 5.1. The transformation of locations into regions is shown in Section 5.2 and the computation of validity intervals is discussed in Section 5.3. Finally Section 5.4 computes the regions for the running example.

5.1 The concept of regions

Recall, that vector $\mathbf{s} = (s_1, ..., s_n)$ represents the n firing times of the general transitions that may occur or may become newly enabled before $\tau_{\max}$. As the geometric representation also includes time, this results in an $n+1$-dimensional representation of the region. Following [10], we define a region as a maximal set of states, that while no event occurs remains unchanged.

Definition 5.1. A region reg is a maximal connected set of $(\mathbf{s}, t)$ points, for which:

$$\forall(\mathbf{s_1}, t_1), (\mathbf{s_2}, t_2) \in reg \begin{cases} \Gamma(\mathbf{s_1}, t_1).\mathbf{m} = \Gamma(\mathbf{s_2}, t_2).\mathbf{m} \\ \Gamma(\mathbf{s_1}, t_1).\mathbf{d} = \Gamma(\mathbf{s_2}, t_2).\mathbf{d} \\ \Gamma(\mathbf{s_1}, t_1).\mathbf{g} = \Gamma(\mathbf{s_2}, t_2).\mathbf{g} \end{cases}$$

Hence within a region, the initial marking $\Gamma(\mathbf{s}, t).\mathbf{m}$, the drift $\Gamma(\mathbf{s}, t).\mathbf{d}$ and the vector of general transition firings $\Gamma(\mathbf{s}, t).\mathbf{g}$ do not change. As shown in [9] the amount of fluid and the clock valuations are linear equations of $\mathbf{s}$ and t. Hence the boundaries between regions, which represent the occurrence of an event, are likewise characterized by linear functions of $\mathbf{s}$ and t and thus represent a *halfspace* in n dimensions. Due to the linearity of all events, these halfspaces are convex polyhedra, for a proof, we refer to [9].

5.2 Transforming locations into regions

In the following, we explain how HyPro [22] is used to create the polyhedra representing each parametric location as a region.

Algorithm 1 createHalfspaceFromEvent(*event*, *createInverse*)

```
1: direction = createVector(event.getGeneralDependencies())
2: hsp = halfspace(direction, event.getTime())
3: if (createInverse is true) then
4:     return -hsp
5: end if
6: return hsp
```

Algorithm 1 turns an event into a halfspace based on its time of occurrence. This is done using a *direction* vector representation (line 1), where the scalar offset is represented by *event*.getTime() (line 2). Thus each halfspace corresponds to one element of the $\mathcal{H}$-polytope. By default, the halfspace vector points upwards in the direction of t. However, if parameter *createInverse* is true, the algorithm returns the inverse halfspace (line 4). This is needed in Algorithm 2, which creates a region based on halfspace intersection of multiple events.

Starting from the parametric location tree, a region is formed based on the entry time of that specific location, which corresponds to the occurrence time of the event leading to that parametric location. In the following this event is called source event. The region is then further bound in the direction of t by the entry times of the children of that location. The events leading to the children of the considered node are called destination events in the following.

Algorithm 2 createRegion(*baseRegion*, *sourceEvent*, *destEvents*)

```
1: region = baseRegion
2: region.insert(createHalfspaceFromEvent(sourceEvent, true))
3: for (Event e in destEvents) do
4:     region.insert(createHalfspaceFromEvent(e, false))
5: end for
6: return region
```

To form the region the source event's halfspace (line 2) as well as one halfspace corresponding to each child node (line 3+4) are created. The halfspaces corresponding to destination events are not inverted and the one that corresponds to the source event is inverted, to obtain a representation of the region that is enclosed by those halfspaces. This iterative process is illustrated in Figure 4 for the root of the parametric location tree. Its entry time ($t = 0$) corresponds to the orange halfspace and inserting the halfspaces $t = s_1$, $t = s_2$ and $t = 5$ corresponding to the destination events leads to the region as presented in step (4).

The *baseRegion* that is initialized in line 1 of the algorithm forms an $n+1$-dimensional polyhedron that contains all possible values of $\mathbf{s}$ and the time bounds $t = 0$ and $t = \tau_{\max}$, as defined in the corresponding parametric location. As such, it provides a context for inserting the halfspaces that represent the occurrence time of the source and destination events.

Algorithm 3 createSTD(*node*)

```
1: baseRegion = region(node)
2: childNodes = getChildNodes(node)
3: region = createRegion(baseRegion, node.getSourceEvent(), getSourceEvents(childNodes))
4: node.setRegion(region)
5: for nodes n in childnodes do
6:     createSTD(n)
7: end for
```

While Algorithms 1 and 2 allow to compute a geometric representation of each parametric location, it is also possible to compute the complete STD (up to some maximum analysis time). This is done using a depth-first visit of the parametric location tree as presented in Algorithm 3. The algorithm creates a region for each location (lines 2-4) and recursively visits each node of the tree (lines 5-7). Note that in contrast to [10], the presented approach allows for parallelization, as each region can be created independently.

5.3 Computing validity intervals

We want to identify the so-called validity intervals, which specify those values of $\mathbf{s}$ that lead to a system state that satisfies a certain property at time t. Hence, to compute the probability to be in a specific system state at time τ, one would perform a multidimensional integration over those intervals, which are subsets of the crossproduct of the domains of all random variables present in the system. We want to identify all regions in which the system can be at time $t = \tau$ and that fulfill a certain property, which can be defined following [12]:

$$\psi = tt \quad | \quad \neg\psi \quad | \quad \psi \wedge \psi \quad | \quad m_i = a \quad | \quad x_k \leq b,$$

where m_i denotes the number of tokens in the discrete place $P_i \in \mathcal{P}^D$ and x_k is the fluid level in the continuous place $P_k \in \mathcal{P}^C$.

Algorithm 4 validityIntervals(*pltree*, ψ, *time*)

```
1: candidates = pltree.getCandidateLocationsForTime(time)
2: for (Node c in candidates) do
3:     baseRegion = region(c)
4:     childNodes = getChildNodes(c)
5:     region = createRegion(baseRegion, c.getSourceEvent(), getSourceEvents(childNodes))
6:     c.setRegion(region)
7: end for
8: parse(candidates, ψ, time)
9: return intervals
```

Algorithm 5 parse(*candidates*, ψ, *time*)

```
1: intervals = vector()
2: if ψ = tt then
3:     return S_0
4: else if ψ = ψ_1 ∧ ψ_2 then
5:     return parse(candidates, ψ_1, time) ∩ parse(candidates, ψ_2, time)
6: else if ψ = ¬ψ then
7:     return S_0 \ parse(candidates, ψ, time)
8: else if ψ is atomic property then
9:     return intervalSetsAtTime(candidates, ψ, time)
10: end if
11: return intervals
```

Algorithm 6 intervalSetsAtTime(*candidates*, ψ, *time*)

```
1: intervals = vector()
2: for (Node c in candidates) do
3:     if ψ.isContinuous() then
4:         fillLevelHsp = createFillLevelHsp(c, ψ)
5:         region = c.getRegion().insert(fillLevelHsp)
6:     else if !c.discreteMarking == ψ.value then
7:         return 0
8:     end if
9:     timeHyperplane = createTimeHyperplane(time)
10:    i = region.getIntersectionPoints(timeHyperplane)
11:    intervals.insert(i)
12: end for
13: return intervals
```

For a specific parametric location tree, formula ψ and predefined analysis time, Algorithm 4 computes the validity intervals, by first collecting all locations from the parametric location tree in which the system can be at time τ (line 1) as candidates. The validity intervals are computed using the geometric representation of the candidates, hence for each of them the child nodes are identified and the corresponding region is created (line 2-7). Then Algorithm 5 is called to parse formula ψ (line 8) and for each type of formula the corresponding computation is called, recursively.

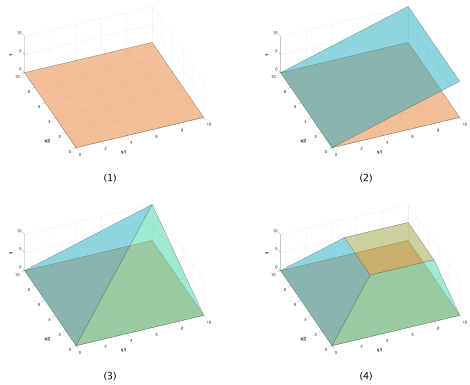

Figure 4: Halfspace intersection used to create root region.

In case of tt, the algorithm returns S_0, which has been defined as the n-dimensional vector $([0, \tau_{\max}], \ldots, [0, \tau_{\max}])$. In case of a negation and conjunction the algorithm is called recursively for the respective subformulas and the results are subtracted from S and intersected, respectively. For a discrete ($m_i = a$) or continuous ($x_k \leq b$) atomic property, Algorithm 6 is called to perform the geometric operations on the regions. When dealing with a continuous property ($x_k \leq b$), recall that the value of the continuous variable x is a linear function of the vector **s**. Hence we construct a hyperplane representing the value of the continuous variable x (line 4) and intersect it with each candidate region (line 5). The resulting halfspace fulfills the continuous property in all points. A discrete property either holds in the entire region or not at all. Hence, it suffices to check whether the desired property holds somewhere in the region (line 6). Then we call HyPro to identify the validity interval(s) by intersecting the (remaining) polyhedron with τ and projecting the result onto the domains of the random variables (line 9-10). This n-dimensional interval is then added to the resulting set of validity intervals (line 11) and returned by the algorithm.

5.4 Running example

Figure 5 illustrates the method of finding validity intervals for the formula $\psi = x_0 \leq 1$ at time $t = 2$. First, seven locations are identified as *candidates* in the PLT, i.e., locations (1), (2), (3), (5), (6), (7) and (11). Their graphical representation is provided in Figure 5 (1). Then for each of these candidates the given property has to be checked. Following Algorithm 6 we construct a halfspace representing the event that the place reaches the desired continuous level. This is the blue halfspace in part 5 of the figure and it is then intersected with all candidate regions. For one candidate, i.e. the halfspace corresponding to location (2), the result of this intersection is shown in Figure 5 (3). Note that the part of the green region which lies behind the blue halfspace is cut off. Finally, we project the part of the remaining region in which the system can be in at time τ on the domains of s_1 and s_2. The resulting validity interval which fulfills the property is shown in (4). Using the bounding box functionality in HyPro then allows to identify the vertices that would be needed to perform a multi-dimensional integration over all random variables. Running on a Intel Core i7 2.5GHz with 16GB

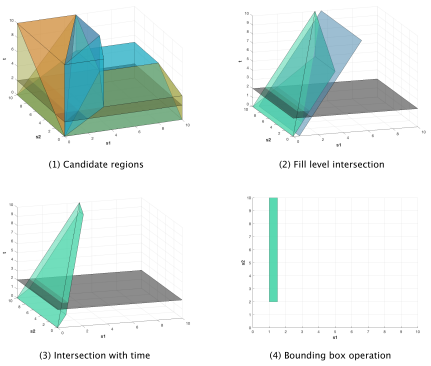

Figure 5: Computing intervals for s_1 and s_2 testing that the continuous place remains below a value of 1 at time $t = 2$.

of Memory, based on a given PLT the STD is create in 50ms. The satisfaction intervals are computed in another 20ms.

6 CONCLUSIONS AND FUTURE WORK

This paper extends the algorithm that has previously been proposed to construct a parametric location tree to HPnGs with multiple stochastic firings. The main contribution of this paper is the transformation of parametric locations into their corresponding graphical representation as regions. A region corresponds to a polyhedron and is constructed using the library HyPro. Furthermore, we present algorithms to compute the validity sets of simple nested formula at a certain time t. The presented approach allows the efficient and exact analysis of HPnGs with multiple stochastic firings, as the graphical representation of nodes is only induced for so-called candidate, i.e., those nodes of the PLT that the system can be in at time t. We expect that this will considerably speed up the analysis, as it prevents the costly graphical representation of the complete state space and also allows for parallelization. Note that the approach is still able to compute the exact validity intervals, using the graphical representation of the candidate regions.

The main advantage of the proposed method is that it allows HPnGs with a large number of multiple stochastic firings, which was not possible with previous approaches. This has been made possible by the library HyPro that allows for a $\mathcal{H}$-representation of multi-dimensional convex polyhedra and their construction using half space intersection. Note that half space arrangement, which was previously required to construct the full STD is no longer necessary. As an added benefit, the proposed approach is able to deal with HPnGs, where several timed transitions are scheduled to fire at the same time, which was previously excluded for the analysis using STDs. While currently, this intrinsic non-determinism in the firing behavior is resolved probabilistically, as is often done in Petri nets, in future work, we will strive to provide min/max schedulers to resolve the nondeterminism.

Furthermore, we plan to provide a direct transformation of the parametric location tree into a hybrid automaton, that hides the stochastic behavior in a global variable **s**. This would allow using existing tools for the analysis of hybrid systems to determine the set of reachable states. Since the continuous behavior is limited to constant derivatives, we could use e.g. PHAVer [5] to compute the set of reachable states. When extending the dynamics of continuous transitions to (non)-linear ODEs, existing analysis tool, e.g. SpaceEx [6] or Flow* [4]) could be used via a proper transformation.

In addition, we plan to construct a larger case study in future work to gather run time information and compare the complexity of the new approach to the existing algorithms.

REFERENCES

[1] H. Alla and R. David. 1998. Continuous and hybrid Petri nets. *Journal of Circuits, Systems, and Computers* 8, 01 (1998), 159–188.
[2] R. Bagnara, P. M. Hill, and E. Zaffanella. 2008. 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 (2008), 3–21.
[3] C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. 1996. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)* 22, 4 (1996), 469–483.
[4] X. Chen, E. Ábrahám, and S. Sankaranarayanan. 2013. Flow*: An analyzer for non-linear hybrid systems. In *Int. Conf. on Computer Aided Verification (CAV) (LNCS)*, Vol. 8044. Springer, 258–263.
[5] G. Frehse. 2005. PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech. In *Hybrid Systems: Computation and Control (HSCC 2005) (LNCS)*, Vol. 3414. Springer, 258–273.
[6] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. 2011. SpaceEx: Scalable Verification of Hybrid Systems. In *23rd Int. Conf. on Computer Aided Verification (CAV), 2011 (LNCS)*, Vol. 6806. Springer, 379–395.
[7] K. Fukuda. 2017. Cdd library website. (2017). https://www.inf.ethz.ch/personal/fukudak/cdd_home/
[8] E. Gawrilow and M. Joswig. 2000. Polymake: a framework for analyzing convex polytopes. In *Polytopes-Combinatorics and Computation*. Springer, 43–73.
[9] H. Ghasemieh. 2017. *Analysis of hybrid Petri nets with random discrete events*. Ph.D. Dissertation. University of Twente.
[10] H. Ghasemieh, A. Remke, B. Haverkort, and M. Gribaudo. 2012. 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 (FORMATS), 2012*. Springer, 139–154.
[11] H. Ghasemieh, A. Remke, and B. R. Haverkort. 2013. Analysis of a sewage treatment facility using hybrid Petri nets. In *7th Int. Conf. on Performance Evaluation Methodologies and Tools (VALUETOOLS), 2013*. ICST, 165–174.
[12] H. Ghasemieh, A. Remke, and B. R. Haverkort. 2014. Hybrid Petri nets with multiple stochastic transition firings. In *8th Int. Conf. on Performance Evaluation Methodologies and Tools (VALUETOOLS), 2014*. ICST, 217–224.
[13] A. Girard. 2005. Reachability of Uncertain Linear Systems Using Zonotopes. In *Hybrid Systems: Computation and Control (HSCC), 2005*, Vol. 3414. 291–305.
[14] A. Godde. 2017. Translating Model Checking of Hybrid Petri Nets into Operations on Nef Polyhedra. (2017).
[15] M. Gribaudo and Remke A. 2010. Hybrid Petri Nets with General One-Shot Transitions for Dependability Evaluation of Fluid Critical Infrastructures. In *IEEE 12th Int. Symp. on High-Assurance Systems Engineering (HASE 2010)*. IEEE, 84–93.
[16] M. Gribaudo and A. Remke. 2016. Hybrid Petri nets with general one-shot transitions. *Performance Evaluation* 105 (2016), 22–50.
[17] J. Hüls and A. Remke. 2016. Coordinated charging strategies for plugin electric vehicles to ensure a robust charging process. In *10th Int. Conf. on Performance Evaluation Methodologies and Tools (VALUETOOLS)*.
[18] J. Hüls and A. Remke. 2016. Energy storage in Smart Homes: Grid-convenience versus self-use and survivability. In *24th Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 385–390.
[19] C. Le Guernic. 2009. *Reachability analysis of hybrid systems with linear continuous dynamics*. Ph.D. Dissertation. Université Joseph-Fourier-Grenoble I.
[20] C. Le Guernic and A. Girard. 2010. Reachability Analysis of Linear Systems using Support Functions. *Nonlinear Analysis: Hybrid Systems* 4, 2 (2010), 250–262.
[21] R. E. Moore, R. B. Kearfott, and M. J. Cloud. 2009. *Introduction to interval analysis*. SIAM.
[22] S. Schupp, E. Ábrahám, I. B. Makhlouf, and S. Kowalewski. 2017. HyPro: A C++ Library of State Set Representations for Hybrid Systems Reachability Analysis. In *NASA Formal Methods - 9th Int. Symp. (NFM) 2017 (LNCS)*, Vol. 10227. Springer, 288–294.
[23] The CGAL Project. 2017. *CGAL User and Reference Manual* (4.10 ed.). CGAL Editorial Board. http://doc.cgal.org/4.10/Manual/packages.html
[24] G. M. Ziegler. 2012. *Lectures on polytopes*. Vol. 152. Springer Science & Business Media.