The present work was submitted to the LuFG Theory of Hybrid Systems

MASTER OF SCIENCE THESIS

COMPOSITIONAL MODELING OF STOCHASTIC HYBRID SYSTEMS

Carolina Gerlach

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr. Ir. Dr. h. c. Joost-Pieter Katoen

Additional Advisor:
Sasan Vakili

Aachen, July 26, 2022

Abstract

The behavior of a hybrid system can be described by continuous flows interleaved with discrete changes. Hybrid systems are often affected by uncertainties, which can be modeled via stochastic hybrid systems. In this thesis, we propose a modeling language for stochastic hybrid systems and discuss our design choices. We define stochastic hybrid automata that extend hybrid automata by spontaneous jumps. Additionally, the jumps are assigned weights to make a probabilistic choice between several jumps that can be taken at the same time. We expect our model to be extensible with other sources of stochasticity regarding, e.g., resets, initial states, or continuous dynamics.

Our modeling language is designed with respect to two goals that we found to be missing from existing approaches for stochastic hybrid automata, while additionally trying to keep the model as expressive as possible. Firstly, we aim to define a compositional model allowing for communication between different components via shared variables. Secondly, we intend to enable easy modeling of competing stochastic events, in particular of stochastically independent competing events. To this end, we associate each stochastic event with a set of jumps. Each stochastic event has a state-dependent probability distribution determining when some associated jump should be taken.

Erklärung

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Carolina Gerlach
Aachen, den 26. Juli 2022

Contents

Chapter 1

Introduction

Hybrid systems combine discrete and continuous behavior. Alur et al. [ACH$^+$95] proposed to model them as *hybrid automata*, which can be viewed as discrete transition systems extended by continuous dynamics. The behavior of a hybrid automaton is governed by instantaneous *jumps* between discrete control *locations* and continuous evolution of real-valued *variables* while control stays in some location. For example, a thermostat can be modeled by a continuous variable representing the temperature and transitions between discrete locations representing the heater being on or off [ACH$^+$95]. If the heater is on (off), the temperature increases (decreases), and once a certain threshold is reached, the heater is switched off (on).

Hybrid systems are often affected by uncertainty, for example, with regard to when control transitions from one location to another (*jump times*) and which jump is taken or how variables are affected by a jump (*resets*). Similarly, the continuous evolution of variables in the locations may also be affected by uncertainty. In real-life applications, these choices often follow certain probability distributions. For example, the heater from the example above might break with a certain probability, which increases over time. Hence, determining the point in time when control transitions to some "broken" location via an appropriate probability distribution yields a more accurate model than a non-deterministic choice. For example, determining the probability of some undesirable or unsafe outcome would be more useful than only knowing whether it could happen.

Stochastic hybrid systems extend hybrid systems by stochastic behavior. Overviews of continuous-time models of stochastic hybrid systems, including, for example, *piecewise-deterministic Markov processes*, *switching diffusion processes*, and *stochastic hybrid automata*, are provided in [CL07, TSS14, PBLD03]. We focus on continuous-time models here and refer to [LSAZ21] for discrete-time models. Models for stochastic hybrid systems differ with respect to whether they offer stochastic continuous dynamics, stochastic discrete dynamics, or a combination of both, and how this stochasticity is offered. Figure 1.1 provides an overview of features offered by different formalisms, which we elaborate on later in this chapter.

In this thesis, we focus on stochastic hybrid automata, which are sometimes also called stochastic hybrid systems. This term is used to refer to several different modeling languages extending hybrid automata with stochastic behavior in different ways. We propose a modeling language targeting two aspects regarding compositional modeling and the modeling of stochastic events, which we found to be missing from previ-

	[BL04]	[BSA04]	[HLS00]	[Hes07]	[AHS13]	This work
SDE	✓	✓	✓	✓	✗	✗
Spontaneous jumps	✓	✗	✗	✓	✓	✓
Forced jumps	✓	✓	✓	✗	✗	✗
Stoch. variable resets	✓	✓	✓	✗	✗	✗
Stoch. jump choice	✓	✓	✓	✗	✗	✓
Stoch. initial states	✓	✓	✗	✗	✗	✗

Figure 1.1: Overview of features offered by several different stochastic hybrid system modeling formalisms.

ous models. Additionally, we aim for our model to be as expressive as possible. Our modeling language is based on previous Bachelor's theses by Scherer [Sch22], Hua [Hua21], and Appenzeller [App21]. In the following, we elaborate on our modeling goals and give an overview of existing modeling languages for each aspect.

Expressivity Firstly, we intend to design a model that is as expressive as possible. Models for stochastic hybrid systems differ vastly in terms of the offered stochasticity, as can be seen in Figure 1.1.

For the continuous dynamics, [HLS00, BSA04, GAM97] offer *stochastic differential equations* (*SDEs*), while [FHH$^+$11] only allows differential equations and [AHS13] even restricts this to linear differential equations. Other approaches like [Hes14, NHR21] specify *ordinary differential equations* (*ODEs*) via (locally) Lipschitz continuous functions, which implies the existence of a unique maximal (but not necessarily global) solution of the corresponding ODE for each initial value problem [Mar04].

The discrete dynamics can be extended by stochastic behavior in different ways. Some models offer *forced* jumps, which are taken if the boundary of the location *invariant* is hit [Dav84, HLS00]. In contrast, the jump times of *spontaneous* jumps are determined by probability distributions, which may depend on the location and the jump [AHS13, Hes14], on the *state* - consisting of the location and the current values of the variables - [BBR$^+$20], or on the state and jump [Hes07]. We discuss these different possibilities in more detail when elaborating on how to model stochastic events later in this chapter. Spontaneous jumps also include the special case of *urgent* jumps, which have to be taken as soon as they are enabled, as, for example, in [HHHK13].

Some models offer stochastic choices over jumps (sometimes simply in the form of a stochastic choice of the next location) and resets of continuous variables after jumps [FHH$^+$11], others only over resets [HLS00], or neither [Hes07].

Initial locations and *initial variable valuations* are also sometimes chosen stochastically [BL04, BSA04].

Some models impose restrictions on the predicates describing when jumps can be taken (*guards*) or under which conditions time can pass in locations (location invariants). For example, [HLS00] requests invariants to be open sets and [Gbu18] defines stochastic rectangular hybrid systems that only allow convex guards, and only convex flows as well.

In this thesis, we support ODE-based flow specifications and guarded spontaneous jumps. More specifically, we introduce stochasticity with respect to the jump times and the choice between jumps. We also discuss possible extensions with other sources of stochasticity, which should mostly be rather straightforward (see Chapter 4). In order to define a meaningful compositional modeling language, we restrict how components can influence each other by imposing some constraints on variable resets, location invariants, and initial states.

Compositionality Modeling a (stochastic) hybrid system as a single (stochastic) hybrid automaton becomes more difficult with increasing complexity and size. Instead, it is more convenient to model such systems by several *components*, which might communicate via shared variables or jump synchronization, such that their composition describes the behavior of the complete system. In the following, we give a short overview of different compositional modeling languages.

Bouyer et al. [BBCM16] define the parallel composition of a subclass of stochastic *timed* automata [1]. Their proposed stochastic timed automata include state-dependent stochastic choices over edges and jump times. The composition operator corresponds to the interleaving of stochastic timed automata and allows neither jump synchronization nor shared variables. The authors argue that only a subclass of stochastic timed automata can be safely composed such that their composition reflects a race between the components with respect to which component may take a jump next. This illustrates that the composition of stochastic hybrid systems is not necessarily trivial.

Communication via jump synchronization is offered, for example, by the models defined in [DDL+12], [CL07, Chapter 3] and [HHHK13].

David et al. [DDL+12] give stochastic semantics for networks of hybrid automata, whose jumps are labeled with *input* and *output* actions. Stochasticity is included in the form of state-dependent stochastic choices over jump times, outputs, and states reached by performing an action. The component sampling the smallest jump time broadcasts a sampled output action while all other components synchronize by performing a corresponding input action. David et al. ensure that such synchronization is always possible by only considering *input-enabled* hybrid automata, i.e., it must be possible to take each input action at each state.

Strubbe and van der Schaft [CL07, Chapter 3] propose communicating piecewise-deterministic Markov processes as a compositional modeling framework for stochastic hybrid systems, which includes stochastic as well as non-stochastic jumps. At each location, the outgoing stochastic jumps are in a race with each other with respect to which one will be taken, but they are not allowed to synchronize with other stochastic or non-stochastic jumps. Components can, however, synchronize on specified non-stochastic jumps in different ways: *Active* jumps can either be required to synchronize with other active jumps, or trigger the execution of *passive* jumps.

HModest [HHHK13] is a compositional modeling framework for stochastic hybrid systems. The symbolic semantics of a HModest process is given via stochastic hybrid automata, where the locations correspond to process behaviors and the edges are defined by inference rules. HModest allows for process synchronization via *patient* and *impatient* actions, similar to the distinction between passive and active jumps in [CL07, Chapter 3]. Additionally, processes may communicate via global variables. We emphasize that although the HModest framework is compositional, this work does not define compositional stochastic hybrid automata.

[1]In a timed automaton, all variables must evolve linearly (i.e., with derivative 1) and may only be reset to 0. They are called *clocks*.

We found only very few modeling languages for stochastic hybrid systems that allow communication via shared variables and therefore chose this as one of our design goals in order to fill this niche. Two approaches for compositional modeling using shared variables are presented in [TEF11] and [BSA04].

For discrete time, Teige et al. [TEF11] define concurrent probabilistic hybrid automata allowing for probabilistic branching. Each component distinguishes between local variables and global variables controlled by other components. A component may write non-local variables in resets and guards may refer to global variables and the current locations of other components. In addition to communication via the variables, components synchronize on transitions by trying to find compatible local transitions. If the variable resets proposed by different components are inconsistent, the system deadlocks. Teige et al. explicitly prefer this to avoiding such conflicts by restricting the use of (global) variables.

For continuous time, Bernadsky et al. [BSA04] propose a modeling language for concurrent stochastic hybrid systems, which allows *agents* (similar to hybrid automata) to communicate via shared variables, with more fine-grained read/write accesses. Each agent controls *private* and *observable* variables and can additionally read *external* variables, which correspond to variables declared observable by other agents. This model offers specification of continuous dynamics via SDE and forced jumps with stochastic choices over target locations and variable resets. We are not aware of any other modeling language for continuous-time stochastic hybrid systems that allows for communication via shared variables.

Since the influence of external variables can cause instantaneous violation of location invariants, Bernadsky et al. allow to leave a location exactly when the invariant is violated. We prefer not to weaken the meaning of invariants in this way and therefore propose a different modeling approach for shared variables, which restricts the influence of external/non-controlled variables. We discuss differences between their model and our model in more detail in Section 3.4.

In summary, this thesis focuses on communication between components via shared variables, distinguishing between variables controlled by a component itself and other components. In Chapter 4, we discuss how our model could be extended with jump synchronization.

Modeling Stochastic Events We want to design our model in such a way that it is easy to model "competing" stochastic events. For example, the thermostat from the introductory example might break/fail due to various reasons, like the wear of different components or a power outage. So in a sense, there are different stochastic events that compete with regard to which one will happen first. For example, the probability of one event might increase over time, while the probability distribution associated with another event might assign the same probability to all possible points in time. There are several possibilities how these stochastic events can be modeled.

Many approaches associate one stochastic event with each location, such that one "global" jump time is determined per location and the decision about which jump is taken only happens at jump time [BBB$^+$14, BBR$^+$20, DDL$^+$12]. This approach is advantageous for verification, but impractical for our goal of modeling stochastic events that compete at each location, since the modeler then has to "merge" several stochastic events into one event for each location.

It would be more natural to associate the stochastic events with the jumps instead of the locations. Then, at each location, the outgoing jumps are in a race with each

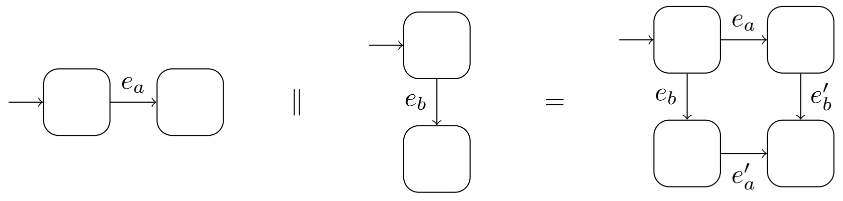


Figure 1.2: Illustration of the composition of two stochastic hybrid automata.

other with respect to which will be triggered first. This approach is, for example, taken by Hespanha in [Hes07], where the jump times are determined by probability distributions depending on the jump and state.

However, if each stochastic event is only allowed to be associated with one jump, then the modeling of stochastically independent events becomes problematic. Consider, for example, the composition of two stochastic hybrid automata as illustrated in Figure 1.2. Both components have a single jump associated with stochastic events a and b, respectively, which we assume to be stochastically independent. If e_a is triggered in the composition before e_b is triggered, then after taking e_a, we need a copy e'_b of e_b to model the continuation of stochastic event b. However, if each stochastic event may only be associated with a single jump, then we cannot assign b to e'_b. Hence, we would have to sample a new jump time instead of continuing with the jump time sampled for e_b. As a result, we could only model the continuation of memoryless stochastic events this way. In the case of continuous probability distributions, this would mainly allow the exponential distribution [Fel91].

In this thesis, we allow several jumps to be associated with the same stochastic event. A similar mechanism could be implemented using the singular automata with random clocks defined by Pilch et al. [PKRA20]. We compare the approaches in further detail in Section 3.4. We are not aware of any other modeling formalism for stochastic hybrid systems allowing to bind together several jumps to model the same stochastic event.

In summary, the main contributions of this thesis are the following:

1. We present a modeling language for stochastic hybrid systems in the form of stochastic hybrid automata with spontaneous jumps and stochastic choice between jumps, which should be easily extensible to other sources of stochasticity.

2. Our modeling language is compositional in the sense that components can communicate via shared variables.

3. Our language allows to model competing stochastically independent events.

4. We discuss our design choices regarding all mentioned aspects.

This thesis is structured as follows. We start by adapting the hybrid automata defined by Alur et al. [ACH$^+$95] to our needs in Chapter 2 and defining the syntactic parallel composition of hybrid automata. In Chapter 3, we then extend hybrid automata by stochastic behavior, define the syntactic parallel composition of these stochastic hybrid automata, and compare our modeling language to other approaches. In Chapter 4, we discuss possible extensions of our model to include, for example, other sources of stochastic behavior, forced jumps, or jump synchronization. Lastly, we summarize our model and outline possible future work in Chapter 5.

Notation We use $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, $\mathbb{N}$ and $\mathbb{N}_{>0}$ to denote the sets of all real numbers and non-negative real numbers, as well as the natural numbers with and without 0, respectively. For sets A and B with $a \in A$, $b \in B$, and a function $f\colon A \to B$, we use $f[a \mapsto b]$ to denote $g\colon A \to B$ with $g(a) = b$ and $g(a') = f(a')$ for all $a' \in A \setminus \{a\}$. Furthermore, for $A' \subseteq A$ we define $f|_{A'}\colon A' \to B$ with $f|_{A'}(a) = f(a)$ for all $a \in A'$. For sets A, B and C, we use $C = A \mathbin{\dot\cup} B$ to state that $A \cap B = \emptyset$ and $C = A \cup B$.

Chapter 2

Hybrid Automata

Hybrid automata extend discrete transition systems by continuous dynamics. We refer to the discrete states and transitions of the transition system as *locations* and *jumps*, respectively. The continuous behavior is modeled by real-valued *variables* that evolve over time. The values of the variables in some variable set V are specified by functions $\nu\colon V \to \mathbb{R}$, which we call *valuations*. Valuations are sometimes also interpreted as vectors in $\mathbb{R}^n$, given a fixed order of the variables V with $|V| = n$. We often denote a valuation $\nu\colon \{x_1, \dots, x_n\} \to \mathbb{R}$ by a tuple $(x_1 \mapsto \nu(x_1), \dots, x_n \mapsto \nu(x_n))$. The set of all valuations for V is denoted by $\mathcal{V}_V$, or simply $\mathcal{V}$ if the variable set is clear from the context. The evolution of variables over time is described by *activities*, which are continuous functions $f\colon \mathbb{R}_{\geq 0} \to \mathcal{V}_V$. The set of all activities for V is denoted by F_V, or simply F.

The combined discrete-continuous behavior of a hybrid automaton is governed by instantaneous jumps between the locations and continuous evolution of the variables while control stays in some location. The jumps are *guarded* by conditions on the values of the variables and may *reset* variables to other values. Each location is associated with a set of activities, defining how the variables may evolve while control is in this location. Control may only stay in a location as long as the location's *invariant* is satisfied, which imposes restrictions on the variable values. Each location l is additionally assigned a set of *initial valuations*, which acts as an initial condition on the variable values if l is chosen as the initial location.

Our goal is to define a compositional modeling language for (stochastic) hybrid automata that allows communication between different components via shared variables. To avoid conflict between components, we allow each variable to be *controlled* by exactly one component, which means that only this component can determine the evolution of the variable's values and reset its values via jumps. In turn, each component models the behavior of its environment non-deterministically and thus has to allow for all possible evolution of the *non-controlled* variables. The behavior of a component may be influenced by variables it does not control. For example, the evolution of a controlled variable might depend on the evolution of a non-controlled variable, or a jump might be guarded by a condition on some non-controlled variable.

The seminal model by Alur et al. [ACH+95] also defined the notion of controlled variables, but per location instead of per component. Associating variables with components can be considered a special case of associating them with locations. Two hybrid automata can be composed if they never try to control the same variable. If the variables are controlled by the components, this can be easily decided by checking whether the intersection of their variable sets is empty. If the variables are controlled by the locations, we could also simply check whether the set of variables controlled by any location of one component is disjoint from the corresponding set for the other component. However, we are actually interested in whether, during the parallel execution of the two hybrid automata, we ever reach a point where the current locations of both components want to control the same variable. This is a reachability problem, which is in general undecidable for hybrid automata [HKPV98]. Since our goal is to model communication between components, we choose to associate variables with components here, but allowing a more fine-grained control by associating them with locations might also be possible.

A hybrid automaton synchronizes with its environment by taking a matching *environmental* step whenever another component takes a jump updating the variables. Alur et al. [ACH+95] defined these environmental steps as syntactical τ-jumps that only allow changes of the non-controlled variables, thus restricting the write access to the controlled variables for each location. Since adding these τ-jumps for all possible changes of the environment is quite laborious for the modeler, we instead add such environmental steps semantically, which may only change variables controlled by other components. Note that, in contrast to Alur et al., we do not allow jumps to synchronize via labels. Our goal is to model stochastic hybrid systems and we will see in Section 4.3 that synchronization of spontaneous jumps is not straightforward.

Even though the behavior of a hybrid automaton may depend on variables controlled by other components, we will see that these dependencies should not be allowed everywhere. We say that a valuation set is *closed* with respect to the non-controlled variables if it is not influenced by them. In the following definition we formally introduce the concept of valuation sets being closed with respect to some set of variables. To ease notation throughout this thesis, we also first define an equivalence relation expressing that two valuations agree on a set of variables.

Definition 2.0.1 (Valuation Set Closure w.r.t. a Variable Set). *Let V^+, V^- be two disjoint variable sets and $V = V^- \uplus V^+$.*

For all ν,$\nu' \in \mathcal{V}_V$ we define $\nu \approx_{V^+} \nu'$ iff $\nu|_{V^+} = \nu'|_{V^+}$.

*A set $S \subseteq \mathcal{V}_V$ is V^--*closed *iff $S = \{\nu' \in \mathcal{V}_V \mid \exists \nu \in S.\ \nu \approx_{V^+} \nu'\}$.*

For some set of variables $V = V^- \uplus V^+$, a set of valuations $S \subseteq \mathcal{V}_V$ is V^--closed if all valuations agreeing on V^+ with some valuation in S are also contained in S. In other words, V^- may not influence S. We will see a few examples illustrating what it means for a valuation set to be closed with respect to the non-controlled variables when we give examples for hybrid automata (see, e.g., Examples 2.1.1 to 2.1.3). In the following, we discuss the syntax and semantics of hybrid automata and define their syntactic parallel composition.

2.1 Syntax of Hybrid Automata

Following [ACH+95], we choose a more semantic approach for defining hybrid automata and specify activities and conditions on valuations as functions and sets, respectively, instead of adopting a more syntactic style using ordinary differential equations (ODEs) and predicates. However, we do specify the activity sets as solution sets of ODE systems because ODEs allow for nicer modeling of dependencies between the evolution of different variables while guaranteeing desirable properties like time-invariance. In particular, we only consider linear first-order ODE systems in order to be able to guarantee the existence of a unique global solution [Zil12]. For convenience, we often use predicates to specify guards and invariants in text and graphical representations. We will discuss our design choices in more detail after having given the syntax of hybrid automata.

Our modeling language is based on a previous Bachelor's thesis on compositional modeling of stochastic hybrid systems [Sch22]. It adapts the original framework for hybrid systems proposed by Alur et al. [ACH+95] to communication between components via shared variables.

Definition 2.1.1 (Syntax of Hybrid Automata). *A* hybrid automaton $\mathcal{H}$ *is a tuple* $(\mathit{Loc}, (\mathit{Con}, \mathit{NCon}), \mathit{Inv}, \mathit{Init}, \mathit{Edge}, \mathit{Act})$*, where*

- *Loc is a non-empty finite set of* locations*;*
- *Con and NCon are two disjoint sets of* controlled *and* non-controlled variables*, respectively; we set $V = \mathit{Con} \uplus \mathit{NCon}$, and write $\mathcal{V}$ and F to denote $\mathcal{V}_V$ and F_V, respectively;*
- *$\mathit{Inv}\colon \mathit{Loc} \to 2^{\mathcal{V}}$ assigns an* invariant *$\mathit{Inv}(l) \subseteq \mathcal{V}$ to each location $l \in \mathit{Loc}$ such that $\mathit{Inv}(l)$ is NCon-closed;*
- *$\mathit{Init}\colon \mathit{Loc} \to 2^{\mathcal{V}}$ assigns a set of* initial valuations *$\mathit{Init}(l) \subseteq \mathcal{V}$ to each location $l \in \mathit{Loc}$ such that $\mathit{Init}(l)$ is NCon-closed and $\mathit{Init}(l) \subseteq \mathit{Inv}(l)$;*
- *$\mathit{Edge} \subseteq \mathbb{N} \times \mathit{Loc} \times 2^{\mathcal{V}^2} \times \mathit{Loc}$ is a finite set of* jumps *such that for all $(id, l, \mu, l') \in \mathit{Edge}$ and all $(\nu,\nu') \in \mu$ it holds that $\nu \approx_{\mathit{NCon}} \nu'$ and $\nu' \in \mathit{Inv}(l')$;*
- *$\mathit{Act}\colon \mathit{Loc} \to 2^F$ assigns an* activity *set $\mathit{Act}(l)$ to each location $l \in \mathit{Loc}$, consisting of all $f \in \mathit{Act}(l)$ that satisfy a first-order system of linear ordinary differential equations of the form $\dot{\mathbf{x}}_{\mathit{Con}} = A_l \mathbf{x}_{\mathit{Con}} + B_l \mathbf{x}_{\mathit{NCon}} + \mathbf{c}_l$, where*
  - *$\mathbf{x}_{\mathit{Con}}, \mathbf{x}_{\mathit{NCon}}$ are vectors of the controlled and non-controlled variables, respectively, and $\dot{\mathbf{x}}_{\mathit{Con}}$ represents the first derivative of the evolution of the controlled variables,*
  - *A_l, B_l are real-valued matrices of suitable dimensions,*
  - *$\mathbf{c}_l$ is a vector of real-valued constants.*

Let in the following $\mathcal{H} = (\mathit{Loc}, (\mathit{Con}, \mathit{NCon}), \mathit{Inv}, \mathit{Init}, \mathit{Edge}, \mathit{Act})$ be a hybrid automaton. A *state* σ of $\mathcal{H}$ consists of a discrete location l and a valuation ν of the continuous variables. We use $\Sigma = \mathit{Loc} \times \mathcal{V}_V$ to denote the set of all states. We call $(l, \nu) \in \Sigma$ an *initial state* if $\nu \in \mathit{Init}(l)$. We now elaborate on the different parts of hybrid automata and the imposed conditions, before illustrating the definition with examples.

Inv If an invariant in some component depends on a variable x controlled by another component, the invariant might become violated if that component changes the value of x. Since the location invariants have to be satisfied at all times, the former component would block this variable update and would thus effectively influence the evolution of a variable that it is not supposed to control. To avoid this, we require that invariants should be *NCon*-closed. Nevertheless, the environmental behavior can indirectly lead to the invariant being violated, since the evolution of controlled variables may depend on non-controlled variables. The environment can however not cause an instantaneous violation of the invariant anymore, in contrast to the case where the invariant depends directly on a non-controlled variable.

Instead of requiring the invariants to be *NCon*-closed, we could weaken the meaning of invariants such that invariants are allowed to be violated at jump time, and only then, i.e., time may not pass while the invariant is violated. This approach is chosen in [BSA04], but we find it undesirable. Another possibility would be to explicitly add τ-jumps for all possible environmental changes, allowing a component to synchronize with any environmental updates. However, this is quite cumbersome since we have to add a jump for all states and all possible changes. Therefore, we require invariants to be *NCon*-closed.

Init Initial valuations should satisfy the location's invariant. Initial valuation sets are also required to be *NCon*-closed for technical reasons, to allow us to focus on the most crucial aspects. This does not reduce expressivity because any initial condition on a non-controlled variable can be modeled by adding it as an initial condition to the component controlling the concerned variable.

Edge The set of all jumps is called *Edge* for historical reasons, but we use the term 'jump' for elements of *Edge* to reflect the contrast between continuous evolution and instantaneous jumps.

A jump $e = (id, l, \mu, l') \in Edge$ is called *enabled* in a state $\sigma \in \Sigma$ if the current valuation satisfies the jump's guard.

Definition 2.1.2 (Guard, Enabled)**.** *Let $\mathcal{H} = (Loc, (Con, NCon), Inv, Init, Edge, Act)$ be a hybrid automaton and $e = (id, l, \mu, l') \in Edge$ some jump.*

The guard *of e is defined as $g_e = \{\nu \in \mathcal{V}_V \mid \exists \nu' \in \mathcal{V}_V.\ (\nu, \nu') \in \mu\}$.*

We call e enabled *in state $(l, \nu) \in \Sigma$ if $\nu \in g_e$.*

For each $\nu \in g_e$, there may exist several $\nu' \in \mathcal{V}_V$ such that $(\nu, \nu') \in \mu$. However, we require that resets may only change controlled variables since the value of a variable should only be determined by the component controlling the variable. Note that it is possible to reset a controlled variable to the current value of a non-controlled variable and that guards may depend on non-controlled variables. Hence, the environment can influence when which jump can be taken.

Note that, per definition, we also require the new valuation to satisfy the location invariant of the target location. If the jumps were specified in a more syntactic style, it would make more sense to impose this requirement only in the semantics. Otherwise, it would be quite cumbersome for the modeler to assure that the target invariant will be satisfied after a controlled variable has been reset to a non-controlled variable.

Each jump has a jump identifier $id \in \mathbb{N}$, which must not necessarily be unique. The jump identifiers essentially enable us to define multisets, i.e., to have several jumps with different jump identifiers but the same guard, reset, source location, and target location. This is needed for the composition of stochastic hybrid automata, as we elaborate in Section 3.3. In order to improve readability, we often omit the jump identifiers when they are not relevant.

Act Since the activities are defined as solutions of ordinary differential equations, the controlled variables must evolve continuously in closed systems. For non-closed systems, we assume that the non-controlled variables are controlled by other hybrid automata and thus also evolve continuously, which implies that the controlled variables must evolve continuously as well. Hence, we assume that all variables evolve continuously on $\mathbb{R}_{\geq 0}$.

As mentioned above, a hybrid automaton should not restrict the continuous evolution of the non-controlled variables. A hybrid automaton may only specify the evolution of its controlled variables, which may be influenced by the non-controlled variables. This property is inherently captured in our ODE systems of the form $\dot{\mathbf{x}}_{Con} = A_l \mathbf{x}_{Con} + B_l \mathbf{x}_{NCon} + \mathbf{c}_l$ (with the same meaning as in Definition 2.1.1).

We chose to restrict the ODE systems to linear first-order systems here, because they guarantee the existence of a unique global solution for every initial value problem [Zil12]. However, this solution cannot always be computed explicitly. It might be possible to extend our model to more general ODE systems. For example, ODEs specified by Lipschitz continuous functions must have unique maximal solutions, but not necessarily global ones [Mar04].

In our setting, for each state $(l,\nu) \in \Sigma$ and each fixed environmental evolution $f_N \in F_{NCon}$ with $f_N(0) = \nu|_{NCon}$, the linear ODE system associated with location l has a unique solution $f_C \in F_{Con}$ of the initial value problem $f_C(0) = \nu|_{Con}$ induced by ν [Zil12]. Only if the ODE does not depend on non-controlled variables, the evolution of the controlled variables is uniquely determined at each state. Otherwise, the solution of the ODE depends on the evolution of the environment, which we cannot determine or predict. In general, we only know the direction of the next infinitesimally small continuous update at each state. For example, consider the simple ODE $\dot{x} = y$ with $x \in Con, y \in NCon$. The unique solution $f \in F$ satisfying the initial value problem $f(0)(x) = x_0$ is given by $f(t)(x) = \int_{r=0}^{t} f(r)(y)dr + x_0$. However, at time 0, we do not know the activity $f(r)(y)$ describing how y will evolve until time t, and thus we cannot compute the integral. Hence, the evolution of the controlled variables is not necessarily uniquely determined in each state, i.e., for each (l,ν) there may be several $f \in Act(l)$ such that $f(0) = \nu$.

The continuous dynamics should also be memoryless, i.e., the evolution of the variables should only depend on the current state. Thus, in the state (l, ν) the same evolution of the variables should be possible independently of when location l was entered. This means that all activity sets $Act(l)$ should be time-invariant, i.e., $f \in Act(l)$ should imply $(f+t) \in Act(l)$ for all $t \in \mathbb{R}_{\geq 0}$, where $(f+t)(t') = f(t+t')$ for all $t' \in \mathbb{R}_{\geq 0}$. Since the activity sets in our model are determined by ODEs of the form $\dot{\mathbf{x}}_{Con} = f(V)$, they are time-invariant by nature. Note that, if we allowed ODEs of the form $\dot{\mathbf{x}}_{Con} = f(t, V)$, the activities would not necessarily be time-invariant.

The hybrid automata introduced in the following two examples serve as running examples that are reused and extended throughout this thesis.

Example 2.1.1. *Figure 2.1 presents a graphical representation of the hybrid automaton* $\mathcal{H}_1 = (Loc_1, (Con_1, NCon_1), Inv_1, Init_1, Edge_1, Act_1)$ *with the following formal definition:*

- $Loc_1 = \{l_0, l_1\}$;
- $V_1 = \{x\}, Con_1 = \{x\}, NCon_1 = \emptyset$, $\mathcal{V}_1 = \{\nu : V_1 \to \mathbb{R}\}$;

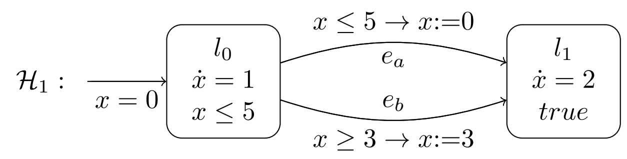


Figure 2.1: Depiction of the hybrid automaton $\mathcal{H}_1$ without non-controlled variables defined in Example 2.1.1.

- $\mathit{Inv}_1(l_0) = \{\nu \in \mathcal{V}_1 \mid \nu(x) \leq 5\}$,
  $\mathit{Inv}_1(l_1) = \mathcal{V}_1$;
- $\mathit{Init}_1(l_0) = \{(x \mapsto 0)\}$,
  $\mathit{Init}_1(l_1) = \emptyset$;
- $\mathit{Edge}_1 = \{e_a = (l_0,\{(\nu,\nu') \in \mathcal{V}_1 \times \mathcal{V}_1 \mid \nu(x) \leq 5 \wedge \nu'(x) = 0\},l_1),$
  $e_b = (l_0,\{(\nu,\nu') \in \mathcal{V}_1 \times \mathcal{V}_1 \mid \nu(x) \geq 3 \wedge \nu'(x) = 3\},l_1)\}$;
- *$\mathit{Act}_1(l_0)$ contains all solutions to the linear ODE $\dot{x} = 1$, i.e.,*
  $\mathit{Act}_1(l_0) = \{f\colon \mathbb{R}_{\geq 0} \rightarrow \mathcal{V}_1, t \mapsto (x \mapsto t + c) \mid c \in \mathbb{R}\}$,
  $\mathit{Act}_1(l_1)$ contains all solutions to the linear ODE $\dot{x} = 2$, i.e.,
  $\mathit{Act}_1(l_1) = \{f\colon \mathbb{R}_{\geq 0} \rightarrow \mathcal{V}_1, t \mapsto (x \mapsto 2t + c) \mid c \in \mathbb{R}\}$.

It can easily be verified that $\mathcal{H}_1$ satisfies the stipulated conditions.

Example 2.1.2. *Figure 2.2 depicts the hybrid automaton $\mathcal{H}_2 = (\mathit{Loc}_2, (\mathit{Con}_2, \mathit{NCon}_2), \mathit{Inv}_2, \mathit{Init}_2, \mathit{Edge}_2, \mathit{Act}_2)$ with the following formal definition:*

- $\mathit{Loc}_2 = \{l_2, l_3\}$;
- $V_2 = \{x, y\}$, $\mathit{Con}_2 = \{y\}$, $\mathit{NCon}_2 = \{x\}$, $\mathcal{V}_2 = \{\nu\colon V_2 \rightarrow \mathbb{R}\}$;
- $\mathit{Inv}_2(l_2) = \mathcal{V}_2$,
  $\mathit{Inv}_2(l_3) = \mathcal{V}_2$;
- $\mathit{Init}_2(l_2) = \{(x \mapsto c, y \mapsto 0) \mid c \in \mathbb{R}\}$,
  $\mathit{Init}_2(l_3) = \emptyset$;
- $\mathit{Edge}_2 = \{(l_2, \{(\nu,\nu') \in \mathcal{V}_2 \times \mathcal{V}_2 \mid \nu(x) > 2 \wedge \nu = \nu'\},l_3)\}$;
- $\mathit{Act}_2(l_2) = \{f\colon \mathbb{R}_{\geq 0} \rightarrow \mathcal{V}_2 \mid \frac{df(t)(y)}{dt} = f(t)(x)\}$,
  $\mathit{Act}_2(l_3) = \{f\colon \mathbb{R}_{\geq 0} \rightarrow \mathcal{V}_2, t \mapsto (x \mapsto f_x(t), y \mapsto t + c) \mid c \in \mathbb{R}, f_x\colon \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}\}$.

For $\mathcal{H}_2$ it can again be easily seen that it satisfies the required conditions.

The following examples present slight variations of the hybrid automaton $\mathcal{H}_2$ from Example 2.1.2 which would lead to invariants or initial valuation sets not being NCon_2-closed anymore, or the conditions on jumps not being met.

Example 2.1.3. *Consider the following variations of $\mathcal{H}_2$ from Example 2.1.2. If the invariant of l_2 was changed to '$x < 20$', then $\mathit{Inv}_2(l_2)$ would depend on a non-controlled variable and would thus not be NCon_2-closed anymore. NCon_2-closure for the set of initial valuations of location l_2 could for example be violated by choosing the initial valuation set $\{(x \mapsto c, y \mapsto 0) \mid c \in \mathbb{R}, c < 5\}$.*

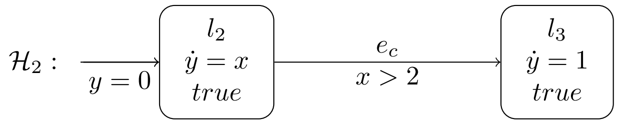


Figure 2.2: Hybrid automaton $\mathcal{H}_2$ with dependencies on a non-controlled variable, as defined in Example 2.1.2.

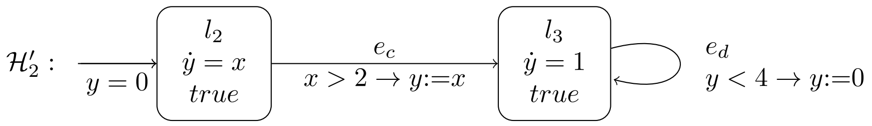


Figure 2.3: Hybrid automaton $\mathcal{H}'_2$ with indirect dependencies on the environment, as defined in Example 2.1.6.

Example 2.1.4. *If the reset of jump e_c in $\mathcal{H}_2$ was '$x := 6$', then the first requirement on $Edge_2$ would not be satisfied. We would have $\mu_c = \{(\nu, \nu') \in \mathcal{V}_2 \times \mathcal{V}_2 \mid \nu(x) > 2 \wedge \nu'(x) = 6 \wedge \nu'(y) = \nu(y)\}$. Thus, for $\nu = (x \mapsto 4, y \mapsto 0)$ and $\nu' = (x \mapsto 6, y \mapsto 0)$ we have $(\nu, \nu') \in \mu_c$ but $\nu \not\approx_{NCon_2} \nu'$.*

The second requirement on $Edge_2$ must be satisfied since both jumps have l_1 as target location, whose invariant is 'true'. If the location invariant was '$x > 1$' instead, then resetting x to 0 by taking jump e_a would violate this condition.

Note that, if $\mathcal{H}_2$ was composed with another hybrid automaton controlling x, resetting x when taking e_c could potentially violate the invariant of the other hybrid automaton's current location. For example, $\mathcal{H}_1$ controls x and its initial location has the invariant '$x \leq 5$', which would clearly become violated if $\mathcal{H}_2$ took the above reset.

Guards allow comparing different variables, in particular also comparing the values of controlled to the values of non-controlled variables.

Example 2.1.5. *If the guard of e_c in $\mathcal{H}_2$ was '$x > y$', the hybrid automaton would still be valid. However, without knowing how x evolves, we cannot know whether e_c will ever become enabled or for how long.*

The following example illustrates that there may exist implicit dependencies on the environment in addition to the explicit ones.

Example 2.1.6. *The hybrid automaton $\mathcal{H}'_2$ presented in Figure 2.3 extends $\mathcal{H}_2$ from Example 2.1.2 by a self-loop e_d on l_3. Additionally, the reset of e_c is changed to '$y := x$'. Even though the guard of e_d and the invariant and activity set of l_3 do not depend on the non-controlled variable x, the enabledness of e_d in l_3 indirectly depends on the environment, because we reset y to the value of x when transitioning from l_2 to l_3.*

We conclude this section by defining notions of determinism with regard to the component itself and its environment, which we will need when we extend hybrid automata to stochastic hybrid automata in Chapter 3.

2.1.1 Deterministic Hybrid Automata

When we extend hybrid automata by stochastic behavior regarding the jump times in Chapter 3, all other non-determinism regarding the behavior of this component needs to be resolved. Here, we choose to do so by requiring the hybrid automaton to be *init-*, *reset-*, and *activity-deterministic.*

Definition 2.1.3 (Init-, Reset-, Activity-deterministic Hybrid Automata)**.** *A hybrid automaton $\mathcal{H} = (Loc, (Con, NCon), Inv, Init, Edge, Act)$ is called*

- init-deterministic *if there exists a unique $l_0 \in Loc$ such that for all $l \in Loc$ we have $Init(l) \neq \emptyset$ iff $l = l_0$, and for all $\nu, \nu' \in Init(l_0)$ we have $\nu \approx_{Con} \nu'$,*
- reset-deterministic *if for each $(id, l, \mu, l') \in Edge$ and $\nu \in \mathcal{V}$ there is at most one $\nu' \in \mathcal{V}$ such that $(\nu, \nu') \in \mu$, and*
- activity-deterministic *if for each $l \in Loc$ and $\nu \in \mathcal{V}$ and for each fixed evolution of the environment $f_N \in F_{NCon}$ with $f_N(0) = \nu|_{NCon}$, there exists a unique $f \in Act(l)$ such that $f \approx_{NCon} f_N$ and $f(0) = \nu$.*

A hybrid automaton is init-deterministic if there exists exactly one location with a non-empty initial valuation set and all valuations in this set agree on the controlled variables. In other words, there is a unique initial state with respect to the controlled variables.

Reset-determinism means that for each valuation satisfying the guard of some jump, it is uniquely defined how the (controlled) variables should be reset after taking the jump. Recall that resets may only affect controlled variables and hence the reset of the non-controlled variables must always be uniquely determined anyways.

A hybrid automaton is activity-deterministic if in each state the evolution of controlled variables is uniquely determined for each fixed evolution of the non-controlled variables. Recall that this must hold for all hybrid automata by definition, since there is a unique solution to the initial value problem for linear ODEs.

Example 2.1.7. *The hybrid automata $\mathcal{H}_1$ and $\mathcal{H}_2$ from Examples 2.1.1 and 2.1.2 are init-, reset-, and activity-deterministic.*

2.1.2 Closed Hybrid Automata

Since we model the environmental behavior non-deterministically, we can only properly analyze *closed* hybrid automata, which are not affected by the environment. In particular, we can only define a probability space over paths of stochastic hybrid automata if the underlying hybrid automaton is closed.

Definition 2.1.4 (Closed Hybrid Automaton)**.** *A hybrid automaton $\mathcal{H} = (Loc, (Con, NCon), Inv, Init, Edge, Act)$ is called* closed *if $NCon = \emptyset$.*

If a hybrid automaton is closed, then for each state $(l, \nu) \in \Sigma$ there exists a unique $f \in Act(l)$ such that $f(0) = \nu$.

We could alternatively also define a weaker notion of closedness, which allows the presence of non-controlled variables if they do not influence the hybrid automaton's behavior in any way. If no guard, reset, or activity set depends on any non-controlled variable, removing the non-controlled variables from the hybrid automaton would not affect its behavior. Closing the hybrid automaton in this manner would then enable us to analyze its behavior. For simplicity, we nevertheless choose a stricter definition of closedness here.

Example 2.1.8. *The hybrid automaton $\mathcal{H}_1$ defined in Example 2.1.1 is closed. Even if the set of non-controlled variables was non-empty, but everything else remained unchanged, we could still consider this hybrid automaton to be closed in the weaker sense discussed above: The behavior of $\mathcal{H}_1$ does not depend on its environment.*

$\mathcal{H}_2$ defined in Example 2.1.2 is clearly not closed.

2.2 Semantics of Hybrid Automata

For the definition of the semantics, we again follow Alur et al. [ACH+95]. A hybrid automaton can take an *execution step* from a state σ by letting time pass or taking a discrete jump. It additionally synchronizes with its environment by taking *environmental steps* that match discrete steps taken by other components. Note again that our model currently does not offer jump synchronization via labels, but components can instead communicate using shared variables. The behavior of a hybrid automaton can be described by *paths* consisting of consecutive execution steps.

Definition 2.2.1 (Semantics of Hybrid Automata)**.** *The semantics of a hybrid automaton $\mathcal{H} = (Loc, (Con, NCon), Inv, Init, Edge, Act)$ is given by an operational semantics consisting of three rules:*

$$\frac{f \in Act(l) \qquad t > 0 \qquad f(0) = \nu \qquad \forall t' \in [0,t].\ f(t') \in Inv(l)}{(l,\nu) \xrightarrow{t,f} (l, f(t))}\ \mathtt{Rule}_{\mathtt{time}}$$

$$\frac{e = (id, l, \mu, l') \in Edge \qquad (\nu,\nu') \in \mu}{(l,\nu) \xrightarrow{e} (l',\nu')}\ \mathtt{Rule}_{\mathtt{discrete}}$$

$$\frac{\nu \approx_{Con} \nu' \qquad \nu' \in Inv(l)}{(l,\nu) \xrightarrow{\tau} (l,\nu')}\ \mathtt{Rule}_{\mathtt{environment}}$$

An execution step

$$\rightarrow = \left(\bigcup_{t \in \mathbb{R}_{\geq 0}, f \in F} \xrightarrow{t,f} \right) \cup \left(\bigcup_{e \in Edge} \xrightarrow{e} \right) \cup \xrightarrow{\tau}$$

of $\mathcal{H}$ is either a time step or a discrete step or an environmental step. Instead of writing $(\sigma, \sigma') \in \rightarrow$, we use the infix notation $\sigma \rightarrow \sigma'$, and analogously write $\sigma \not\rightarrow \sigma'$ for $(\sigma, \sigma') \notin \rightarrow$.

A path *π of $\mathcal{H}$ is a (finite or infinite) sequence of states connected by execution steps $\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \rightarrow \ldots$ such that $\nu_0 \in Inv(l_0)$.*

A state σ of $\mathcal{H}$ is called reachable *iff there exists a path of $\mathcal{H}$ leading to σ which starts in an initial state.*

In order to be able to make a time step $(l,\nu) \xrightarrow{t,f} (l, \nu')$, the location's invariant needs to hold during the complete interval $[0,t]$. We keep track of the activity f used to evolve from ν to ν' since there might be several different activities in $Act(l)$ with $f(0) = \nu$ and $f(t) = \nu'$, because we model the environmental behavior nondeterministically.

For a discrete step $(l,\nu) \xrightarrow{e} (l',\nu')$, the valuations ν and ν' need to satisfy the jump's guard and reset. We already ensured syntactically that ν' must satisfy the target location's invariant.

We allow the environment to take arbitrary jumps changing (some of) our non-controlled variables, as long as the location invariants still hold. Since we require location invariants to only depend on controlled variables, they can, in fact, not be directly violated by the environment. As mentioned above, Alur et al. [ACH+95] defined such environmental steps syntactically. Since adding these jumps for all possible changes of the environment is quite laborious for the modeler, we choose to include them semantically instead.

Recall that initial valuations have to satisfy the location invariants. Since no execution step can violate a location invariant, the location invariants must be satisfied in each state along each reachable path.

Let us briefly illustrate the concepts of execution steps and paths with our running examples.

Example 2.2.1. *Consider again the hybrid automata $\mathcal{H}_1$ and $\mathcal{H}_2$ defined in Examples 2.1.1 and 2.1.2. In $\mathcal{H}_1$, all infinite paths from the unique initial state $(l_0, (x \mapsto 0))$ must transition to l_1 before '$x > 5$', using either e_a or e_b, and finally stay in l_1 forever. For example, the finite path*

$$\pi = (l_0, (x \mapsto 0)) \xrightarrow{4,f_0} (l_0, (x \mapsto 4)) \xrightarrow{e_b} (l_1, (x \mapsto 3)) \xrightarrow{1,f_1} (l_1, (x \mapsto 5))$$

of $\mathcal{H}_1$ with $f_0(t)(x) = t$, $f_1(t)(x) = 2t + 3$ can be extended to an infinite path π' of $\mathcal{H}_1$ by adding infinitely many time steps that follow the prescribed evolution of x.

Since $\mathcal{H}_1$ is closed, we cannot make any environmental steps in $\mathcal{H}_1$. However, $\mathcal{H}_2$ does contain a non-controlled variable x and hence, for example, the environmental step $(l_2, (y \mapsto 0, x \mapsto 0)) \xrightarrow{\tau} (l_2, (y \mapsto 0, x \mapsto 3))$ is possible.

2.2.1 Zenoness, Timelock, and Deadlock

Not all behavior that can be described by the semantics is realistic. For example, the semantics allows to define infinite paths that only take finite time, called *time-convergent* paths [BK08]. Analogously, infinite paths of infinite duration are called *time-divergent.*

Definition 2.2.2 (Time-Convergence)**.** *For a hybrid automaton $\mathcal{H} = (Loc, (Con, NCon), Inv, Init, Edge, Act)$, we define the time duration of an execution step α by the function $ExecTime\colon (Edge \cup (\mathbb{R}_{\geq 0} \times F) \cup \{\tau\}) \to \mathbb{R}_{\geq 0}$ with*

$$ExecTime(\alpha) = \begin{cases} t & \text{if } \alpha = (t,f) \in (\mathbb{R}_{\geq 0} \times F) \\ 0 & \text{otherwise.} \end{cases}$$

The time duration of an infinite path $\pi = \sigma_0 \xrightarrow{\alpha_0} \sigma_1 \xrightarrow{\alpha_1} \sigma_2 \xrightarrow{\alpha_2} \dots$ of $\mathcal{H}$ is given by the overloaded function

$$ExecTime(\pi) = \sum_{i=0}^{\infty} ExecTime(\alpha_i).$$

An infinite path π of $\mathcal{H}$ is called time-divergent *if $ExecTime(\pi) = \infty$ and* time-convergent *otherwise.*

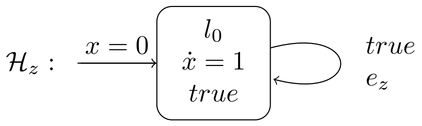


Figure 2.4: Hybrid automaton $\mathcal{H}_z$ containing a Zeno path, as defined in Example 2.2.3.

In the following, we first explore a special kind of time-convergent paths called *Zeno* paths and then discuss states from which no, or no realistic, paths can be taken, which is captured by the notions of *deadlock* and *timelock*.

Zenoness

The semantics allows to define so-called *Zeno* paths, which are infinite paths of finite duration that include infinitely many discrete steps. A hybrid automaton is called non-Zeno if no path from an initial state is Zeno, i.e., no Zeno path is reachable. We start by defining Zeno paths for closed hybrid automata based on [AL94, LTS99, BK08].

Definition 2.2.3 (Zeno Path)**.** *Let $\mathcal{H} = (Loc, (Con, NCon), Inv, Init, Edge, Act)$ be a closed hybrid automaton. An infinite path π of $\mathcal{H}$ is called a* Zeno path *if it is time-convergent and infinitely many discrete steps are taken within π. The hybrid automaton $\mathcal{H}$ is* non-Zeno *if no Zeno path of $\mathcal{H}$ is starting from an initial state.*

After illustrating this definition with two examples, we will discuss whether and how it could be extended to non-closed systems.

Example 2.2.2. *The hybrid automaton $\mathcal{H}_1$ from Example 2.1.1 is non-Zeno since every infinite path of $\mathcal{H}_1$ contains at most one discrete step, either jump e_a or e_b.*

Example 2.2.3. *Consider the closed hybrid automaton $\mathcal{H}_z$ presented in Figure 2.4, which controls a variable x and has a single location whose only outgoing jump e_z leads back to that location. Consider an infinite path of $\mathcal{H}_z$ with progressively smaller time steps in between jumps, e.g.,*

$$\begin{aligned}\pi = (l_0, (x \mapsto 0)) &\xrightarrow{t_0, f_0} (l_0, (x \mapsto t_0)) \xrightarrow{e_z} \\ &(l_0, (x \mapsto t_0)) \xrightarrow{t_1, f_1} (l_0, (x \mapsto t_0 + t_1)) \xrightarrow{e_z} \\ &(l_0, (x \mapsto t_0 + t_1)) \xrightarrow{t_2, f_2} (l_0, (x \mapsto t_0 + t_1 + t_2)) \xrightarrow{e_z} \\ &\ldots\end{aligned}$$

with $t_i = \frac{1}{2^i}$ and $f_i(t)(x) = t + \sum_{k=0}^{i-1} t_k$ for $i \in \mathbb{N}, t \in \mathbb{R}_{\geq 0}$. π is time-convergent since $ExecTime(\pi) = \sum_{i=0}^{\infty} t_i = 2$, but takes e_z infinitely often. Hence, π is a Zeno path.

If we want to analyze the Zeno behavior of non-closed hybrid automata, we should also consider paths that contain infinitely many environmental steps as Zeno paths, since an environmental step corresponds to a discrete step taken by another component.

A Zeno path containing infinitely many environmental steps but finitely many discrete steps is not the "fault" of this hybrid automaton, since another component takes infinitely many discrete steps. In fact, paths with infinitely many environmental steps are unavoidable since, in particular, environmental steps that do not change the values of the variables are always possible [1].

Zeno paths containing infinitely many discrete steps could be considered “locally” Zeno paths for non-closed systems, expressing that the Zenoness is caused by the component itself. These would also be considered Zeno if we simply applied the standard definition to non-closed hybrid automata. A locally Zeno path might only be possible under certain environmental behavior, for example, it might contain environmental steps or discrete steps taking jumps whose guards depend on non-controlled variables.

The following example illustrates our considerations.

Example 2.2.4. *Recall the hybrid automaton $\mathcal{H}_2'$ from Example 2.1.6 (Figure 2.3), where the guard of the jump e_c depends on the environment and the enabledness of the jump e_d also depends on the environment, but only implicitly. A time-convergent path taking e_c and then infinitely often e_d with progressively smaller time steps is a locally Zeno path. For example, consider the following infinite path*

$$\begin{aligned}(l_2,(y\mapsto 0,x\mapsto 0)) &\xrightarrow{\tau} (l_2,(y\mapsto 0,x\mapsto 5)) \xrightarrow{e_c} (l_3,(y\mapsto 5,x\mapsto 5))\\ &\xrightarrow{e_d} (l_3,(y\mapsto 0,x\mapsto 3)) \xrightarrow{e_d} (l_3,(y\mapsto 0,x\mapsto 3)) \xrightarrow{e_d} \dots\end{aligned}$$

continuing to loop in l_3 using e_d, which has time duration 0.

Further, we can build Zeno paths, which are not necessarily locally Zeno, from any initial state by taking infinitely many environmental steps, for example,

$$(l_2,(y\mapsto 0,x\mapsto 0)) \xrightarrow{\tau} (l_2,(y\mapsto 0,x\mapsto 1)) \xrightarrow{\tau} (l_2,(y\mapsto 0,x\mapsto 2)) \xrightarrow{\tau} \dots$$

which does not let time pass at all.

We could alternatively consider defining a locally Zeno path as a Zeno path that is possible under all possible environmental behavior. However, we deem this definition to be too strict since it only considers paths to be locally Zeno if they do not have any dependencies on non-controlled variables in guards or activities. Thus, this definition would likely only detect Zeno paths in systems that are only influenced by non-controlled variables in a very restricted way.

In summary, we do not see any possibility to define Zenoness for non-closed hybrid automata in a way that captures the desired notion sufficiently and allows a meaningful analysis.

Timelock and Deadlock

A hybrid automaton might contain states from which only time-convergent paths are possible. We say that such a state has a *timelock*. A *deadlock* is a special kind of timelock where neither time-divergent nor time-convergent paths are possible.

To determine whether a state has a deadlock or timelock, we need to consider the future behavior of the system. If the system is not closed, it might depend on the future behavior of the environment whether a (time-divergent) path exists. We start by defining timelocks and deadlocks for closed hybrid automata based on [BK08, Tri99, Bow01][2]. Afterwards, we discuss how these notions could be extended to non-closed hybrid automata.

[1]Note that it would not make sense to exclude such steps in the semantics since this might block certain jumps where the variables are either reset to the same values as before or not reset at all.

[2]Note that our terminology differs from Bowman’s terminology.

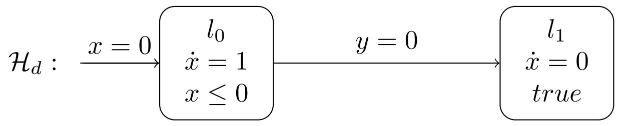


Figure 2.5: Non-closed hybrid automaton $\mathcal{H}_d$ that can be forced to deadlock by the environment, as defined in Example 2.2.6.

Definition 2.2.4 (Timelock, Deadlock)**.** *Let $\mathcal{H} = (Loc, (Con, NCon), Inv, Init, Edge, Act)$ be a closed hybrid automaton.*

A state $\sigma \in \Sigma$ has a timelock *if there does not exist any time-divergent path of $\mathcal{H}$ starting at σ. The hybrid automaton is said to be* timelock-free *if none of its reachable states have a timelock.*

A state $\sigma \in \Sigma$ has a deadlock *if there does not exist any infinite path of $\mathcal{H}$ starting from σ. The hybrid automaton is said to be* deadlock-free *if none of its reachable states have a deadlock.*

A very simple, but quite restrictive, way to avoid timelocks (and thus also deadlocks) is to allow only the invariant '*true*', because then time can always diverge by staying in the current location. The following example illustrates the concepts of timelock and deadlock.

Example 2.2.5. *The hybrid automaton $\mathcal{H}_1$ is timelock-free because no reachable state has a timelock. A state of the form (l_0, ν) is only reachable if $\nu(x) \leq 5$, which means that at least jump e_a can be taken from that state. From states of the form (l_1, ν) it is always possible to take a time-divergent path by letting time elapse in location l_1 forever.*

If the invariant of l_1 in $\mathcal{H}_1$ was '$x \leq 10$' instead of 'true', then $\mathcal{H}_1$ would not be timelock-free, since there would not exist any time-divergent paths from any state (l_1, ν). Hence, there would not exist any time-divergent path from any state (l_0, ν) either and thus all reachable states would have a timelock. In particular, the state $(l_1, (x \mapsto 10))$ would be deadlocked.

For non-closed hybrid automata, there are at least two possibilities for the definition of timelocks, and analogously for deadlocks. Firstly, we could consider a state to have a timelock if all paths are time-convergent, as for closed hybrid automata. However, this would only capture timelocks that occur independently of the environmental behavior. Alternatively, we could consider a state to be timelocked if there exists some environmental behavior such that all paths are time-convergent. This condition, however, seems too strong since we can envision many cases where a guard or ODE depends on the environment in such a way that there always exists some environmental behavior that forces the component into time-convergent behavior. The following example illustrates these conditions for deadlocks.

Example 2.2.6. *Consider the hybrid automaton $\mathcal{H}_d$ presented in Figure 2.5, which has a controlled variable x and its only jump depends on the non-controlled variable y. From any initial state (l_0, ν) with $\nu(x) = 0$, taking any time step would violate the location's invariant. Hence, we must immediately leave the location via the only jump.*

If $\nu(y) \neq 0$, leaving the location is only possible if we assume that an environmental step is taken first, which resets y such that the jump's guard is satisfied. Thus, we *can always construct a (time-divergent) path from any initial state and we could argue that no initial state has a deadlock, following the first considered possibility to define a timelock.*

Conversely, from any initial state there exists some environmental behavior such that no time-divergent path can be taken, namely if we assume that the environment initially takes some jump which resets y to a value other than 0. Thus, we could also argue that each initial state does have a deadlock, following the second alternative for the definition of timelocks.

In summary, both considered definitions of timelock and deadlock for non-closed systems seem insufficient. We currently do not see any way to formulate a meaningful definition of timelock and deadlock for non-closed hybrid automata, because the actual behavior of a hybrid automaton depends on the environment.

Excluding Unrealistic Behavior

Time-convergent paths and timelocks clearly do not describe realistic behavior and Zeno paths in particular are often excluded from analysis in other works (e.g. [Hes05, PKRA20, LLA$^+$21]). However, we have seen that Zeno paths containing infinitely many environmental steps are unavoidable in the compositional context. In fact, timelocks and Zeno paths often occur in models of real systems in general [ZJLS01]. For example, the modeling of a bouncing ball usually contains a timelock and a Zeno path [JELS99]. Furthermore, if one is interested in proving that a system is, e.g., deadlock-free, it is of course not sensible to require systems to be deadlock-free in the first place. In this thesis, we always explicitly mention when we exclude any of the described behavior.

2.3 Composition of Hybrid Automata

In this section, we define the *syntactic parallel composition* of hybrid automata. When two hybrid automata are executed in parallel, they either both take consistent time steps or one of them takes a discrete jump while the other one takes a matching environmental step. As mentioned before, several components may communicate via shared variables. Our model does not offer jump synchronization or concurrent writing of shared variables, but we discuss in Chapter 4 how these features could be included in our language. Currently, we only allow each variable to be written by exactly one component. Consequently, we consider two hybrid automata *composable* if their sets of controlled variables are disjoint. Additionally, we require that jumps from different components have different identifiers, which is necessary for the composition of stochastic hybrid automata with stochastic jumps and is explained in more detail in Section 3.3.

Definition 2.3.1 (Composability of Hybrid Automata)**.** *Two hybrid automata $\mathcal{H}_1$ and $\mathcal{H}_2$ are* composable *if $Con_1 \cap Con_2 = \emptyset$ and for all $(id_1, l_1, \mu_1, l_1') \in Edge_1$, $(id_2, l_2, \mu_2, l_2') \in Edge_2$ we have $id_1 \neq id_2$.*

In order to keep the definition of the composition focused on the most relevant parts, we first define the *extension* of a hybrid automaton by a set of variables. More specifically, we want to extend its set of non-controlled variables by some variables controlled by other hybrid automata. This will allow us to define the composition of hybrid automata with different variable sets more easily.

2.3.1 Extension of Hybrid Automata

When we extend a hybrid automaton by adding new variables to the set of non-controlled variables, we need to extend all jumps, activity sets, invariants, and initial valuation sets such that all possible behavior of the added variables is allowed. In particular, all valuation sets should be closed with respect to the added variables.

Definition 2.3.2 (Extension of Hybrid Automata)**.** *Let $\mathcal{H} = (Loc, (Con, NCon), Inv, Init, Edge, Act)$ be a hybrid automaton and V' be some set of real-valued variables such that $Con \cap V' = \emptyset$. The* extension *of $\mathcal{H}$ by V' is the hybrid automaton*

$$\mathcal{H}^+ = (Loc^+, (Con^+, NCon^+), Inv^+, Init^+, Edge^+, Act^+)$$

with

- *$Loc^+ = Loc$;*
- *$Con^+ = Con$, $NCon^+ = NCon \cup V'$ and $V^+ = Con^+ \cup NCon^+$; we write $\mathcal{V}^+$ and F^+ to denote $\mathcal{V}_{V^+}$ and F_{V^+}, respectively;*
- *$Inv^+(l) = \{\nu^+ \in \mathcal{V}^+ \mid \exists \nu \in Inv(l).\ \nu = \nu^+|_V\}$ for all $l \in Loc^+$;*
- *$Init^+(l) = \{\nu^+ \in \mathcal{V}^+ \mid \exists \nu \in Init(l).\ \nu = \nu^+|_V\}$ for all $l \in Loc^+$;*
- *$(id, l, \mu^+, l') \in Edge^+$ iff there exists some $(id, l, \mu, l') \in Edge$ such that*

$$\mu^+ = \left\{ (\nu_1^+, \nu_2^+) \in \mathcal{V}^+ \times \mathcal{V}^+ \;\middle|\; \begin{array}{l} \exists (\nu_1, \nu_2) \in \mu.\ \nu_1 = \nu_1^+|_V \wedge \nu_2 = \nu_2^+|_V \\ \wedge\ \nu_1^+ \approx_{NCon^+} \nu_2^+ \end{array} \right\};$$

  for each $e \in Edge$ we denote the unique corresponding jump in $Edge^+$ by e^+;
- *$Act^+(l)$ consists of all $f \in F_{V^+}$ satisfying $\dot{\mathbf{x}}_{Con} = A_l \mathbf{x}_{Con} + B_l^+ \mathbf{x}_{NCon^+} + \mathbf{c}_l$ for all $l \in Loc^+$, where*
  - *$\dot{\mathbf{x}}_{Con} = A_l \mathbf{x}_{Con} + B_l \mathbf{x}_{NCon} + \mathbf{c}_l$ is the ODE associated with l in $\mathcal{H}$,*
  - *$\mathbf{x}_{NCon^+}$ is a vector of the extended non-controlled variables, and*
  - *B_l^+ is a suitable real-valued matrix such that $B_l \mathbf{x}_{NCon} + 0\mathbf{x}_{V'} = B_l^+ \mathbf{x}_{NCon^+}$, where $\mathbf{x}_{V'}$ is a vector of the added variables.*

Loc*, *Con*, *NCon We require the set of new variables to be disjoint from the set of controlled variables in order to avoid conflict between two hybrid automata wanting to control the same variable. When extending a hybrid automaton by some variable set, the locations, and controlled variables remain unchanged. The new variables are added to the set of non-controlled variables, since we assume them to be controlled by some other component. Valuations in the new extended automaton now also need to include values for the newly added variables.

Inv*, *Init The invariant and the initial valuation set of each location should not impose any restrictions on the added variables either, i.e., they should be closed with respect to the added variables. In other words, we allow all possible values of the added variables in those valuation sets. Since they are closed with respect to the original non-controlled variables and with respect to the added variables, they are closed with respect to the extended set of non-controlled variables, as required. It is easy to see that the extension preserves the inclusion of the initial valuation set in the invariant for each location.

Edge Each jump needs to be extended such that its guard allows all possible values for the new variables and such that the new variables are not reset, while the behavior on the original variables is unaffected. In particular, the invariant of the target location must still be satisfied.

Act For each location $l \in \mathit{Loc}^+$, the set of activities needs to be extended such that it allows for all possible evolution of the added variables. Hence, $\mathit{Act}^+(l)$ contains all $f \in F_{V^+}$ for which there exists some $f \in \mathit{Act}(l)$ such that $f(t) = f^+(t)|_V$ for all $t \in \mathbb{R}_{\geq 0}$.

The following examples illustrate the extension of hybrid automata using our running examples.

Example 2.3.1. *Recall the hybrid automaton $\mathcal{H}_1$ from Example 2.1.1 with variable set $V_1 = \mathit{Con}_1 = \{x\}$. Extending $\mathcal{H}_1$ by another variable y yields the hybrid automaton $\mathcal{H}_1^+ = (\mathit{Loc}_1^+, (\mathit{Con}_1^+, \mathit{NCon}_1^+), \mathit{Inv}_1^+, \mathit{Init}_1^+, \mathit{Edge}_1^+, \mathit{Act}_1^+)$ with*

- $\mathit{Loc}_1^+ = \{l_0, l_1\}$;
- $V_1^+ = \{x, y\}$, $\mathit{Con}_1^+ = \{x\}$, $\mathit{NCon}_1^+ = \{y\}$, $\mathcal{V}_1^+ = \{\nu \colon V_1^+ \to \mathbb{R}\}$;
- $\mathit{Inv}_1^+(l_0) = \{\nu \in \mathcal{V}_1^+ \mid \nu(x) \leq 5\}$,
  $\mathit{Inv}_1^+(l_1) = \mathcal{V}_1^+$;
- $\mathit{Init}_1^+(l_0) = \{(x \mapsto 0, y \mapsto c) \mid c \in \mathbb{R}\}$,
  $\mathit{Init}_1^+(l_1) = \emptyset$;
- $\mathit{Edge}_1^+ = \{(l_0, \mu_a^+, l_1), (l_0, \mu_b^+, l_1)\}$ *with*
  - $\mu_a^+ = \{(\nu, \nu') \in \mathcal{V}_1^+ \times \mathcal{V}_1^+ \mid \nu(x) \leq 5 \wedge \nu'(x) = 0 \wedge \nu(y) = \nu'(y)\}$,
  - $\mu_b^+ = \{(\nu, \nu') \in \mathcal{V}_1^+ \times \mathcal{V}_1^+ \mid \nu(x) \geq 3 \wedge \nu'(x) = 3 \wedge \nu(y) = \nu'(y)\}$;
- $\mathit{Act}_1^+(l_0) = \{f \colon \mathbb{R}_{\geq 0} \to \mathcal{V}_1^+, t \mapsto (x \mapsto t + c, y \mapsto f_y(t)) \mid c \in \mathbb{R}, f_y \colon \mathbb{R}_{\geq 0} \to \mathbb{R}\}$,
  $\mathit{Act}_1^+(l_1) = \{f \colon \mathbb{R}_{\geq 0} \to \mathcal{V}_1^+, t \mapsto (x \mapsto 2t + c, y \mapsto f_y(t)) \mid c \in \mathbb{R}, f_y \colon \mathbb{R}_{\geq 0} \to \mathbb{R}\}$.

It can be easily verified that $\mathcal{H}_1^+$ again satisfies the conditions imposed on hybrid automata.

Example 2.3.2. *Consider the hybrid automaton $\mathcal{H}_2$ from Example 2.1.2. The extension of $\mathcal{H}_2$ with $\{x\}$ yields $\mathcal{H}_2$ again, since x is already contained in the non-controlled variables NCon_2.*

The latter example illustrates that hybrid automata are invariant under extension by their own non-controlled variables.

2.3.2 Syntactic Parallel Composition of Hybrid Automata

Using the extension, we can now construct the syntactic parallel composition of two hybrid automata. Composing hybrid automata restricts their possible behavior, since (some of) the non-determinism regarding the behavior of the environment is resolved. A location of the composed hybrid automaton comprises one location of each component. A variable is considered to be controlled by the composition if it is controlled by either component. All other variables are considered to be non-controlled. Valuations of the joined variable set combine valuations from the components that agree on the values of the shared variables. Equivalently, they can be viewed as valuations of the variables of both extended components. States of the composed hybrid automaton thus consist of a location from each component and a valuation of their combined variables.

When two hybrid automata are executed in parallel, they either both take matching time steps or one of them takes a discrete jump while the other one synchronizes via a matching environmental step. We can take a jump in the composition if and only if we can take a corresponding jump in one of the extended components. The evolution of the variables in some combined location is defined by those activities that are possible in both corresponding locations in the extended components. Similarly, the invariants and initial valuation sets consist of all valuations allowed in both components.

Definition 2.3.3 (Syntactic Parallel Composition of Hybrid Automata)**.** *Let $\mathcal{H}_1, \mathcal{H}_2$ be two hybrid automata that are composable. Let*

$$\mathcal{H}_1^+ = (Loc_1^+, (Con_1^+, NCon_1^+), Inv_1^+, Init_1^+, Edge_1^+, Act_1^+),$$
$$\mathcal{H}_2^+ = (Loc_2^+, (Con_2^+, NCon_2^+), Inv_2^+, Init_2^+, Edge_2^+, Act_2^+)$$

be the extensions of $\mathcal{H}_1, \mathcal{H}_2$ by $V_2 \setminus Con_1$ and $V_1 \setminus Con_2$, respectively. The parallel composition $\mathcal{H}_1 \parallel \mathcal{H}_2 = (Loc, (Con, NCon), Inv, Init, Edge, Act)$ *is the hybrid automaton with*

- $Loc = Loc_1^+ \times Loc_2^+$;
- $Con = Con_1^+ \cup Con_2^+$ *and* $NCon = (NCon_1^+ \cup NCon_2^+) \setminus (Con_1^+ \cup Con_2^+)$;
- $Inv(l_1,l_2) = Inv_1^+(l_1) \cap Inv_2^+(l_2)$ *for all* $(l_1,l_2) \in Loc$;
- $Init(l_1,l_2) = Init_1^+(l_1) \cap Init_2^+(l_2)$ *for all* $(l_1,l_2) \in Loc$;
- $(id, (l_1,l_2), \mu, (l_1',l_2')) \in Edge$ *iff*
  - $l_2 = l_2'$ *and there exists* $(id, l_1, \mu, l_1') \in Edge_1^+$, *or*
  - $l_1 = l_1'$ *and there exists* $(id, l_2, \mu, l_2') \in Edge_2^+$;
- $Act(l_1, l_2) = Act_1^+(l_1) \cap Act_2^+(l_2)$ *for all* $(l_1,l_2) \in Loc$.

In the following, let $\mathcal{H}_1, \mathcal{H}_2$ be two hybrid automata that are composable. The parallel composition again defines a valid hybrid automaton since we defined the extensions in such a way that they are not restricted by the respective new non-controlled variables.

Inv The invariant of each combined location consists of all valuations satisfying the corresponding invariants in both components. Since these do not restrict their non-controlled variables, any variable that is not controlled by either component must still be allowed to take on any value. Hence, the invariants are *NCon*-closed.

Since the invariants of the components have to be closed with respect to their non-controlled variables, it must follow that any invariant of a combined location (l_1, l_2) may only be empty if the invariant of either location l_1 or l_2 is empty. Each component may only impose conditions on its own controlled variables and thus the invariants of two hybrid automata cannot come into conflict.

Init It can be easily seen that all possible initial valuations of a location in the composition must satisfy the location's invariant. The initial valuation set of each combined location is *NCon*-closed again with the same reasoning as for the invariants. As the intersection of two *NCon*-closed initial valuation sets from composable hybrid automata, it is empty if and only if either set is.

Edge Each jump in the composed hybrid automaton corresponds to a jump in one of the components, and thus also to a jump in the extension of that component. Hence, by construction, a jump may only reset the variables controlled by its component, and thus only variables controlled by the composed hybrid automaton. Together with the fact that the location invariants of both hybrid automata must be closed with respect to their respective non-controlled variables, this implies that a jump in one component cannot violate any location invariant of the other component. Thus, the invariant of the combined location in the composed hybrid automaton must always hold after a jump was taken.

Act The activities of a combined location $(l_1,l_2) \in \mathit{Loc}$ must satisfy the ODEs associated with both corresponding locations in the respective extended components. Since a hybrid automaton only specifies ODEs for the evolution of its controlled variables and does not restrict the evolution of the non-controlled variables, the ODE systems of composable hybrid automata can again be combined into a first-order linear ODE system with one equation per controlled variable.

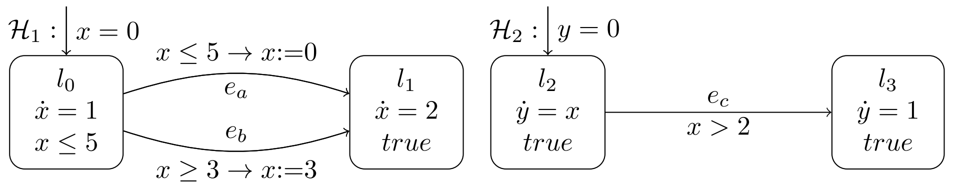
In particular, the set of activities satisfying the combined ODE system can only be empty if one of the activity sets associated with l_1 and l_2 in the respective extended hybrid automaton is empty, since both components allow all possible environmental behavior.

In the following example, we construct the composition of our two running examples for hybrid automata.

Example 2.3.3. *Consider the hybrid automata $\mathcal{H}_1$ and $\mathcal{H}_2$ defined in Example 2.1.1 and Example 2.1.2, respectively. Figure 2.6 depicts $\mathcal{H}_1$ and $\mathcal{H}_2$ as well as their parallel composition $\mathcal{H}_1 \parallel \mathcal{H}_2$, which we will now construct.*

In Example 2.3.1, we defined the extension $\mathcal{H}_1^+$ of $\mathcal{H}_1$ by the set of variables $V_2 \setminus \mathit{Con}_1 = \{y\} \setminus \{x\} = \{y\}$. In Example 2.3.2, we have seen that the extension of $\mathcal{H}_2$ by $V_1 \setminus \mathit{Con}_2 = \{x,y\} \setminus \{y\} = \{x\}$ again yields $\mathcal{H}_2$. Using these extended hybrid automata, we define $\mathcal{H}_1 \parallel \mathcal{H}_2 = (\mathit{Loc}, (\mathit{Con}, \mathit{NCon}), \mathit{Inv}, \mathit{Init}, \mathit{Edge}, \mathit{Act})$ as follows:

- $\mathit{Loc} = \{(l_0,l_2),(l_1,l_2),(l_0,l_3),(l_1,l_3)\}$;
- $V = \{x,y\} = \mathit{Con}$, $\mathit{NCon} = \emptyset$, $\mathcal{V} = \{\nu : \{x,y\} \to \mathbb{R}\}$;
- $\mathit{Inv}(l_0,l_2) = \mathit{Inv}(l_0,l_3) = \{\nu \in \mathcal{V} \mid \nu(x) \leq 5\}$,
  $\mathit{Inv}(l_1,l_2) = \mathit{Inv}(l_1,l_3) = \mathcal{V}$;
- $\mathit{Init}(l_0,l_2) = \{(x \mapsto 0, y \mapsto 0)\}$,
  $\mathit{Init}(l) = \emptyset$ *for all other* $l \in \mathit{Loc}$;
- $\mathit{Edge} = \left\{\begin{matrix} ((l_0,l_2),\mu_a,(l_1,l_2)), & ((l_0,l_3),\mu_a,(l_1,l_3)), \\ ((l_0,l_2),\mu_b,(l_1,l_2)), & ((l_0,l_3),\mu_b,(l_1,l_3)), \\ ((l_0,l_2),\mu_c,(l_1,l_3)), & ((l_1,l_2),\mu_c,(l_1,l_3)) \end{matrix}\right\}$ *with*


$\mathcal{H}_1$: $x = 0$
$x \leq 5 \to x{:=}0$
l_0 $\dot{x} = 1$ $x \leq 5$
e_a
e_b
l_1 $\dot{x} = 2$ $true$
$x \geq 3 \to x{:=}3$
$\mathcal{H}_2$: $y = 0$
l_2 $\dot{y} = x$ $true$
e_c
$x > 2$
l_3 $\dot{y} = 1$ $true$

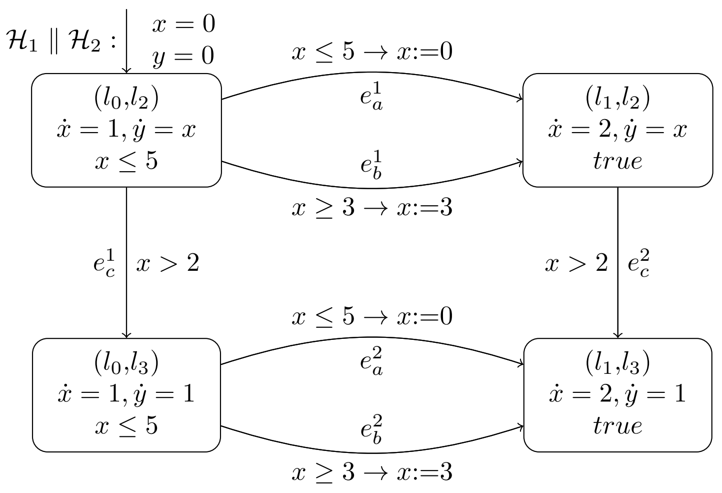

$\mathcal{H}_1 \parallel \mathcal{H}_2$: $x = 0$ $y = 0$
$x \leq 5 \to x{:=}0$
(l_0,l_2) $\dot{x} = 1, \dot{y} = x$ $x \leq 5$
e_a^1
e_b^1
(l_1,l_2) $\dot{x} = 2, \dot{y} = x$ $true$
$x \geq 3 \to x{:=}3$
e_c^1 $x > 2$
$x > 2$ e_c^2
$x \leq 5 \to x{:=}0$
(l_0,l_3) $\dot{x} = 1, \dot{y} = 1$ $x \leq 5$
e_a^2
e_b^2
(l_1,l_3) $\dot{x} = 2, \dot{y} = 1$ $true$
$x \geq 3 \to x{:=}3$

Figure 2.6: Hybrid automata $\mathcal{H}_1$ and $\mathcal{H}_2$ and their composition $\mathcal{H}_1 \parallel \mathcal{H}_2$, as defined in Example 2.3.3. The jump names in the composition consist of the original jump name annotated with an upper index indicating whether it is the first or second "copy".

- $\mu_a = \{(\nu, \nu') \in \mathcal{V}^2 \mid \nu(x) \leq 5 \wedge \nu'(x) = 0 \wedge \nu(y) = \nu'(y)\}$,
- $\mu_b = \{(\nu, \nu') \in \mathcal{V}^2 \mid \nu(x) \geq 3 \wedge \nu'(x) = 3 \wedge \nu(y) = \nu'(y)\}$,
- $\mu_c = \{(\nu, \nu') \in \mathcal{V}^2 \mid \nu(x) > 2 \wedge \nu = \nu'\}$;

- *Act*(l_0, l_2) *contains all solutions to the linear ODE system* $\dot{x} = 1, \dot{y} = x$, *i.e.,* *Act*$(l_0, l_2) = \{f \colon \mathbb{R}_{\geq 0} \to \mathcal{V}, t \mapsto (x \mapsto t + c, y \mapsto \frac{1}{2}t^2 + ct + d) \mid c, d \in \mathbb{R}\}$, *and similarly for the other locations:*
Act$(l_1, l_2) = \{f \colon \mathbb{R}_{\geq 0} \to \mathcal{V}, t \mapsto (x \mapsto 2t + c, y \mapsto t^2 + ct + d) \mid c,d \in \mathbb{R}\}$,
Act$(l_0, l_3) = \{f \colon \mathbb{R}_{\geq 0} \to \mathcal{V}, t \mapsto (x \mapsto t + c, y \mapsto t + d) \mid c,d \in \mathbb{R}\}$,
Act$(l_1, l_3) = \{f \colon \mathbb{R}_{\geq 0} \to \mathcal{V}, t \mapsto (x \mapsto 2t + c, y \mapsto t + d) \mid c,d \in \mathbb{R}\}$.

Note that this is a valid hybrid automaton, which is timelock-free, non-Zeno, and closed. All dependencies on the environment are resolved and the evolution of all variables is uniquely determined for all states.

Even if it is possible to reach a valuation in both components in the same amount of time, the same in not necessarily possible in their composition. This is shown in the following example.

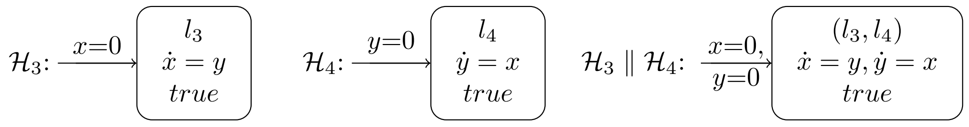


Figure 2.7: Composition of the hybrid automata with coupled differential equations defined in Example 2.3.4.

Example 2.3.4. *Consider the hybrid automata $\mathcal{H}_3$ and $\mathcal{H}_4$ presented in Figure 2.7 with the following formal definitions, where $\mathcal{V} = \{x,y\}$, $\mathcal{V} = \mathcal{V}_V$, and $F = F_V$:*

- $Loc_3 = \{l_3\}$;
- $Con_3 = \{x\}, NCon_3 = \{y\}$;
- $Inv_3(l_3) = \mathcal{V}$;
- $Init_3(l_3) = \{(x \mapsto 0)\}$;
- $Edge_3 = \emptyset$;
- $Act_3(l_3)=\{f \in F | \frac{df(t)(x)}{dt}=f(t)(y)\}$;

- $Loc_4 = \{l_4\}$;
- $Con_4 = \{y\}, NCon_4 = \{x\}$;
- $Inv_4(l_4) = \mathcal{V}$;
- $Init_4(l_4) = \{(y \mapsto 0)\}$;
- $Edge_4 = \emptyset$;
- $Act_4(l_4)=\{f \in F | \frac{df(t)(y)}{dt}=f(t)(x)\}$.

Since neither hybrid automaton contains any jumps, their behavior is fully determined by the continuous evolution of the variables. The evolution of the variables depends on the behavior of the respective other hybrid automaton since the differential equations specifying the activity sets for l_3 and l_4 are coupled.

Their syntactic parallel composition $\mathcal{H}_3 \parallel \mathcal{H}_4$, also depicted in Figure 2.7, is then defined as follows:

- $Loc = \{(l_3, l_4)\}$;
- $Con = \{x,y\}, NCon = \emptyset$;
- $Inv(l_3,l_4) = \mathcal{V}$;
- $Init(l_3,l_4) = \{(x \mapsto 0, y \mapsto 0)\}$;
- $Edge = \emptyset$;
- $Act(l_3,l_4) = \{f \in F \mid \frac{df(t)(x)}{dt} = f(t)(y) \wedge \frac{df(t)(y)}{dt} = f(t)(x)\}$.

The unique activity satisfying the ODE system $\dot{x} = y, \dot{y} = x$ and the initial condition $f(0)(x) = 0, f(0)(y) = 0$ is $f\colon \mathbb{R}_{\geq 0} \to \mathcal{V}, t \mapsto (x \mapsto 0, y \mapsto 0)$. Hence, the initial state $((l_3,l_4), (x \mapsto 0, y \mapsto 0))$ is the only reachable state. In particular, the valuation $(x \mapsto 2, y \mapsto 2)$ is not reachable from the initial state even though it is reachable in both components by a time step of length 2 via the activities

$$f_3\colon \mathbb{R}_{\geq 0} \to \mathcal{V}, t \mapsto (x \mapsto t^2, y \mapsto 2t),$$
$$f_4\colon \mathbb{R}_{\geq 0} \to \mathcal{V}, t \mapsto (x \mapsto 2t, y \mapsto t^2).$$

We see that $f_3 \in Act_3(l_3)$, $f_4 \in Act_4(l_4)$, and that both activities satisfy the composition's initial condition. However, $f_3, f_4 \notin Act(l_3,l_4)$. Hence, neither activity can be used to reach the desired valuation in the composition.

It can be easily verified that init-, reset-, and activity-determinism are preserved under composition. The composition of two closed hybrid automata again yields a closed hybrid automaton, though their extensions with the variables of the other hybrid automaton are of course not closed. If hybrid automata are executed in parallel, they refer to the same “global” time. Hence, if one component forces the other one into time-convergent behavior or deadlock, then time cannot diverge for the first component either.

Chapter 3

Stochastic Hybrid Automata

In this chapter, we extend the hybrid automata introduced in the previous chapter by stochastic behavior. We start with some preliminary notes on probability theory. We assume familiarity with probability theory, especially with the notion of probability spaces (see, e.g., [MU05, Fel91, BK08]). For probability distributions, we distinguish between discrete and continuous distributions as follows.

Definition 3.0.1 (Probability Distribution)**.** *Let $S \subseteq \mathbb{R}$ be a set and $f \colon S \to \mathbb{R}_{\geq 0}$ a function. We define the* support *of f as $support(f) = \{s \in S \mid f(s) > 0\}$ and call f*

- *a* discrete probability distribution *if it holds that $support(f)$ is countable and $\sum_{s \in S} f(s) = 1$, and*
- *a* continuous probability distribution *if it holds that $support(f)$ is uncountable and $\int_{s \in S} f(s)ds = 1$.*

We use Distr to denote the set of all probability distributions.

For simplicity, we always follow the notation for uncountable support in the following and use integrals to calculate probabilities. For discrete distributions these integrals should be replaced by (potentially infinite) sums.

We use $U_{[a,b]}$ to denote the uniform distribution over the interval $[a,b] \subset \mathbb{R}$ with $a,b \in \mathbb{Q}$ and $a < b$, i.e., $U_{[a,b]}(t) = \frac{1}{b-a}$ if $a \leq t \leq b$ and $U_{[a,b]}(t) = 0$ otherwise. Further, we write $c \sim P$ to denote that c is sampled according to the probability distribution P.

Analogous to the previous chapter, we first give the syntax and semantics of stochastic hybrid automata and then define their syntactic parallel composition. Lastly, we compare our modeling language to two approaches that are similar to our language with regard to different aspects.

3.1 Syntax of Stochastic Hybrid Automata

We now extend hybrid automata by stochastic behavior regarding the jump times and the decision between several jumps that can be taken at the same time. All other non-determinism regarding the component itself, i.e., concerning the initial states, continuous evolution of controlled variables, and resets, needs to either be resolved by a scheduler or the choices must be deterministic. Here, we choose the latter and assume the underlying hybrid automaton to be init-, reset-, and activity-deterministic. These properties only require determinism regarding the controlled variables since we again do not want to restrict the behavior of the environment.

Stochastic hybrid automata extend hybrid automata by associating the jumps with stochastic events and weights, which together determine when which jump should be taken. Several jumps can be bound together to model a stochastic event. We define probability distributions for all stochastic events, which are used to determine when a jump should be taken. More specifically, we do not sample a global point in time for each stochastic event, but a combined enabling duration for the jumps associated with the same event. These enabling durations are sampled initially for each event and then each time a jump is taken. In the latter case, we only need to sample a new enabling duration for the event associated with the jump, based on the state reached by taking the jump.

At each location, the outgoing jumps are in a race with each other with respect to which one can be taken first. If several jumps can be taken at the same time, their associated weights are used to make a probabilistic choice.

Our stochastic hybrid automaton model is again based on [Sch22].

Definition 3.1.1 (Syntax of Stochastic Hybrid Automata)**.** *A* stochastic hybrid automaton (SHA) *is a tuple $\mathcal{A} = (\mathcal{H}, \mathit{Lab}, \mathit{Evt}, \mathit{Dur}, \mathit{Wgt})$, where*

- $\mathcal{H} = (\mathit{Loc}, (\mathit{Con}, \mathit{NCon}), \mathit{Inv}, \mathit{Init}, \mathit{Edge}, \mathit{Act})$ *is an init-, reset-, and activity-deterministic hybrid automaton;*
- *Lab is a non-empty set of labels specifying stochastic events;*
- *$\mathit{Evt}\colon \mathit{Edge} \to \mathit{Lab}$ assigns an event to each jump such that all e_1, e_2 with the same source location have $\mathit{Evt}(e_1) \neq \mathit{Evt}(e_2)$;*
- *$\mathit{Dur}\colon \mathit{Lab} \times \Sigma \to \mathit{Distr}$ assigns a probability distribution to each pair in $\mathit{Lab} \times \Sigma$;*
- *$\mathit{Wgt}\colon \mathit{Edge} \to \mathbb{N}_{>0}$ assigns a positive weight to each jump.*

Let in the following $\mathcal{A} = (\mathcal{H}, \mathit{Lab}, \mathit{Evt}, \mathit{Dur}, \mathit{Wgt})$ be a stochastic hybrid automaton. A jump is considered *fireable* if it is enabled and its enabling duration has run out. A formal definition follows in Definition 3.2.1.

Lab It should be noted that the elements of *Lab* are not used for jump synchronization between components, but to allow modeling a stochastic event by several jumps.

Evt At each location, the events associated with the outgoing edges are in a race with respect to which will become fireable first. Since a stochastic event cannot be in a race with itself, we do not allow two edges originating from the same source location to be associated with the same event. This means that only at most one jump can be enabled for each event at the same time at each location.

Dur The probability distributions used to determine the enabling durations are allowed to depend on the state. Hence, the sampling of an enabling duration can depend on the values of the continuous variables and the location. This would, for example, allow us to express that the probability of a car breaking down depends on the age

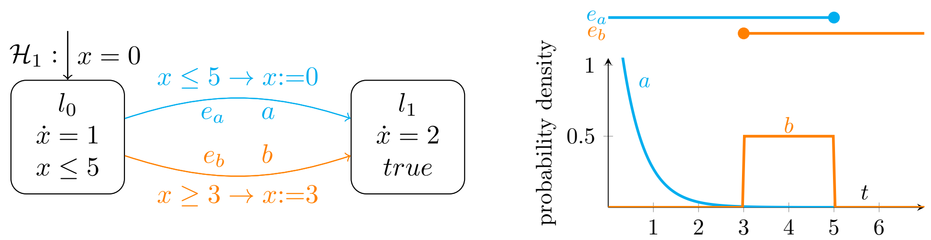


Figure 3.1: Left: Depiction of the hybrid automaton introduced in Example 2.1.1 with a race between jumps e_a and e_b. Right: Above, the enabled intervals of both jumps; below, the probability density functions associated with the jumps, drawn against global time.

and "history" of the car. This is more expressive than just assigning one distribution to each stochastic event. In particular, via the dependency on the location, the probability distributions can even depend on the evolution of the controlled variables. However, the location and the currently enabled jump associated with some event might change before the corresponding sampled enabling duration is reached. Thus, it is not always possible to take all future evolution of the controlled variables into account for the definition of the probability distribution. On the other hand, a dependency on the state might not always be necessary. These considerations show that other design choices for the modeling of the stochastic events might also be sensible.

If there does not exist an outgoing jump with event label r in a location l, it does not matter how to define the probability distribution $Dur(r, (l, \nu))$ for $\nu \in \mathcal{V}$, because this distribution will never be used.

Since reachability is undecidable for hybrid automata [HKPV98], it is in general not possible to determine how long a jump will be enabled or whether a certain enabling duration will be reached. Hence, we cannot avoid assigning a positive probability to enabling durations that cannot be reached. As a result, the probability distributions do not directly give the probabilities of taking a jump after some specified enabling duration. This is illustrated in the following example.

Example 3.1.1. *Consider the race between jumps e_a and e_b in hybrid automaton $\mathcal{H}_1$ from Example 2.1.1. Assume these jumps are associated with stochastic events a and b and an exponential and a uniform distribution, respectively, which do not depend on the state. Figure 3.1 displays $\mathcal{H}_1$ again, as well as both probability density functions against global time and the enabled intervals of both jumps. In the intervals* $[0,3)$ *and* $(5,\infty]$*, only either jump e_a or e_b is enabled, while in the interval* $[3,5]$ *both jumps are enabled. If an enabling duration $t_a < 3$ is sampled for stochastic event a, clearly jump e_a will be taken. On the other hand, if some $t_a > 5$ is sampled, then e_a will never become fireable and instead with probability 1, jump e_b will be taken at some point. Hence, the probability of taking e_a in the interval* $[5,\infty)$ *flows into the probability of taking jump e_b.*

In general, the probability of sampling a too high enabling duration for some jump may flow into the probabilities of taking other jumps or staying in the location forever. In Example 3.1.1, the latter is not possible: Even if l_0's invariant was '*true*', the probability distribution assigned to e_b would ensure that a jump is taken before '$x > 5$'. In some cases, it may not be possible to redistribute the probability at all, which means that the system will deadlock with that probability 1. We discuss how to define deadlocks for stochastic hybrid automata in Section 3.2.2.

Note that this redistribution of probabilities does not mean that the probability distributions are changed, but only that the probabilities are rescaled. This rescaling is not trivial since, as previously mentioned, the current location and the currently enabled jump associated with some stochastic event r may change before the enabling duration sampled for r is reached.

These considerations show that the probability distributions must be chosen carefully, in particular to avoid introducing timelocks.

Wgt The weights can be interpreted as probabilistic priorities over a finite set of choices. If two or more jumps are fireable at the same time, their weights are used to compute probabilities for each jump to be taken. Since this calculation involves dividing by a sum of weights, we only allow strictly positive weights in order to avoid division by 0.

However, the probability of two or more jumps being fireable at the same time is 0 if the corresponding probability distributions determining the enabling durations are continuous. Thus, the weights are only relevant for discrete distributions.

Instead of assigning weights to the jumps directly, we could also assign them to the events, which would in turn induce weights on the jumps.

We now extend the hybrid automata $\mathcal{H}_1$ and $\mathcal{H}_2$ introduced in Chapter 2 to stochastic hybrid automata.

Example 3.1.2. *Extending Example 3.1.1, we define the SHA* $\mathcal{A}_1 = (\mathcal{H}_1, Lab_1, Evt_1, Dur_1, Wgt_1)$ *using the hybrid automaton* $\mathcal{H}_1$ *from Example 2.1.1 and*

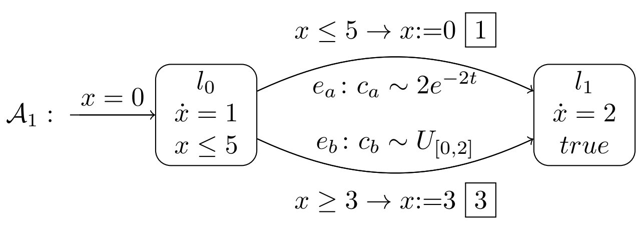
- $Lab_1 = \{a, b\}$;
- $Evt_1(e_a) = a,\ Evt_1(e_b) = b$;
- $Dur_1(a,(l_0,\nu))(t) = 2e^{-2t},\ Dur_1(b,(l_0,\nu)) = U_{[0,2]}$ *for all* $\nu \in \mathcal{V}$;
- $Wgt_1(e_a) = 1,\ Wgt_1(e_b) = 3$.

Figure 3.2 presents a graphical representation of $\mathcal{A}_1$.

Note that the probability distributions are chosen in such a way that with probability 1, one of the jumps must become fireable before the location invariant of l_0 *is violated. However, this is not the case for all possible probability distributions. As we will see in Section 3.2.2, this means that a stochastic hybrid automaton might have a timelock even though the underlying hybrid automaton is timelock-free.*

Example 3.1.3. *Figure 3.3 depicts the stochastic hybrid automaton* $\mathcal{A}_2 = (\mathcal{H}_2, Lab_2, Evt_2, Dur_2, Wgt_2)$ *that extends the hybrid automaton* $\mathcal{H}_2$ *from Example 2.1.2 by*

- $Lab_2 = \{c\}$;
- $Evt_2(e_c) = c$;
- $Dur_2(c,(l_2,\nu))(t) = 2e^{-2t}$;
- $Wgt_2(e_c) = 1$.



$x \leq 5 \to x{:=}0$ $\boxed{1}$
$\mathcal{A}_1$:
$x = 0$
l_0
$\dot{x} = 1$
$x \leq 5$
$e_a\colon c_a \sim 2e^{-2t}$
$e_b\colon c_b \sim U_{[0,2]}$
l_1
$\dot{x} = 2$
true
$x \geq 3 \to x{:=}3$ $\boxed{3}$

Figure 3.2: Depiction of the stochastic hybrid automaton $\mathcal{A}_1$ defined in Example 3.1.2. Weights are displayed in boxes.

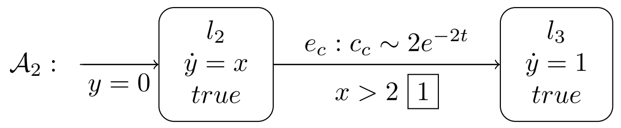


Figure 3.3: Stochastic hybrid automaton $\mathcal{A}_2$ with dependencies on the environment, as defined in Example 3.1.3.

3.2 Semantics of Stochastic Hybrid Automata

When executing stochastic hybrid automata, we need to keep track of the state of the underlying hybrid automaton and when to take jumps. To this end, we define $V_{Lab} = \{c_r \mid r \in Lab\}$ to be a set of variables associated with the stochastic event labels *Lab*. The set of valuations for V_{Lab} is denoted by $\mathcal{V}_{Lab}$. Each variable c_r is initialized with some sampled enabling duration and runs down linearly as long as a jump associated with r is enabled and c_r is non-negative. Hence, a variable c_r can be interpreted as a clock that runs backwards [1]. A jump can only fire when it is enabled and the clock of the associated stochastic event reached 0.

In the following, we consider *states* $\hat{\sigma}$ of SHA $\mathcal{A}$ to be elements of $\Sigma \times \mathcal{V}_{Lab} = Loc \times \mathcal{V} \times \mathcal{V}_{Lab}$. We now give the semantics of stochastic hybrid automata by defining execution steps between these states, which restrict the execution steps possible in the underlying hybrid automaton. A discrete step can be taken from a state $\hat{\sigma}$ if and only if some jump is fireable.

Definition 3.2.1 (Fireable)**.** *Let $\mathcal{A} = (\mathcal{H}, Lab, Evt, Dur, Wgt)$ be a SHA. For a jump $e = (id, l, \mu, l') \in Edge$ and a state $(l, \nu, \nu_{Lab}) \in \Sigma \times \mathcal{V}_{Lab}$ of $\mathcal{A}$, we define fireable$((l, \nu, \nu_{Lab}), e) = true$ iff e is enabled in state (l, ν) of $\mathcal{H}$ and $\nu_{Lab}(c_{Evt(e)}) = 0$.*

The semantics for stochastic hybrid automata needs to ensure that no time can pass between a jump becoming fireable and being taken. Thus, none of the clocks $c_r \in \mathcal{V}_{Lab}$ should become negative during a time step. After a time step, each clock c_r needs to be decreased by the total amount of time that any associated jump was enabled during the time step. For simplicity, we require that the enabledness of the jumps may not change during time steps, but only at the endpoints of time steps. We define a predicate *invEn* to express that, during some time step, the enabledness of all jumps stays invariant and all clocks are reset appropriately afterwards.

[1]We could alternatively also use “normal” clocks whose values increase linearly until the sampled enabling duration is reached. However, since this would require us to store the sampled enabling durations separately, backwards running clocks are the more convenient choice here.

Definition 3.2.2 (*invEn*). *Let $\mathcal{A} = (\mathcal{H}, Lab, Evt, Dur, Wgt)$ be a SHA. For $l \in Loc$, $\nu \in \mathcal{V}$, $\nu_{Lab}, \nu'_{Lab} \in \mathcal{V}_{Lab}$, $t \in \mathbb{R}_{\geq 0}$ and $f \in Act(l)$ such that $f(0) = \nu$, we define $invEn(l, \nu, \nu_{Lab}, \nu'_{Lab}, t, f) = true$ iff for all $r \in Lab$ it holds that*

- *either there exists some $e = (id, l,\mu,l') \in Edge$ with $r = Evt(e)$ such that*
  1. *$\forall t' \in (0,t).\ \exists \nu', \nu'' \in \mathcal{V}.\ (l,\nu) \xrightarrow{t',f} (l,\nu') \wedge (l,\nu') \xrightarrow{e} (l',\nu'')$, and*
  2. *$\nu'_{Lab}(c_r) = \nu_{Lab}(c_r) - t \geq 0$,*
- *or for all $e = (id, l,\mu,l') \in Edge$ with $r = Evt(e)$ it holds that*
  1. *$\forall t' \in (0,t).\ \neg\Big(\exists \nu', \nu'' \in \mathcal{V}.\ (l,\nu) \xrightarrow{t',f} (l,\nu') \wedge (l,\nu') \xrightarrow{e} (l',\nu'')\Big)$, and*
  2. *$\nu'_{Lab}(c_r) = \nu_{Lab}(c_r)$.*

As mentioned above, the predicate *invEn* expresses two aspects. Firstly, it checks that all jumps are invariantly en- or disabled in the open time interval $(0,t)$, i.e., for all jumps e either e can be taken from all states reached by time steps of length $t' \in (0,t)$ or from none of these states. Recall that at each location all outgoing jumps must be associated with different stochastic events. Hence, for each stochastic event, at most one associated jump can be enabled at all times. Note that the interval must be open in order to allow for changes of enabledness at the endpoints. Secondly, it ensures that the clocks associated with the stochastic events are handled correctly: If a jump associated with some event r is enabled in the interval $(0,t)$, the value of clock c_r is decreased accordingly, until 0 is reached. Otherwise, the value remains unchanged.

Using the predicates *invEn* and *fireable*, we can now give the semantics of stochastic hybrid automata. As for hybrid automata, we define an operational semantics consisting of rules for time, discrete, and environmental steps.

Definition 3.2.3 (Semantics of SHA). *Let $\mathcal{A} = (\mathcal{H}, Lab, Evt, Dur, Wgt)$ be a SHA. The semantics of $\mathcal{A}$ is given by an operational semantics consisting of three rules:*

$$\frac{(l,\nu) \xrightarrow{t,f} (l,\nu') \qquad invEn(l, \nu, \nu_{Lab}, \nu'_{Lab}, t, f)}{(l, \nu, \nu_{Lab}) \xRightarrow{t,f} (l, \nu', \nu'_{Lab})}\ \mathtt{Rule}_{time}$$

$$\frac{\begin{array}{c}(l,\nu) \xrightarrow{e} (l',\nu') \qquad fireable((l, \nu, \nu_{Lab}), e) \\ Evt(e) = r \qquad u \in support(Dur(r,(l',\nu')))\end{array}}{(l, \nu, \nu_{Lab}) \xRightarrow{e} (l',\nu', \nu_{Lab}[c_r \mapsto u])}\ \mathtt{Rule}_{discrete}$$

$$\frac{(l,\nu) \xrightarrow{\tau} (l,\nu')}{(l,\nu, \nu_{Lab}) \xRightarrow{\tau} (l,\nu', \nu_{Lab})}\ \mathtt{Rule}_{environment}$$

An execution step

$$\Rightarrow = \left(\bigcup_{t \in \mathbb{R}_{\geq 0}, f \in F} \xRightarrow{t,f}\right) \cup \left(\bigcup_{e \in Edge} \xRightarrow{e}\right) \cup \xRightarrow{\tau}$$

of $\mathcal{A}$ is either a time step or a discrete step or an environmental step. As for hybrid automata, we use infix notation for the relation $\Rightarrow$.

A path *π of $\mathcal{A}$ is a (finite or infinite) sequence of states of $\mathcal{A}$ connected by execution steps $\hat{\sigma}_0 \Rightarrow \hat{\sigma}_1 \Rightarrow \hat{\sigma}_2 \Rightarrow \ldots$ such that $\nu_0 \in Inv(l_0)$ for $\hat{\sigma}_0 = (l_0, \nu_0, \nu_{0,Lab})$.*

As explained above, we only allow time steps during which the enabledness of jumps does not change and none of the clocks $c_r \in \mathcal{V}_{Lab}$ become negative. We do not require these time steps to be maximal though, so the enabledness of jumps does not always have to change between time steps. When we check whether the enabledness stays invariant during a time step, we need to know which activity was used to make the corresponding time step in the hybrid automaton, as there does not necessarily exist a unique activity $f \in Act(l)$ with $f(0) = \nu$ for each state $(l,\nu) \in \Sigma$. Additionally, it is advantageous for algorithmic aspects to cut the time steps into intervals during which the enabledness of all jumps stays invariant.

If we take a discrete step $(l, \nu, \nu_{Lab}) \overset{e}{\Rightarrow} (l',\nu', \nu_{Lab}[c_r \mapsto u])$, we have to sample a new enabling duration u for the corresponding event $r = Evt(e)$ according to the probability distribution $Dur(r, (l', \nu'))$ and reset c_r to this value.

An environmental step is possible if and only if a corresponding step is possible in the underlying hybrid automaton. The clocks associated with the stochastic events are not changed since neither time is passing nor some jump associated with these events is taken.

Example 3.2.1. *Consider the stochastic hybrid automaton $\mathcal{A}_1$ defined in Example 3.1.2. Assume we sampled enabling durations $t_a = 4$ and $t_b = 1$, then the initial state is $\hat{\sigma}_0 = (l_0, (x \mapsto 0), (c_a \mapsto 4, c_b \mapsto 1))$. From $\hat{\sigma}_0$, we can take time steps of length at most 3, since the enabledness of e_b changes at $x = 3$.*

At state $(l_0, (x \mapsto 4), (c_a \mapsto 0, c_b \mapsto 0))$, both e_a and e_b are fireable, so we use the associated weights to make a probabilistic choice: With probability $\frac{1}{4}$, e_a is taken and with probability $\frac{3}{4}$, e_b is taken. In $\mathcal{A}_1$, no environmental steps are possible since none are possible in the underlying closed hybrid automaton $\mathcal{H}_1$.

Stochastic hybrid automata allow a more fine-grained analysis than simply asking whether a state is reachable. Instead, we are now interested in how probable it is to reach a state. To calculate these probabilities, we need to know how probable it is to take the paths leading to a specific state. However, it does not make sense to measure the probability of single paths: If a jump is associated with a continuous probability distribution, the probability of taking this jump after any single enabling duration is 0. Hence, the probability of taking a single path is 0 in general. Instead of calculating the probabilities of single paths, we therefore calculate the probability of certain sets of paths. Using these calculations, we then elaborate on how to calculate the probability of reaching a certain state, and extend the notions of time-convergence, Zeno paths, timelock, and deadlock to stochastic hybrid automata.

3.2.1 Path Probabilities

In general, it does not make sense to calculate the probabilities of single paths, but only of sets of paths, as explained above. Here, we choose to measure the probabilities of *symbolic paths*, which correspond to sequences of jumps where the jump times are not fixed.

Definition 3.2.4 (Symbolic Path)**.** *Let $\mathcal{A} = (\mathcal{H}, Lab, Evt, Dur, Wgt)$ be a SHA. A* finite symbolic path *π of $\mathcal{A}$ is a tuple $((l,\nu), e_1 \ldots e_n)$ with $n \geq 0$, $(l, \nu) \in \Sigma$, $e_i \in Edge$ for all $1 \leq i \leq n$ and $e_1 = (id, l, \mu, l')$ for some $\mu \subseteq \mathcal{V}^2, l' \in Loc$.*

A symbolic path $(\sigma, e_1 \ldots e_n)$ of a stochastic hybrid automaton $\mathcal{A}$ corresponds to the set of all paths starting in σ whose first n jumps are e_1,...,e_n. For $n = 0$, this is simply the set of all paths starting in σ. Note that if there is some $i < n$ such that the target location of jump e_i does not coincide with the source location of jump e_{i+1}, then the set of paths corresponding to this symbolic path is empty.

It might alternatively also make sense to define symbolic paths over event labels instead of jumps, depending on what one is interested in.

Sometimes it might be desirable to restrict the time durations between two consecutive jumps by time intervals. This is for example necessary to measure more complex properties like Zenoness. For stochastic timed automata, Bertrand et al. [BBB$^+$14] extended their probability measure to *constrained* symbolic paths. We expect that their approach can be adapted to our situation.

Using symbolic paths, we can now properly define a probability space $(\Omega, \mathcal{F}, P)$ over the paths of a SHA. As sample space Ω we take the set of all paths of the SHA and as event[1] space $\mathcal{F}$ the set of all finite symbolic paths and their countable unions and complements, which corresponds to a set of subsets of Ω. The complement of a symbolic path $(\sigma, e_1 \dots e_n)$ of a SHA $\mathcal{A}$ consists of all paths starting at σ that do not follow the jump sequence $e_1 \dots e_n$. The countable union and complement of symbolic paths do not necessarily again correspond to symbolic paths, but can be expressed as the countable union of disjoint sets and the set difference of symbolic paths where one is a subset of the other, and thus they are measurable. For example, the complement of a symbolic path $(\sigma, e_1 \dots e_n)$ of a SHA $\mathcal{A}$ can be expressed as the set of all paths corresponding to $(\sigma,)$ without all paths captured by $(\sigma, e_1 \dots e_n)$, or equivalently as the set of all paths starting at σ and not taking any jump, joined with the set of all paths taking a jump other than e_1 first, joined with the set of all paths not taking e_2 after having taken e_1, and so on.

The remainder of this section is dedicated to the probability function P measuring symbolic paths. To calculate the probability of a symbolic path, we integrate over all possible enabling durations for all stochastic events. For each combination, we take time steps until we reach a state where a jump becomes fireable. If that jump is the next jump in the jump sequence of the symbolic path, we recursively calculate the probability of the remaining path. After the last jump e_n was taken, we assume that the probability of the remaining path is 1.

This assumption only holds if at each state, the probability of taking an infinite path is 1, which is not always the case. Recall that hybrid automata may contain deadlocked states, from which no path can be taken. Additionally, a stochastic hybrid automaton has a deadlock if at some state the probability of sampling a too high enabling duration for some jump can neither be redistributed to some other jump nor to staying in the location (see Section 3.1). We discuss how to formally define deadlocks for stochastic hybrid automata in Section 3.2.2.

According to our definition of deadlock, the probability of taking an infinite path is 1 for all states if and only if the probability of reaching a deadlocked state is 0. Thus, we can only define path probabilities for deadlock-free stochastic hybrid automata. But to check whether this is the case, we already need to calculate the probabilities of paths, so we need to ensure the assumption in some other way. Here, we choose to only consider *invariant-free* stochastic hybrid automata, which only allow the location invariant '*true*'. Then, the underlying hybrid automaton is deadlock-free and if we sample an enabling duration for some jump that will never be reached, we can stay in the location forever instead (or possibly take some other jump). Hence, invariant-free stochastic hybrid automata cannot contain deadlocked states.

[1]Note that "event" does not refer to the stochastic events associated with the jumps here.

Definition 3.2.5 (Invariant-free SHA)**.** *A SHA $\mathcal{A} = (\mathcal{H}, Lab, Evt, Dur, Wgt)$ is called* invariant-free *if for all locations $l \in Loc$ we have $Inv(l) = \mathcal{V}$.*

It would be desirable to find less strict solutions to this problem in future work. One possibility might be to add a kind of forced jump to each location, which allows to transition to a sink location if the boundary of the invariant is hit and the direction of the continuous flow "points out of" the invariant. Thus, the system does not become deadlocked but instead stays in the sink location forever. We discuss how forced jumps could be included in our model in Section 4.2.

In order to define the path probabilities, we also need to resolve all non-determinism regarding the environmental behavior. Hence, we now require SHA to be *closed*.

Definition 3.2.6 (Closedness)**.** *A SHA $\mathcal{A} = (\mathcal{H}, Lab, Evt, Dur, Wgt)$ is called* closed *if $\mathcal{H}$ is closed.*

Recall that for a closed hybrid automaton, the evolution of all variables is uniquely determined at each state and that for stochastic hybrid automata, we only allow for time steps during which the enabledness of jumps stays invariant and the values of all clocks are non-negative. Hence, for each state $\hat{\sigma}$ of a closed stochastic hybrid automaton, there is a unique maximal (possibly infinite) t such that a time step of length t can be taken from $\hat{\sigma}$. If this maximal time duration t is finite, the unique state reached by this maximal time step is called the *maximal time successor* of $\hat{\sigma}$. Note that, in general, we cannot decide whether the maximal time duration is finite, since reachability is undecidable for hybrid automata [HKPV98].

Definition 3.2.7 (Maximal Time Successor)**.** *Let $\mathcal{A} = (\mathcal{H}, Lab, Evt, Dur, Wgt)$ be a closed, invariant-free SHA. For a state $\hat{\sigma} = (l, \nu, \nu_{Lab})$ and the unique activity $f \in Act(l)$ with $f(0) = \nu$, we define $t_m(\hat{\sigma}) = \sup\{t \mid \exists \hat{\sigma}'.\ \hat{\sigma} \xrightarrow{t,f} \hat{\sigma}'\}$. If $t_m(\hat{\sigma}) < \infty$, the* maximal time successor $ts(\hat{\sigma})$ *of $\hat{\sigma}$ is the unique state with $\hat{\sigma} \xrightarrow{t_m(\hat{\sigma}),f} ts(\hat{\sigma})$.*

When defining the duration of the maximal time step $t_m(\hat{\sigma})$, we take the supremum instead of the maximum in order to ensure that $t_m(\hat{\sigma})$ is defined in cases where the enabledness of jumps will never change. For example, in location l_1 of $\mathcal{A}_1$ from Example 3.1.2 (depicted in Figure 3.2), we can take arbitrarily long time steps without changing the enabledness of any jump originating from l_1 since l_1 has no outgoing jumps.

We claim that the supremum also yields the correct result if the maximal possible time step is bounded. Assume a time step of length $t_m(\hat{\sigma}) = \sup\{t \mid \exists \hat{\sigma}'.\ \hat{\sigma} \xrightarrow{t,f} \hat{\sigma}'\}$ is not possible from $\hat{\sigma}$, only time steps of length $t_m(\hat{\sigma}) - \epsilon$ for arbitrarily small $\epsilon > 0$. This can only happen if the invariant of the location of $\hat{\sigma}$ is an open set (e.g. for '$x < 2$'), making it impossible to stay in the location for the full time $t_m(\hat{\sigma})$. We claim that this implies that $\hat{\sigma}$ is timelocked. It is not possible to stay in the current location forever, since the invariant will become violated. But it is also not possible to take some jump: If some clock c_r reached 0 before the end of the time step $t_m(\hat{\sigma})$, then the maximal time step would be smaller than $t_m(\hat{\sigma})$ since we only allow time steps such that no jump changes its enabledness and no clock runs out. Hence, for invariant-free (and thus timelock-free) SHA, we can assume that it is always possible to stay in a location for the full time step of length $t_m(\hat{\sigma}) < \infty$.

Using the notion of maximal time successors, we can now calculate the probabilities of symbolic paths for closed, invariant-free SHA. Note again that for discrete distributions, the integrals should be replaced by sums.

Definition 3.2.8 (Symbolic Path Probabilities for SHA)**.** *Let $\mathcal{A} = (\mathcal{H}, Lab, Evt, Dur, Wgt)$ be a closed, invariant-free SHA with $Lab = \{r_1, \dots, r_k\}$. Let further $\pi = ((l,\nu), e_1 \dots e_n)$ be a symbolic path of $\mathcal{A}$. The probability of π is 1 for $n = 0$ and otherwise*

$$P(\pi) = \int_{t_1=0}^{\infty} Dur(r_1,(l,\nu))(t_1) \cdot \ldots \cdot \int_{t_k=0}^{\infty} Dur(r_k,(l,\nu))(t_k) \cdot P'(\hat{\sigma}, e_1 \dots e_n)\ dt_k \dots dt_1,$$

where $\hat{\sigma} = (l,\nu, \nu_{Lab})$ with $\nu_{Lab}(c_{r_i}) = t_i$ for all i and

$$P'(\hat{\sigma}, e_1 \dots e_n) = \begin{cases} W \cdot Step & \text{if } fireable(\hat{\sigma}, e_1) \\ 0 & \text{if } \neg fireable(\hat{\sigma}, e_1) \wedge \exists e \in Edge.\ fireable(\hat{\sigma}, e) \\ P'(ts(\hat{\sigma}), e_1 \dots e_n) & \text{if } t_m(\hat{\sigma}) < \infty \wedge \forall e \in Edge.\ \neg fireable(\hat{\sigma},e) \\ 0 & \text{if } t_m(\hat{\sigma}) = \infty \wedge \forall e \in Edge.\ \neg fireable(\hat{\sigma},e), \end{cases}$$

where $W = \frac{Wgt(e_1)}{\sum_{e \in Edge, fireable(\hat{\sigma},e)} Wgt(e)}$ and

$$Step = \begin{cases} 1 & \text{if } n = 1 \\ \int_{t=0}^{\infty} Dur(r,(l',\nu'))(t) \cdot P'((l', \nu', \nu_{Lab}[c_r \mapsto t]), e_2 \dots e_n)\ dt & \text{if } n > 1, \end{cases}$$

where $r = Evt(e_1)$, $e_1 = (l, \mu, l')$ and ν' is the unique valuation such that $(\nu, \nu') \in \mu$.

By assumption, the probability of taking any path from state σ of SHA $\mathcal{A}$ is 1. To compute the probability of a symbolic path $\pi = ((l, \nu), e_1 \dots e_n)$ of $\mathcal{A}$ for $n \geq 1$, we first consider all possible enabling durations for the jumps and with which probability they occur. Each possible combination yields a valuation $\nu_{Lab} \in \mathcal{V}_{Lab}$ and corresponding state $\hat{\sigma} = (l, \nu, \nu_{Lab})$. In state $\hat{\sigma}$, we now distinguish four cases:

1. If e_1 is already fireable in $\hat{\sigma}$, we use the jump weights of all currently fireable jumps to calculate the probability with which e_1 is taken. If e_1 is not the last jump of the symbolic path, we recursively calculate the probability for taking the remaining jumps $e_2, \dots, e_n$. For this, we need to reset the enabling duration for e_1 based on the state reached in $\mathcal{H}$ by taking e_1.

2. If e_1 is not fireable but some other jump is, that other jump will be taken. Thus, the probability of taking e_1 is 0. This does not mean that the probability of the symbolic path π is 0, since it might still be possible to take the specified jump sequence for another valuation ν_{Lab} of the random clocks.

3. If currently no jump is fireable but, in finite time, the enabledness of some jump will change or some jump will become fireable, then we recursively calculate the probability of taking jumps $e_1, \dots, e_n$ from the maximal time successor of the current state.

4. If currently no jump is fireable and no jump will ever change enabledness or become fireable, then the probability of the path π is 0 for the current valuation ν_{Lab} of the random clocks.

Let us first illustrate the calculation of the probability of a symbolic path with an example. Afterwards, we discuss under which conditions the calculation terminates, how the calculation could be refined, and how it can be used to calculate the probability of reaching a certain state.

Example 3.2.2. *We define $\mathcal{A}'_1$ to be the invariant-free version of the SHA $\mathcal{A}_1$ defined in Example 3.1.2 (Figure 3.2), i.e., the only difference between the two SHA is that all locations of $\mathcal{A}'_1$ have the invariant true. The underlying hybrid automaton $\mathcal{H}'_1$ has the unique initial state $\sigma_0 = (l_0, \nu_0)$ with $\nu_0(x) = 0$. The probability of the symbolic path (σ_0, e_b) of $\mathcal{A}'_1$ can be calculated as follows:*

$$P(\sigma_0, e_b) = \int_{t_a=0}^{\infty} Dur(a, \sigma_0)(t_a) \cdot \int_{t_b=0}^{\infty} Dur(b, \sigma_0)(t_b) \cdot P'((l_0, \nu_0, \nu_{Lab}), e_b)\ dt_b dt_a,$$

where $\nu_{Lab} = (c_a \mapsto t_a, c_b \mapsto t_b)$. Let $\hat{\sigma}_0 = (l_0, \nu_0, \nu_{Lab})$. The depiction of the enabled intervals of the two jumps in Figure 3.1 might be helpful for the following considerations. The support of $Dur(b,\sigma_0)(t_b)$ is only $[0,2]$, so the probability of sampling $t_b > 2$ is 0. Further, we observe that if either $t_a < 3$ or $3 \leq t_a \leq 5 \wedge t_b > t_a - 3$, then edge e_a will become fireable before jump e_b, so the probability of taking edge e_b is 0 in these cases. The probability that both edges become fireable at the same time is 0, since the associated probability distributions are continuous. We therefore omit this case in the following considerations.

We now split the integrals at the points where the jumps change enabledness and apply these observations, so $P(\sigma_0, e_b) =$

$$\begin{aligned}
&\left.\int_{t_a=0}^{3} Dur(a, \sigma_0)(t_a) \cdot \int_{t_b=0}^{2} Dur(b, \sigma_0)(t_b) \cdot P'(\hat{\sigma}_0, e_b)\ dt_b dt_a \right\}{=}0\\
&+ \int_{t_a=3}^{5} Dur(a, \sigma_0)(t_a) \cdot \int_{t_b=0}^{2} Dur(b, \sigma_0)(t_b) \cdot P'(\hat{\sigma}_0, e_b)\ dt_b dt_a\\
&+ \int_{t_a=5}^{\infty} Dur(a, \sigma_0)(t_a) \cdot \int_{t_b=0}^{2} Dur(b, \sigma_0)(t_b) \cdot P'(\hat{\sigma}_0, e_b)\ dt_b dt_a\\
= &\int_{t_a=3}^{5} Dur(a, \sigma_0)(t_a) \cdot \int_{t_b=0}^{t_a-3} Dur(b, \sigma_0)(t_b) \cdot P'(\hat{\sigma}_0, e_b)\ dt_b dt_a\\
&+ \left.\int_{t_a=3}^{5} Dur(a, \sigma_0)(t_a) \cdot \int_{t_b=t_a-3}^{2} Dur(b, \sigma_0)(t_b) \cdot P'(\hat{\sigma}_0, e_b)\ dt_b dt_a \right\}{=}0\\
&+ \int_{t_a=5}^{\infty} Dur(a, \sigma_0)(t_a) \cdot \int_{t_b=0}^{2} Dur(b, \sigma_0)(t_b) \cdot P'(\hat{\sigma}_0, e_b)\ dt_b dt_a.
\end{aligned}$$

Jump e_a is only enabled in the interval $[0,5]$ and thus not for the whole support of $Dur(a, \sigma_0)$. The probability of sampling an enabling duration $t_a > 5$ for e_a is therefore redistributed: If we sample some $t_a > 5$, we take jump e_b with probability 1.

Further, if $3 \leq t_a \leq 5$ and $0 \leq t_b < t_a - 3$, then jump e_a will become fireable after e_b. At state σ_0, no jump is fireable, so we take the maximal time step, which is of length 3. If $t_b = 0 < t_a - 3$, then jump e_a is fireable at state $ts(\sigma_0)$. Since this is just one discrete point in a continuous probability distribution, this case is negligible. Otherwise, we take the maximal time step from $ts(\sigma_0)$, which leads us to the state $\hat{\sigma}' = ts(ts(\sigma_0))$ where e_a becomes fireable.

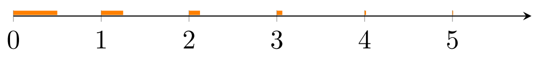


Figure 3.4: Illustration of the guard $g_e = \{\nu \in \mathcal{V} \mid \exists n \in \mathbb{N}.\, \nu(x) \in [n, n + \frac{1}{2^n}]\}$ on the interval [0,6). Values x satisfying the guard are marked in orange. If this guard was combined with an activity f with $f(t)(x) = t$, our probability calculation would not terminate.

Hence, $P(\sigma_0, e_b) =$

$$\begin{aligned}
&\int_{t_a=3}^{5} \mathit{Dur}(a,\sigma_0)(t_a) \cdot \int_{t_b=0}^{t_a-3} \mathit{Dur}(b,\sigma_0)(t_b) \cdot P'(\hat{\sigma}_0, e_b)\ dt_b dt_a \\
&\quad + \int_{t_a=5}^{\infty} \mathit{Dur}(a,\sigma_0)(t_a)\ dt_a \\
&= \int_{t_a=3}^{5} \mathit{Dur}(a,\sigma_0)(t_a) \cdot \int_{t_b=0}^{t_a-3} \mathit{Dur}(b,\sigma_0)(t_b) \cdot P'(ts(ts(\sigma_0)), e_b)\ dt_b dt_a + \frac{1}{e^{10}} \\
&= \int_{t_a=3}^{5} \mathit{Dur}(a,\sigma_0)(t_a) \cdot \int_{t_b=0}^{t_a-3} \mathit{Dur}(b,\sigma_0)(t_b) \cdot W \cdot \mathit{Step}\ dt_b dt_a + \frac{1}{e^{10}} \\
&= \int_{t_a=3}^{5} \mathit{Dur}(a,\sigma_0)(t_a) \cdot \int_{t_b=0}^{t_a-3} 0.5 \cdot \frac{\mathit{Wgt}(e_b)}{\mathit{Wgt}(e_b)} \cdot 1\ dt_b dt_a + \frac{1}{e^{10}} \\
&\approx 0.000619688 + 0.000045400 = 0.000665088.
\end{aligned}$$

Termination Our calculation terminates under the condition that for each reachable state $\hat{\sigma}$, either the length of the maximal time step $t_m(\hat{\sigma})$ is unbounded, or after finitely many time steps we will take a discrete jump. Otherwise, we can have a time-divergent path where the enabledness of some jump changes infinitely often and the enabled intervals become progressively smaller, such that the total enabling duration is finite. For example, assume some location has an outgoing jump e with a guard of the form $g_e = \{\nu \in \mathcal{V} \mid \exists n \in \mathbb{N}.\, \nu(x) \in [n, n + \frac{1}{2^n}]\}$, which is illustrated in Figure 3.4, and assume that the variable x evolves linearly. Then the enabledness of this jump would change infinitely often, but the total enabling duration would be finite since $\sum_{n=0}^{\infty} n + \frac{1}{2^n} - n = 2$.

If the sampled enabling duration for such a jump is larger than its total enabling duration, the associated clock would converge to some finite $c \in \mathbb{R}_{\geq 0}$. The calculation does not terminate if and only if one of the clocks

Note that such behavior may also be caused by an activity with a variable x evolving as depicted in Figure 3.5. The displayed function has infinitely many roots in the interval [0,1] and the distance between those roots progressively decreases. Consider a location l with the depicted evolution of x, which has only one outgoing jump guarded by '$x > 0$'. Then, any path π starting at $(l, x \mapsto 0)$ with $\mathit{ExecTime}(\pi) \geq 1$ must contain infinitely many time steps. Hence, our probability calculation would not terminate if it was possible to sample an enabling duration for the jump that could only be reached by a path π with $\mathit{ExecTime}(\pi) \geq 1$. However, we do not expect combinations of activities and guards as presented in these two examples to appear in models of real-life hybrid systems.

One might expect that allowing arbitrary time steps instead of invariantly enabled ones in the semantics would solve the underlying problem here. For arbitrary time

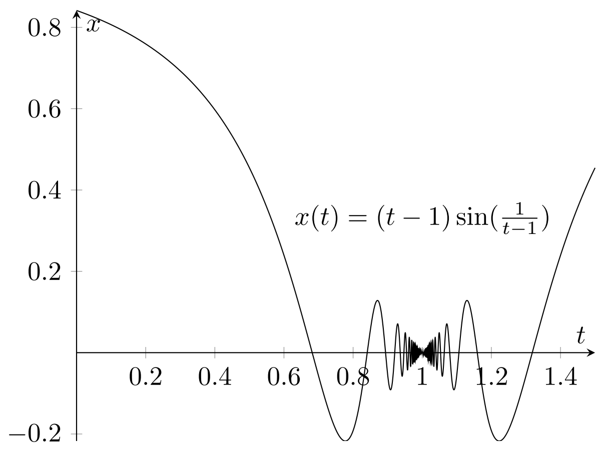


Figure 3.5: Example for an activity for a variable x, which would prevent the calculation from terminating if combined with a guard of the form '$x > 0$'.

steps, we would need to keep track of how long each jump was enabled and appropriately reset the clocks $\mathcal{V}_{Lab}$. In particular, the semantics should still ensure that time steps are only possible as long as no jump becomes fireable. In the probability calculation, if no jump is currently fireable, we would then need to find the longest time step possible that would reach a state where a jump becomes fireable. However, we do not see how this could be done in such a way that it always terminates in cases as described above.

Possible Refinement Sometimes we might be interested in the probability of taking a certain sequence of jumps and then staying in the last location forever, as opposed to taking at least one more jump. We can refine our probability calculation by measuring this probability as follows. For some symbolic path $\pi = (\sigma, e_1 \dots e_n)$ of a SHA, the probability of taking another jump after having taken the specified jumps is given by the sum of the probabilities of all symbolic paths of the form $(\sigma, e_1 \dots e_n e_{n+1})$ for $e_{n+1} \in \mathit{Edge}$. Subtracting this probability from the probability of π consequently yields the probability of staying in the last location forever.

Probability of Reaching a State Our original motivation for defining the probabilities of paths was to determine how probable it is to reach a certain state. Indeed, the probability of reaching a state (l, ν) can be defined as a countable sum of probabilities of symbolic paths. Reaching the state (l,ν) can be equivalently expressed as reaching l and then taking an added urgent jump e_ν leading to some sink location, which has to be taken immediately if the current valuation is ν. Then, the probability of reaching (l, ν) is equivalent to the sum of the probabilities of all symbolic paths $(\hat{\sigma}_0, e_1 \dots e_n e_\nu)$ starting at the initial state $\hat{\sigma}_0$ and then taking some jump sequence ending with e_ν.

3.2.2 Zenoness, Timelock, and Deadlock

Time-convergent and time-divergent paths for stochastic hybrid automata are defined analogously to hybrid automata (see Section 2.2.1).

Definition 3.2.9 (Time-Convergence)**.** *Let $\mathcal{A} = (\mathcal{H}, Lab, Evt, Dur, Wgt)$ be a stochastic hybrid automaton. Using the function ExecTime defined for hybrid automata, the time duration of an infinite path $\pi = \hat{\sigma}_0 \xRightarrow{\alpha_0} \hat{\sigma}_1 \xRightarrow{\alpha_1} \hat{\sigma}_2 \xRightarrow{\alpha_2} \ldots$ of $\mathcal{A}$ is given by the overloaded function*

$$ExecTime(\pi) = \sum_{i=0}^{\infty} ExecTime(\alpha_i).$$

We call π time-divergent *if $ExecTime(\pi) = \infty$ and* time-convergent *otherwise.*

In the following, we discuss how the notions of Zenoness, timelock, and deadlock could be extended to stochastic hybrid automata. Instead of talking about the existence of paths, we now consider with which probability paths exists. Recall that we have only defined path probabilities for closed, invariant-free stochastic hybrid automata, which cannot contain deadlocked states. We will show that they may still contain Zeno paths and timelocks. However, since our probability calculations currently do not distinguish between time-convergent and time-divergent paths, in general we cannot calculate the probabilities of Zeno paths, timelocks, or deadlocks. We will discuss whether and how our probability calculations could be extended to measure the desired sets of paths. Additionally, we will take a brief look at stochastic hybrid automata that do contain invariants. Since we only defined the notions of Zenoness, timelock, and deadlock for closed hybrid automata, we will only consider closed stochastic hybrid automata here. Our considerations for non-closed hybrid automata can also be applied to stochastic hybrid automata.

Zeno Paths

Zeno paths can be defined analogously to hybrid automata. We do not need to take any path probabilities into account for deciding whether a single path is Zeno or not. Recall that for hybrid automata we only properly defined Zeno behavior for closed systems.

Definition 3.2.10 (Zeno Path)**.** *Let $\mathcal{A} = (\mathcal{H}, Lab, Evt, Dur, Wgt)$ be a closed stochastic hybrid automaton. An infinite path π of $\mathcal{A}$ is called a* Zeno path *if it is time-convergent and infinitely many discrete steps are taken within π.*

We are now interested in whether a stochastic hybrid automaton is *almost-surely non-Zeno*, i.e., whether the probability of taking a Zeno path is 0, instead of asking whether there exists a Zeno path from an initial state. A stochastic hybrid automaton $\mathcal{A}$ must be almost-surely non-Zeno if its underlying hybrid automaton $\mathcal{H}$ is non-Zeno, since each path of $\mathcal{A}$ corresponds to a path of $\mathcal{H}$. If $\mathcal{H}$ does contain Zeno paths, it depends on the probability distributions whether $\mathcal{A}$ is almost-surely non-Zeno. This is illustrated in the following examples.

Example 3.2.3. *The closed, invariant-free stochastic hybrid automaton $\mathcal{A}_1'$ defined in Example 3.2.2 must be almost-surely non-Zeno since the underlying hybrid automaton is non-Zeno. Similarly, the stochastic hybrid automaton $\mathcal{A}_1$, which includes location invariants other than true, is also almost-surely non-Zeno.*

Example 3.2.4. *Consider the stochastic hybrid automaton $\mathcal{A}_z$ presented in Figure 3.6. The underlying hybrid automaton extends the hybrid automaton from Example 2.2.2 by a controlled variable that counts how often a jump has been taken. We*

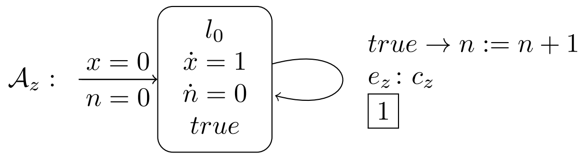


Figure 3.6: Stochastic hybrid automaton $\mathcal{A}_z$ containing a Zeno path and a timelock, as defined in Example 3.2.4.

choose $Lab_z = \{z\}$, $Evt_z(e_z) = z$ and $Wgt_z(e_z) = 1$. Let $t_i = \frac{1}{2^i}$ for $i \in \mathbb{N}$. We define Dur_z as follows:

$$Dur_z(z, (l_0, (x \mapsto t, n \mapsto i))) = U_{[0,t_i]} \text{ for } t \in \mathbb{R}_{\geq 0}, i \in \mathbb{N}.$$

The path with the longest possible execution time in $\mathcal{A}_z$ is

$$\begin{aligned}\pi = (l_0, (x \mapsto 0, n \mapsto 0), (c_z \mapsto t_0)) &\xRightarrow{t_0, f_0} (l_0, (x \mapsto t_0, n \mapsto 0), (c_z \mapsto 0)) \xRightarrow{e_z} \\ (l_0, (x \mapsto t_0, n \mapsto 1), (c_z \mapsto t_1)) &\xRightarrow{t_1, f_1} (l_0, (x \mapsto t_0 + t_1, n \mapsto 1), (c_z \mapsto 0)) \xRightarrow{e_z} \\ (l_0, (x \mapsto t_0 + t_1, n \mapsto 2), (c_z \mapsto t_2)) &\xRightarrow{t_2, f_2} (l_0, (x \mapsto \sum_{k=0}^{2} t_k, n \mapsto 2), (c_z \mapsto 0)) \xRightarrow{e_z} \\ \ldots\end{aligned}$$

where $f_i(t) = (x \mapsto t + \sum_{k=0}^{i-1} t_i, n \mapsto i)$ for $i \in \mathbb{N}$, $t \in \mathbb{R}_{\geq 0}$. π contains infinitely many jumps and $ExecTime(\pi) = 2$, so π is a Zeno path. All other possible infinite paths must also take e_z infinitely often, but be of shorter time duration. Hence, all possible infinite paths of $\mathcal{A}_z$ are Zeno paths and $\mathcal{A}_z$ cannot be almost-surely non-Zeno.

It is easy to see that a different choice of probability distributions would not lead to Zeno behavior, e.g., if the time between two jumps would always have to be 1. For more complex probability distributions it would however be more difficult to determine whether Zeno behavior occurs and with which probability.

In order to determine whether a stochastic hybrid automaton is almost-surely non-Zeno, we need to calculate the probability that any reachable path is Zeno. However, our probability calculations currently do not allow to measure Zeno paths. Bertrand et al. [BBB$^+$14] define stochastic timed automata in a similar way to our stochastic hybrid automata and show that the set of all Zeno paths from a state of a stochastic timed automaton is measurable, using *constrained* symbolic paths. We expect that their approach can be adapted to our situation. Bertrand et al. also showed that certain classes of stochastic timed automata are almost-surely non-Zeno. It would be interesting to investigate whether it is possible to identify classes of almost-surely non-Zeno stochastic hybrid automata in a similar way.

Timelock and Deadlock

We would consider a state $\hat{\sigma} \in \Sigma \times \mathcal{V}_{Lab}$ of a stochastic hybrid automaton to have a timelock or deadlock if the probability of taking a time-divergent path or any path, respectively, is 0. Note that we can compute probabilities of symbolic paths starting at a state $\hat{\sigma} \in \Sigma \times \mathcal{V}_{Lab}$, instead of a state $\sigma \in \Sigma$, via P' defined in Definition 3.2.8.

However, we currently do not see how these probability calculations could be used to calculate the desired probabilities of timelocks and deadlocks. In the following, we discuss this in more detail for both timelocks and deadlocks separately, after having looked at examples for stochastic hybrid automata which we would consider to contain timelocked or deadlocked states, respectively. Recall that for hybrid automata we only defined the notions of timelock and deadlock for closed systems.

Timelock Path probabilities were only defined for (closed and) invariant-free stochastic hybrid automata, because this subclass is deadlock-free (see Section 3.2). However, invariant-free stochastic hybrid automata may still contain states which we consider to be timelocked, as illustrated by the following example.

Example 3.2.5. *Recall the stochastic hybrid automaton $\mathcal{A}_z$ defined in Figure 3.6, where all infinite paths are Zeno. This directly implies that the initial state must be timelocked. Note that the underlying hybrid automaton is timelock-free, so this timelock is only caused by a "bad" choice of probability distributions.*

$\mathcal{A}_z$ does not contain a deadlocked state since the probability of taking any path is 1 at each reachable state.

In our example, it is relatively easy to see that a timelock must occur since all possible paths of the stochastic hybrid automaton are time-convergent. In general, however, this might be less straightforward if there also exist time-divergent paths and we need to calculate whether they are taken with a positive probability.

To check whether a state has a timelock, we need to be able to distinguish between the probabilities of time-convergent and time-divergent paths. One possible approach for expressing time-convergent paths might be the constrained symbolic paths proposed by Bertrand et al. [BBB$^+$14], which they use to define the set of all Zeno paths. However, symbolic paths only take finitely many execution steps into account, which is not sufficient to detect whether a path with finitely many discrete steps but infinitely many time steps is time-convergent.

Deadlock Invariant-free (stochastic) hybrid automata must be deadlock-free (see Section 3.2). Stochastic hybrid automata that do allow arbitrary location invariants however may also contain deadlocked states, even if the underlying hybrid automaton is deadlock-free. The following example illustrates that probability distributions must be chosen carefully in order to avoid introducing deadlocks.

Example 3.2.6. *Recall the closed stochastic hybrid automaton $\mathcal{A}_1$ from Example 3.1.2. If the support of the distribution $Dur(b, (l_0, \nu))$ was $\mathbb{R}$ instead of $[0{,}2]$, we would consider $\mathcal{A}_1$ to contain a deadlocked state (and hence also a timelocked state): If we sampled enabling durations $t_a > 5$ for e_a and $t_b > 2$ for e_b, neither jump would become fireable before the invariant of l_0 is violated. We would consider all initial states $(l_0, (x \mapsto 0), (c_a \mapsto t_a, c_b \mapsto t_b))$ with $t_a > 5$ and $t_b > 2$ to be timelocked and all states $(l_0, (x \mapsto 5), (c_a \mapsto t_a, c_b \mapsto t_b))$ with $t_a, t_b > 0$ to be deadlocked.*

By defining the distribution $Dur(b, (l_0, \nu))$ such that its support is contained in $[0{,}2]$, we can ensure that always some jump will become fireable before the invariant of l_0 is violated.

To check whether a state has a deadlock, we need to calculate the probability of taking any path. This can be expressed as the sum of the probabilities of taking any outgoing jump plus the probability of taking a path that does not leave the location. The first part can surely be calculated as the probability of symbolic paths, but the second part is more difficult. In Section 3.2.1, we discussed how to calculate the probability of staying in a location as a possible refinement of our probability calculation. However, this only works under the assumption that the stochastic hybrid automaton does not contain a deadlocked state, i.e., the probability of taking some path is 1 at all reachable states. We currently do not see how this could alternatively be calculated.

3.3 Composition of Stochastic Hybrid Automata

We consider two SHA *composable* if their underlying hybrid automata are composable and they have disjoint sets of labels specifying the stochastic events. Otherwise, a location in the composition of two SHA could have two outgoing jumps associated with the same event, which is forbidden. Note that merging these jumps is also not an option, since it is unclear how to merge arbitrary probability distributions in a meaningful way.

Definition 3.3.1 (Composability of SHA)**.** *Two stochastic hybrid automata* $\mathcal{A}_1 = (\mathcal{H}_1, Lab_1, Evt_1, Dur_1, Wgt_1)$ *and* $\mathcal{A}_2 = (\mathcal{H}_2, Lab_2, Evt_2, Dur_2, Wgt_2)$ *are considered* composable *if* $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *are composable and* $Lab_1 \cap Lab_2 = \emptyset$.

Analogous to the composition of hybrid automata (Section 2.3), we first define the extension of a SHA by a set of variables before constructing the syntactic parallel composition of two SHA. We extend a SHA by first extending the underlying hybrid automaton and then adapting the assignment of events, probability distributions, and weights such that they refer to the extended jumps and states.

Definition 3.3.2 (Extension of SHA)**.** *Let* $\mathcal{A}$ *be a SHA and* V' *a set of real-valued variables* $Con \cap V' = \emptyset$. *The* extension *of* $\mathcal{A}$ *by* V' *is the hybrid automaton* $\mathcal{A}^+ = (\mathcal{H}^+, Lab^+, Evt^+, Dur^+, Wgt^+)$, *where*

- $\mathcal{H}^+$ *is the extension of the hybrid automaton* $\mathcal{H}$ *by* V';
- $Lab^+ = Lab$;
- $Evt^+(e^+) = Evt(e)$ *for the unique* $e \in Edge$ *corresponding to* $e^+ \in Edge^+$;
- $Dur^+(r,(l,\nu^+)) = Dur(r,(l,\nu^+|_V))$ *for all* $r \in Lab^+$, $l \in Loc^+$, $\nu^+ \in \mathcal{V}^+$;
- $Wgt^+(e^+) = Wgt(e)$ *for the unique* $e \in Edge$ *corresponding to* $e^+ \in Edge^+$.

We can now define the composition of two SHA based on the composition of their underlying hybrid automata and the extensions of the SHA by the variables of the respective other SHA. Each jump in the composition corresponds to an jump in one of the components and can consequently be assigned the same stochastic event and weight as the corresponding jump in its respective component. Since the SHA may not share any event labels, each pair of event label and state can be assigned the same distribution as the corresponding label-state pair in the component indicated by the label.

Definition 3.3.3 (Syntactic Parallel Composition of SHA)**.** *Let $\mathcal{A}_1, \mathcal{A}_2$ be two SHA that are composable. Let*

$$\mathcal{A}_1^+ = (\mathcal{H}_1^+, \mathit{Lab}_1^+, \mathit{Evt}_1^+, \mathit{Dur}_1^+, \mathit{Wgt}_1^+),$$
$$\mathcal{A}_2^+ = (\mathcal{H}_2^+, \mathit{Lab}_2^+, \mathit{Evt}_2^+, \mathit{Dur}_2^+, \mathit{Wgt}_2^+)$$

be the extensions of $\mathcal{A}_1, \mathcal{A}_2$ by $V_2 \backslash \mathit{Con}_1$ and $V_1 \backslash \mathit{Con}_2$, respectively. The parallel composition $\mathcal{A}_1 \parallel \mathcal{A}_2 = (\mathcal{H}, \mathit{Lab}, \mathit{Evt}, \mathit{Dur}, \mathit{Wgt})$ *is the SHA with*

- $\mathcal{H} = \mathcal{H}_1 \parallel \mathcal{H}_2$*;*
- $\mathit{Lab} = \mathit{Lab}_1 \cup \mathit{Lab}_2$*;*
- $\mathit{Evt}(\mathit{id}, (l_1,l_2), \mu, (l_1', l_2')) = \begin{cases} \mathit{Evt}_1^+(e_1) & \text{if } e_1 = (\mathit{id}, l_1, \mu, l_1') \in \mathit{Edge}_1^+ \wedge l_2 = l_2' \\ \mathit{Evt}_2^+(e_2) & \text{if } e_2 = (\mathit{id}, l_2, \mu, l_2') \in \mathit{Edge}_2^+ \wedge l_1 = l_1'; \end{cases}$
- $\mathit{Dur}(r,((l_1,l_2), \nu)) = \begin{cases} \mathit{Dur}_1^+(r, (l_1, \nu)) & \text{if } r \in \mathit{Lab}_1 \\ \mathit{Dur}_2^+(r, (l_2, \nu)) & \text{if } r \in \mathit{Lab}_2; \end{cases}$
- $\mathit{Wgt}(\mathit{id}, (l_1,l_2), \mu, (l_1', l_2')) = \begin{cases} \mathit{Wgt}_1^+(e_1) & \text{if } e_1 = (\mathit{id}, l_1, \mu, l_1') \in \mathit{Edge}_1^+ \wedge l_2 = l_2' \\ \mathit{Wgt}_2^+(e_2) & \text{if } e_2 = (\mathit{id}, l_2, \mu, l_2') \in \mathit{Edge}_2^+ \wedge l_1 = l_1'. \end{cases}$

If we did not require the jumps in the two components to have (different) jump identifiers, there might exist self-loops $e_1 \in \mathit{Edge}_1^+$ and $e_2 \in \mathit{Edge}_2^+$ from locations $l_1 \in \mathit{Loc}_1$ and $l_2 \in \mathit{Loc}_2$, respectively, that have the same guard and the same reset, but are associated with different stochastic events. Then, we could not distinguish between e_1 and e_2 in location (l_1,l_2) of the composition of the two stochastic hybrid automata and the jump set of the composition would only contain one of these identical jumps. For hybrid automata this is not a problem, but for stochastic hybrid automata we need two distinguishable jumps associated with the two different stochastic events $\mathit{Evt}(e_1)$ and $\mathit{Evt}(e_2)$. Hence, we only consider (stochastic) hybrid automata to be composable if they do not use the same jump identifiers.

The following example illustrates the syntactic parallel composition of SHA using our running examples. We omit the jump identifiers again since all jumps can already be distinguished by their guards.

Example 3.3.1. *Consider again the SHA $\mathcal{A}_1$ and $\mathcal{A}_2$ from Examples 3.1.2 and 3.1.3, as well as the composition $\mathcal{H}_1 \parallel \mathcal{H}_2$ of their underlying hybrid automata defined in Example 2.3.4. The composition $\mathcal{A}_1 \parallel \mathcal{A}_2 = (\mathcal{H}, \mathit{Lab}, \mathit{Evt}, \mathit{Dur}, \mathit{Wgt})$ has the hybrid automaton $\mathcal{H}_1 \parallel \mathcal{H}_2$ and*

- $\mathit{Lab} = \{a, b, c\}$*;*
- $\mathit{Evt}(e_r^i) = r$ *for* $r \in \mathit{Lab}$ *and* $i \in \{1,2\}$*;*
- $\mathit{Dur}(a,((l_0,l_2),\nu))(t) = \mathit{Dur}(a,((l_0,l_3),\nu))(t) = 2e^{-2t}$*,*
  $\mathit{Dur}(b,((l_0,l_2),\nu)), \mathit{Dur}(b,((l_0,l_3),\nu)) = U_{[0,2]}$*,*
  $\mathit{Dur}(c,((l_0,l_2),\nu))(t) = \mathit{Dur}(c,((l_1,l_2),\nu))(t) = 2e^{-2t}$*;*
- $\mathit{Wgt}(e_b^1) = \mathit{Wgt}(e_b^2) = 3$ *and* $\mathit{Wgt}(e) = 1$ *for all other edges* e*.*

Revisiting our example from the introduction, we see that it is now indeed possible to compose the stochastic hybrid automata as intended.

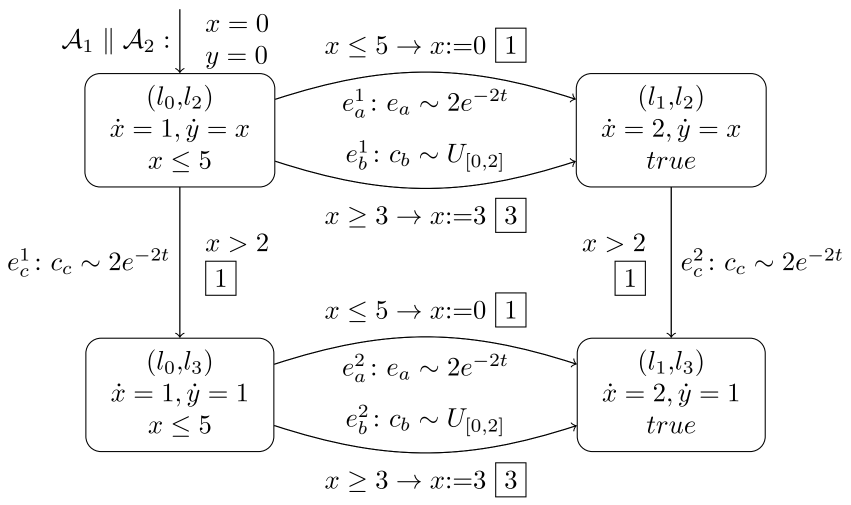

$\mathcal{A}_1 \parallel \mathcal{A}_2$:
$x = 0$
$y = 0$
(l_0,l_2)
$\dot{x} = 1, \dot{y} = x$
$x \leq 5$
(l_1,l_2)
$\dot{x} = 2, \dot{y} = x$
true
(l_0,l_3)
$\dot{x} = 1, \dot{y} = 1$
$x \leq 5$
(l_1,l_3)
$\dot{x} = 2, \dot{y} = 1$
true
$x \leq 5 \to x{:=}0$ $\boxed{1}$
$e_a^1 \colon e_a \sim 2e^{-2t}$
$e_b^1 \colon c_b \sim U_{[0,2]}$
$x \geq 3 \to x{:=}3$ $\boxed{3}$
$e_c^1 \colon c_c \sim 2e^{-2t}$
$x > 2$ $\boxed{1}$
$x > 2$ $\boxed{1}$
$e_c^2 \colon c_c \sim 2e^{-2t}$
$x \leq 5 \to x{:=}0$ $\boxed{1}$
$e_a^2 \colon e_a \sim 2e^{-2t}$
$e_b^2 \colon c_b \sim U_{[0,2]}$
$x \geq 3 \to x{:=}3$ $\boxed{3}$

Figure 3.7: Composition of the stochastic hybrid automata $\mathcal{A}_1$ and $\mathcal{A}_2$, as defined in Example 3.3.1.

3.4 Comparison to Other Approaches

We now compare our model to two stochastic hybrid automaton formalisms that are similar to our approach with respect to different aspects. Bernadsky et al. [BSA04] also offer communication via shared variables for continuous-time stochastic hybrid automata, while the modeling language proposed by Pilch et al. [PKRA20] is closest to our approach with regard to the modeling of stochastic events. Both were already discussed in Chapter 1.

[BSA04]: Shared Variables Bernadsky et al. propose a modeling language for continuous-time concurrent stochastic hybrid systems, where components can read variables that other components have marked as observable. This allows a more fine-grained read/write access than our model, where all controlled variables can be read by other components. However, adding a distinction between private and observable controlled variables in our model should be straightforward.

The authors do not want to restrict the influence of external/non-controlled variables. As discussed in Section 3.1, this might allow the environment to violate location invariants instantaneously. This is solved by allowing forced jumps, which can be taken as soon as the invariant is violated. We prefer not to weaken the invariant semantics and instead do not allow non-controlled variables to influence location invariants or initial conditions, or to reset non-controlled variables.

Our model offers spontaneous jumps, i.e., to determine the jump times via probability distributions. In contrast, Bernadsky et al. only allow forced jumps. Their model however does offer SDE-based flow specifications and stochastic resets, which our model currently does not include. We discuss how forced jumps, SDE, and stochastic resets could be included in our model in Section 4.2.

In conclusion, Bernadsky et al. chose a different approach for handling the influence of non-controlled variables and their offered stochasticity is orthogonal to our approach.

[PKRA20]: Modeling Stochastic Events The model defined by Pilch et al. is the only one we are aware of which also allows to bind together jumps such that their jump time is determined by the same stochastic event. Pilch et al. extend singular automata by random clocks syntactically, while we add random clocks only semantically to protect their intended usage. In their model, jumps can reset the random clocks to random values according to specified continuous probability distributions. In each location, each clock may either run down linearly or stay constant.

By guarding several jumps with the same condition on the same random clock, these jumps can effectively be bound together in a similar way to our approach. However, modeling spontaneous guarded jumps as in our model would take more effort. We use the random clocks to measure for each stochastic event how long any associated jump has been enabled in total, which means that it depends on the enabledness of jumps whether a clock is currently running down or staying constant. In [PKRA20], however, it depends on the location whether a clock runs down or not. Hence, one has to introduce auxiliary transitions and locations in order to model the behavior of our random clocks via these syntactic random clocks.

We argue that our approach is better suited for modeling competing stochastic events. Additionally, Pilch et al. only extend a very restricted class of hybrid automata by stochastic behavior.

In conclusion, our model fares well in comparison to the two considered approaches regarding shared variable communication and modeling of stochastic events, respectively. Additionally, our modeling language has the advantage of combining both considered aspects.

Chapter 4

Language Extensions

In this section, we discuss possible extensions of our modeling language. We first consider how other sources of stochasticity regarding the discrete and continuous dynamics could be included. Secondly, we look further into forced jumps, which have been mentioned in Section 3.2. Finally, we elaborate on how the communication between components could be extended to jump synchronization or concurrent writing of variables.

4.1 Other Sources of Stochasticity

As discussed in the introduction, hybrid systems can be extended by stochastic behavior in many different ways. Our model currently includes stochastic choices for the jump times and the decision between jumps. Other possibilities are stochastic resets, stochastic choice of initial states, and stochastic continuous dynamics (see, e.g., [HLS00, BL04]).

Our modeling language is based on the stochastic hybrid automata proposed in [Sch22], which offer probability distributions over variable resets as well as initial locations and valuations. This model also distinguishes between controlled and non-controlled variables, but imposes different conditions on the activities and on the influence of the non-controlled variables. We expect that this approach for stochastic resets and initial states can be easily adapted for our model.

Allowing for stochastic continuous dynamics, on the other hand, might be more challenging. The combination of stochastic differential equations with guarded spontaneous jumps and invariants yields very complex dynamics. Many models that include SDE and location invariants only allow forced jumps [HLS00, BSA04]. Bujorianu and Lygeros [BL04] proposed a model which additionally also includes spontaneous jumps, but they are not guarded.

4.2 Forced Jumps

Our model currently offers spontaneous jumps, whose jump times are determined by probability distributions. Other modeling languages also (or only) allow to leave a location if the boundary of the location invariant is hit [Dav84, BSA04, BLB06]. In Section 3.2, we discussed that introducing such forced jumps might enable us to avoid deadlocks. We see two possibilities of how our model can be extended to include forced jumps.

Our model requires to leave a location before its invariant is violated. Bernadsky et al. [BSA04] instead choose a weaker notion of invariants, where time can only evolve as long as the invariant is satisfied, but as soon as the invariant is violated a jump has to be taken. If we weakened our invariants analogously, we could model forced jumps as urgent jumps guarded with the negation of the location invariant. Note that weakening the invariants would also mean that non-controlled variables causing an instantaneous violation of the invariant would not be a problem anymore.

Instead of weakening the invariant semantics, we could introduce another type of jumps. More specifically, we could add non-stochastic jumps that can be taken from a state (l,ν) if there exists some activity $f \in \mathit{Act}(l)$ with $f(0) = \nu$ such that the invariant would become violated in any next time step using f. Hence, the jump can be taken if there exists an activity f such that no time step using f is possible. Our probability calculations would need to be adjusted to reflect that a forced jump is taken with probability 1 if time cannot evolve without violating the invariant.

However, for an open invariant like '$x < 1$', there always exists some time step such that $x < 1$ still holds, and thus we could never take the forced jump. Hence, this approach only works with closed invariants. In non-closed systems, we model the influence of the environment non-deterministically. This means that there might exist different evolutions of the non-controlled variables such that the location invariant is violated in the next time step for one evolution, but not the other. For example, consider a state $(l, (x \mapsto 1, y \mapsto 0))$ where the location invariant of l is '$x \leq 1$' and the evolution of the controlled variable x depends on the non-controlled variable y via $\dot{x} = y$. We allow all possible evolution of y. If we assume that y increases linearly, then the location invariant would be violated in any next time step and thus we can take the forced jump. On the other hand, if we assume that y stays constant or decreases linearly, then we do not need to leave the location.

4.3 Jump Synchronization

It is often convenient to allow several components to communicate via jump synchronization. To this end, the jumps must be additionally associated with some synchronization labels. Two components should synchronize on these labels in the sense that jumps from different components with the same synchronization label should always be executed together. However, it is not immediately clear when they should be executed.

Recall that we only consider stochastic hybrid automata composable if their sets of event labels are disjoint (see Section 3.3). Hence, jumps from different stochastic hybrid automata cannot be associated with the same event label. The probability that two jumps associated with different event labels become fireable at the same time is 0 if their jump times are determined by continuous probability distributions.

Therefore, most approaches for compositional modeling of stochastic hybrid systems only allow to synchronize non-spontaneous jumps [CL07, Chapter 3] or spontaneous with non-spontaneous jumps [Buj05, DDL^{+}12]. We are not aware of any model allowing to synchronize spontaneous jumps with other spontaneous jumps. There exist different approaches for the synchronization of spontaneous with non-spontaneous jumps, which are suited for different scenarios.

Firstly, we could require that whenever we want to take a stochastic jump, matching non-stochastic jumps have to be enabled in all other components. This condition could also be restricted such that only a certain set of stochastic jumps is supposed to be synchronized at all. This approach was chosen by David et al. in [DDL$^+$12], where the time when a component takes an “output” action is determined stochastically and all other components have to “passively” synchronize by performing a corresponding input action. David et al. assure that the latter is always possible by requiring all components to be input-enabled, i.e., all input actions have to be possible in all states.

Alternatively, we could allow a stochastic jump to be taken on its own unless a non-stochastic jump with the same synchronization label is currently enabled in another component, as proposed by Bujorianu [Buj05]. Bujorianu distinguishes between active and passive jumps. Active jumps can be spontaneous or forced, while passive jumps may only be executed if and only if an active jumps with the same synchronization label is executed by another component at the same time. If a component wants to take an active transition but no corresponding passive transition exists in the other component, the former component may nevertheless execute its transition. Hence, it is not assumed that a synchronizing jump exists at all times.

4.4 Shared Variable Concurrency

Our model currently only allows each variable to be written by at most one component. In some cases it might however be desirable to have truly shared variables, which can be written by several components concurrently. Here we only consider how several components could be allowed to write variables via resets and assume that the evolution of all variables is determined by a single component or that the components at least prescribe the same continuous evolution. We see at least three possibilities to include concurrent writing in our model, each facing different challenges.

Firstly, we could allow concurrent writing directly, i.e., simply allowing to reset non-controlled variables as, for example, done by Bernadsky [BSA04]. Then, the invariant of the current location of one component might become violated if another component changes the value of a shared variable. Bernadsky et al. solved this by weakening the invariant semantics such that a forced jump is taken as soon as the location invariant is violated.

Instead of weakening the invariants, we could introduce jump synchronization, such that a jump changing a shared variable must always be taken together with matching jumps in the other components. More specifically, we would need to add non-spontaneous “passive” jumps to all locations, defining where to jump for all possible changes of the shared variables, and add synchronization labels matching the jumps. Similar to the assumption of input-enabledness in [DDL$^+$12], we would need to require that some passive transition exists for every possible change of the shared variables at all times. Jumps violating the current location invariant could either lead to a sink location, or, in the spirit of forced jumps with weakened invariants, allow to transition to a different location to continue the execution.

Another possibility is to implement some handshaking mechanism via additional variables and locations, which could work roughly as follows. We could introduce a dedicated component $\mathcal{A}_w$ which controls the variables that are supposed to be shared. Each component needs to be extended by a set of controlled variables that are used to communicate with $\mathcal{A}_w$. If a component wants to reset a shared variable via some jump, it signals the desired changes to $\mathcal{A}_w$ and transitions to a temporary location while the variable values are changed. $\mathcal{A}_w$ then has to instantaneously change the variable values and signal the successful reset back, which allows the original component to transition to the originally desired target location. During this procedure, no time may pass and hence all jumps have to be urgent. All other components have to synchronize on the changes, possibly via additional flags or jump synchronization. Since implementing a handshaking mechanism is rather involved and complicated, and requires the introduction of many new variables, transitions, and locations, this approach seems less advantageous from a practical point of view.

Chapter 5

Conclusion

In this thesis, we proposed a modeling language for stochastic hybrid systems which extends hybrid automata by stochastic behavior regarding the jump times and choice between jumps. By binding together several jumps to model a stochastic event, we enable modeling competing stochastically independent events.

We saw that the probability distributions determining the jump times must be chosen carefully in order to avoid introducing timelocks or deadlocks. For closed, invariant-free stochastic hybrid automata, we showed how to calculate the probabilities of symbolic paths. A challenge here is that the probability distributions do not always directly give the probabilities of sampling certain enabling durations, since they may also assign positive probabilities to enabling durations that are never reached. These probabilities are redistributed, which introduces implicit stochastic dependencies. We also discussed how our model could be extended by other sources of stochasticity or to allow for forced jumps.

Our model is compositional in the sense that components may communicate via shared variables. Each component models the behavior of the other components non-deterministically. We argued that the influence of non-controlled variables on invariants, initial valuation sets, and resets must be restricted in order to avoid conflict between components. We discussed several possibilities for including shared variable concurrency, i.e., writing non-controlled variables in resets, or jump synchronization.

5.1 Future Work

Future work could extend our modeling language by other features as described in Chapter 4, such as other sources of stochasticity, forced jumps, jump synchronization, or shared variable concurrency.

In Section 3.2 we discussed several possibilities for future research regarding the path probability calculations. Firstly, it would be desirable to be able to define path probabilities for a wider class than just closed, invariant-free stochastic hybrid automata. One possibility might be to avoid deadlocks by introducing forced jumps. Further, one could investigate how to distinguish between the probabilities of time-divergent and time-convergent paths and, based on that, determine the probability of timelocks and Zeno paths. Additionally, it might be worthwhile to identify subclasses of stochastic hybrid automata that are almost-surely non-Zeno, similar to [BBB$^+$14].

Since our modeling language is compositional, it would be beneficial to be able to formulate some compositional semantics which allows us to describe the behavior of the composition of two stochastic hybrid automata without having to explicitly build their syntactic parallel composition. Defining possible execution steps should be rather straightforward. However, we currently do not see a possibility for calculating path probabilities for a composed stochastic hybrid automaton based on the probability calculations of the two components, especially since these calculations are only defined for closed systems.

Finally, it might also be interesting to investigate whether it would be possible to define a fully abstract semantics which allows synchronization of components only via their observable behavior, instead of using all information captured by the paths.

Bibliography

[ACH$^+$95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[AHS13] Duarte Antunes, Joao P. Hespanha, and Carlos Silvestre. Stochastic hybrid systems with renewal transitions: Moment analysis with application to networked control systems with delays. *SIAM Journal on Control and Optimization*, 51(2):1481–1499, 2013.

[AL94] Martin Abadi and Leslie Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543–1571, 1994.

[App21] Matthias Appenzeller. *Comparing Two Modeling Formalisms for Stochastic Hybrid Systems.* Bachelor's thesis, RWTH Aachen University, 2021.

[BBB$^+$14] Nathalie Bertrand, Patricia Bouyer, Thomas Brihaye, Quentin Menet, Christel Baier, Marcus Größer, and Marcin Jurdzinski. Stochastic timed automata. *Logical Methods in Computer Science*, 10(4), 2014.

[BBCM16] Patricia Bouyer, Thomas Brihaye, Pierre Carlier, and Quentin Menet. Compositional design of stochastic timed automata. In Alexander S. Kulikov and Gerhard J. Woeginger, editors, *Computer Science – Theory and Applications*, pages 117–130. Springer, 2016.

[BBR$^+$20] Patricia Bouyer, Thomas Brihaye, Mickael Randour, Cédric Rivière, and Pierre Vandenhove. Decisiveness of stochastic systems and its application to hybrid models. volume 326, pages 149–165. Open Publishing Association, 2020.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking.* MIT press, 2008.

[BL04] Manuela L. Bujorianu and John Lygeros. General stochastic hybrid systems: Modelling and optimal control. In *Proceedings of the 43rd IEEE Conference on Decision Control*, volume 2, pages 1872–1877, 2004.

[BLB06] Luminita Manuela Bujorianu, John Lygeros, and Marius C. Bujorianu. Toward a general theory of stochastic hybrid systems. *Lecture Notes in Control and Information Sciences*, 337:3–30, 2006.

[Bow01] Howard Bowman. Time and action lock freedom properties for timed automata. In Myungchul Kim, Byoungmoon Chin, Sungwon Kang, and Danhyung Lee, editors, *Formal Techniques for Networked and Distributed Systems*, pages 119–134, Boston, MA, 2001. Springer.

[BSA04] Mikhail Bernadsky, Raman Sharykin, and Rajeev Alur. Structured modeling of concurrent stochastic hybrid systems. In Yassine Lakhnech and Sergio Yovine, editors, *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 309–324. Springer, 2004.

[Buj05] Manuela-Luminita Bujorianu. *Stochastic Hybrid System: Modelling and Verification.* Dissertation, University of Stirling, 2005.

[CL07] Christos G. Cassandras and John Lygeros, editors. *Stochastic Hybrid Systems.* Control Engineering Series. CRC, 2007.

[Dav84] Mark H.A. Davis. Piecewise-deterministic Markov processes: A general class of non-diffusion stochastic models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 46(3):353–376, 1984.

[DDL$^+$12] Alexandre David, Dehui Du, Kim G. Larsen, Axel Legay, Marius Mikuč ionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical model checking for stochastic hybrid systems. *Electronic Proceedings in Theoretical Computer Science*, 92:122–136, 2012.

[Fel91] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. John Wiley and Sons, 1991.

[FHH$^+$11] Martin Fränzle, Ernst Moritz Hahn, Holger Hermanns, Nicolás Wolovick, and Lijun Zhang. Measurability and safety verification for stochastic hybrid systems. In *Proceedings of the 14th ACM International Conference on Hybrid Systems: Computation and Control*, pages 43–52. ACM, 2011.

[GAM97] Mrinal K. Ghosh, Aristotle Arapostathis, and Steven I. Marcus. Ergodic control of switching diffusions. *SIAM Journal on Control and Optimization*, 35(6):1952–1988, 1997.

[Gbu18] Daniel Gburek. *Stochastic Transition Tystems: Bisimulation, Logic, and Composition.* Dissertation, TU Dresden, 2018.

[Hes05] Joao Hespanha. A model for stochastic hybrid systems with application to communication networks. *Nonlinear Analysis: Theory, Methods & Applications*, 62:1353–1383, 09 2005.

[Hes07] Joao Hespanha. Modeling and analysis of stochastic hybrid systems. *IEE Proceedings — Control Theory and Applications,* Special Issue on Hybrid Systems, 153:520–535, 01 2007.

[Hes14] Joao Hespanha. Modeling and analysis of networked control systems using stochastic hybrid systems. *Annual Reviews in Control*, 38, 10 2014.

[HHHK13] Ernst Moritz Hahn, Arnd Hartmanns, Holger Hermanns, and Joost-Pieter Katoen. A compositional modelling and analysis framework for stochastic hybrid systems. *Formal Methods in System Design*, 43(2):191–232, 2013.

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.

[HLS00] Jianghai Hu, John Lygeros, and Shankar Sastry. Towards a theory of stochastic hybrid systems. In Nancy Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, pages 160–173. Springer, 2000.

[Hua21] Mengzhe Hua. *Approximate Model Checking for Probabilistic Rectangular Automata with Continuous-Time Probability Distributions on Jumps.* Bachelor's thesis, RWTH Aachen University, 2021.

[JELS99] Karl Henrik Johansson, Magnus Egerstedt, John Lygeros, and Shankar Sastry. On the regularization of zeno hybrid automata. *Systems and Control Letters*, 38(3):141–150, 1999.

[LLA$^+$21] Luca Laurenti, Morteza Lahijanian, Alessandro Abate, Luca Cardelli, and Marta Kwiatkowska. Formal and efficient synthesis for continuous-time linear stochastic hybrid processes. *IEEE Transactions on Automatic Control*, 66(1):17–32, 2021.

[LSAZ21] Abolfazl Lavaei, Sadegh Soudjani, Alessandro Abate, and Majid Zamani. Automated verification and synthesis of stochastic hybrid systems: A survey. arXiv:2101.07491, 2021.

[LTS99] John Lygeros, Claire Tomlin, and Shankar Sastry. Controllers for reachability specifications for hybrid systems. *Automatica*, 35(3):349–370, 1999.

[Mar04] Nelson G. Markley. *Principles of Differential Equations.* Pure and Applied Mathematics: A Wiley Series of Texts, Monographs and Tracts. Wiley, 2004.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.

[NHR21] Mathis Niehage, Arnd Hartmanns, and Anne Remke. Learning optimal decisions for stochastic hybrid systems. In *Proceedings of the 19th ACM-IEEE International Conference on Formal Methods and Models for System Design*, page 44–55. ACM, 2021.

[PBLD03] Giordano Pola, Manuela-Luminita Bujorianu, John Lygeros, and Maria D. Di Benedetto. Stochastic hybrid models: An overview. *Proceedings IFAC Conference on Analysis and Design of Hybrid Systems*, 36(6):45–50, 2003.

[PKRA20] Carina Pilch, Maurice Krause, Anne Remke, and Erika Abraham. A transformation of hybrid Petri nets with stochastic firings into a subclass of stochastic hybrid automata. In *Proceedings NASA Formal Methods Symposium*, pages 381–400. Springer, 2020.

[Sch22] Nadja Scherer. *A Compositional Modeling Language for Stochastic Hybrid Systems.* Bachelor's thesis, RWTH Aachen University, 2022.

[TEF11] Tino Teige, Andreas Eggers, and Martin Fränzle. Constraint-based analysis of concurrent probabilistic hybrid systems: an application to networked automation systems. *Nonlinear Analysis: Hybrid Systems*, 5(2):343–366, 2011.

[Tri99] Stavros Tripakis. Verifying progress in timed systems. In Joost-Pieter Katoen, editor, *Formal Methods for Real-Time and Probabilistic Systems*, pages 299–314. Springer, 1999.

[TSS14] Andrew R. Teel, Anantharaman Subbaraman, and Antonino Sferlazza. Stability analysis for stochastic hybrid systems: A survey. *Automatica*, 50(10):2435–2456, 2014.

[Zil12] Dennis G Zill. *A First Course in Differential Equations with Modeling Applications*. Cengage Learning, 2012.

[ZJLS01] Jun Zhang, Karl Johansson, John Lygeros, and Shankar Sastry. Zeno hybrid systems. *International Journal of Robust and Nonlinear Control*, 11:435 – 451, 04 2001.