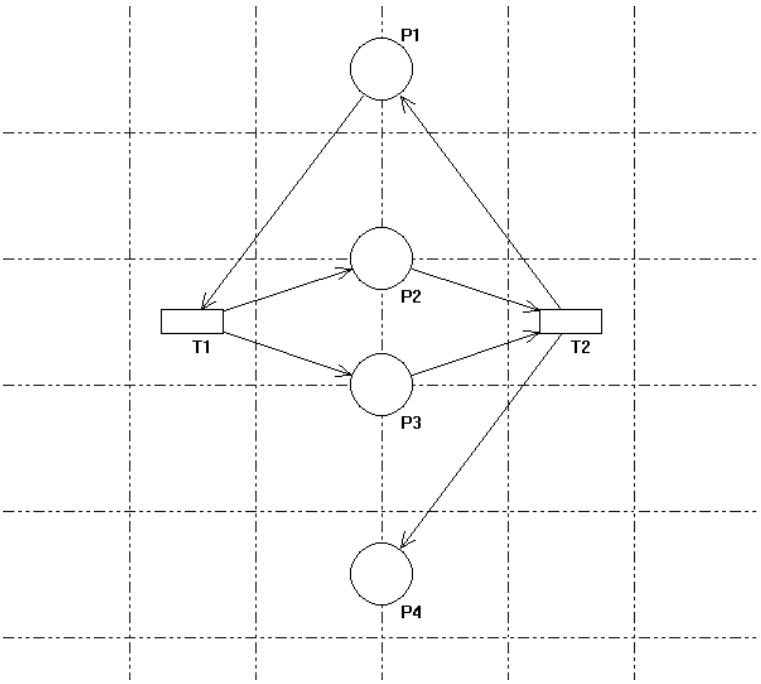
university of groningen
faculty of mathematics and natural sciences

Analysis Methods of Hybrid Systems Applied to Stochastically and Dynamically Colored Petri Nets

J.L. Dokter



Master's Thesis in Applied Mathematics

May 10, 2013

Analysis Methods of Hybrid Systems Applied to Stochastically and Dynamically Colored Petri Nets

Summary

Stochastically and dynamically colored Petri nets (SDCPN) are used to model complex and partially uncertain physical behavior, such as airplane transport. They are powerful due to their precise modeling, but there is a need for additional analysis methods to evaluate a model efficiently. In order to get access to these analytical tools SDCPN has been shown to be mathematically equivalent to hybrid system modeling formalisms, thus inheriting their properties and analysis. This paper investigates what analytical tools are available for a series of classes of hybrid system models, and how these apply to SDCPN. Based on this information we propose a procedure for analyzing SDCPN models.

Master's Thesis in Applied Mathematics
Author: J.L. Dokter
First supervisors: A.J. van der Schaft and M. H. C. Everdij
Second supervisor: E.C. Wit
Date: May 10, 2013

Institute of Mathematics and Computing Science
P.O. Box 407
9700 AK Groningen
The Netherlands

Contents

Chapter 1

Introduction

Imagine the following situation. For safety reasons, the International Civil Aviation Organization (ICAO) administers a rule that aircraft need to keep a minimum distance of five nautical miles apart. This restricts the number of flights per day over heavily used airspace, such as in Western Europe. ICAO would like to change this rule to three nautical miles to increase the amount of air transport. How could one assess the passenger safety of this proposal? Surely it is no option to 'try it out and see what happens'. One needs to obtain data and insights before the situation at hand has ever occurred. It is paramount to investigate these cases closely, as safety is the first priority in air transport. Institutions concerned with air transport safety face difficult dillemmas, like the one described above, on a regular basis. Mathematical modeling provides a powerful approach to this problem.

Mathematical models are used for a variety of purposes. They give insights about reality beyond what we can learn empirically. Models can be used to collect data which is otherwise too expensive, dangerous or simply impossible to gather from physical experiments. Moreover, models give graphical aid to understand causal relations, and provide insight in the system under scrutiny in general [Adl01].

It is generally impossible to find a model that is in some way equal to reality. A model is merely a tool which gives us greater opportunity to investigate relations than reality alone. For example, in a model we can perform a sensitivity analysis to find the influence a parameter has on the behavior of the model. Changing a parameter, which is easy in a model, is often impossible in reality.

This thesis will look at the modeling formalism called stochastically and dynamically colored Petri nets (SDCPN), which is a powerful formalism to model large scale physical situations. SDCPN formalism has been developed to uphold the strong Markov property, which allows for useful mathematical analysis. Researchers also use these models to perform Monte Carlo simulations, which simulate real-life events. They use equivalence relations between SDCPN and other modeling formalisms to make these simulations more efficient. To aid in this search towards similar models with more analysis tools this thesis will research what classes of hybrid models exist, what kinds of analysis tools are available for each class and to what extent this can aid in the development of analysis methods for SDCPN. This thesis will also try to determine what research paths are not applicable, and thereby give boundaries to the scope of research that is useful for SDCPN.

This thesis is written at the National Aerospace Laboratory (NLR: Nationaal Lucht- en

Ruimtevaartlaboratorium), a knowledge enterprise that specializes in the safety, efficiency and sustainability of air transport operations [nlr]. Originally it was founded as the Government Service for Aeronautical Studies in 1919 for the assistance of Dutch military safety. It has long since changed to a non-profit organization that performs research for both governmental agencies and private institutions. The Air Transport Safety Institute, a faculty within the NLR, is focused on research and consultancy concerning the safety of air transport.

This thesis is organized in the following manner. First the reader is introduced to hybrid systems in chapter 2. In this chapter we also discuss stochastic hybrid systems. We will also explore what kind of questions researchers might raise when dealing with hybrid models. Chapter 3 gives an introduction to Petri nets in general and SDCPN in particular. Chapter 4 gives a thorough description of the most extensively used and researched hybrid modeling formalisms besides SDCPN and describes their most important mathematical properties and analysis tools. Chapter 5 explores which of the analysis tools described in the previous chapter can be made applicable to SDCPN. As this is often only possible under strict conditions or reductions performed on SDCPN, this chapter will also discuss the limitations of such reductions in a qualitative manner. Based on chapter 5 we will conclude how these results could be applied in the analysis of SDCPN models.

Chapter 2

Introduction to Hybrid Systems

In this chapter we introduce hybrid systems. We will focus on both the deterministic and the stochastic version. Special attention will be paid to air transport operations on a large scale as a hybrid system. In this chapter we shall also introduce the example that is used throughout this thesis, a simplified version of the landing of an aircraft under nominal or non-nominal conditions [AEN12].

2.1 Hybrid Systems

Hybrid systems are dynamical systems whose evolution depends on a coupling between variables that take values in a continuum and variables that take values in a finite or countable set [SS99]. Many real world variables are continuous, such as the temperature outside, spatial coordinates or time. Other aspects however might be discrete, such as a machine being either on or off, an aircraft braking or not braking, or virtually any aspect of a computer. In many applications mathematicians choose a model to be either entirely discrete or entirely continuous, and model all aspects of the system that way. Computer scientists typically try to discretize the entire system, whereas mathematicians with a systems and control background try to fit everything into dynamical equations. This meant that researchers could model most of a system in their area of expertise, and deal with aspects that could not fit that framework on a case by case basis. However, in cases where the coupling between discrete and continuous elements is very important, i.e. the system to be modeled is itself of a hybrid nature, systematic ways of modeling these couplings might be necessary [SS99].

Generally, hybrid systems portray two kinds of evolution; a continuous dynamic evolution and 'jumps' caused by certain events. In most models the state, i.e. all relevant variables at that moment, vary according to given physical rules, until an event occurs. This event causes instantaneous change to the state in a discontinuous way, and may also change the continuous dynamics of the system. In between events the continuous variables follow ordinary differential equations, or ODEs.

2.2 Stochastic Evolution in Hybrid Systems

Inherent in many hybrid systems is a notion of randomness. There are many factors that are uncertain and which cannot be predicted in advance, in both the continuous and the discrete part of a hybrid system. Firstly we look at uncertainty in discrete aspects of a system.

Uncertainty can be introduced through the occurrence of events. If a machine has a certain chance of failure, for example the communicating devices or the altitude meter, we could model the time until failure through an exponential distribution with some parameter λ. In the model the apparatus fails once an exponentially distributed time has lapsed. Although we are dealing with a continuous variable, the effect on the system is during a single moment, or event. Also the initial values might be probabilistic. In a model of a single flight of an airplane, we perhaps want to vary the location of the destination, so that we have a model of a flight of a single airplane from a given starting point to any destination. This is also a variation that does not continuously change the model, but at a single moment. Lastly, the way that continuous variables are reset after an event is often probabilistic.

The second way randomness might be introduced is through the continuous evolution. In many situations it is too idealistic to describe the continuous evolution by ordinary differential equations. These equations make the system deterministic, which means that as long as no event happens, the evolution is entirely determined by the initial state. However, in many situations that we wish to model this is simply not the case. There are many factors beyond our knowledge. It is often much more accurate to take these unknown factors into account as stochastic inference or diffusion. The ODEs then become stochastic differential equations, or SDEs. Such things as wind, or random deviation by our (only partially accurate) altitude meter might be too important to leave out, but beyond our knowledge. SDEs express these kinds of variations.

2.3 Introduction of the Example

To make the above more clear we will provide an example. We will discuss a simplified model of an aircraft landing on a runway [AEN12]. The reader should keep in mind that this example has been created at NLR to explain aspects of SDCPN modeling, but never for safety assessment purposes. SDCPN models used for safety assessment may be larger and more detailed than the example we present here.

The model concerns an aircraft, and the pilot in this aircraft, during a landing procedure. The model starts at touchdown and ends when the aircraft has stopped moving. An important discrete event in this situation is the initiation of the braking action. This is done after the aircraft has already traveled 400 to 600 meters on the runway. The general situation is portrayed in figure 2.1.

Another important aspect of the model is that it recognizes two modes of braking; nominal and non-nominal. Braking will virtually always happen under normal conditions and result in expected behavior. However, on average once in every 10000 landings, due to technical failure or otherwise, will braking happen at a decreased rate. This is called non-nominal braking. The model should take this into account.

In order to create a clear picture of the system, we will divide it into two important sections. The first is the aircraft itself, how it brakes and how it performs on the runway. Secondly we consider the pilot, and his influence on the braking procedure. The aircraft itself can be divided into two areas of interest. The first is how it performs on the runway, the second is its modus of braking. This creates an overall picture as presented in figure 2.2. AcEvolution records the situation of the aircraft on the runway, Braking performance registers the mode of braking, and PF stands for the pilot flying. The arrows indicate that there is a connection between some different sections. The pilot is obviously influenced by the information he

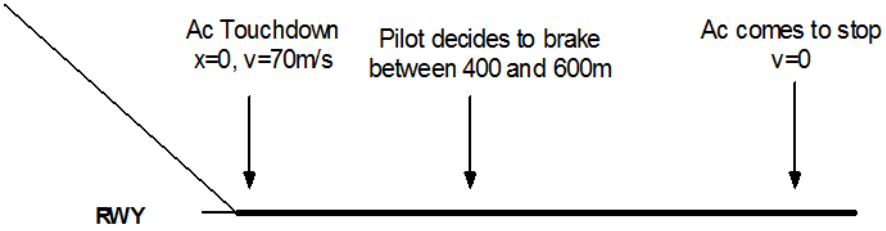


Figure 2.1: Example of a hybrid system, a landing aircraft [AEN12].

gets from the performance of the aircraft, and the performance of the aircraft is obviously influenced by the actions of the pilot. Furthermore, the performance of the aircraft on the runway is influenced by the mode of braking.

It is clear that we are dealing with a hybrid system. There are continuous aspects such as the distance traveled on the runway, the velocity and deceleration of the aircraft. There are also some clear discrete aspects such as whether the pilot has initiated the braking process or has not yet done so, and whether the aircraft is braking in nominal or non-nominal mode.

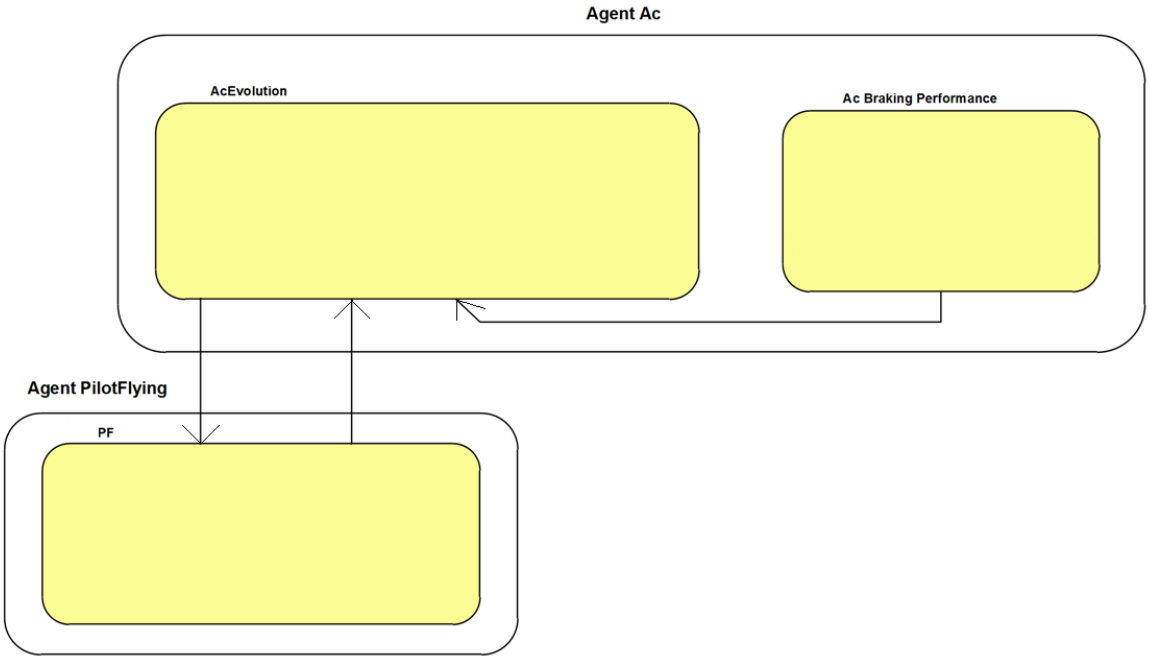


Figure 2.2: General relations in our model of a landing aircraft (based on [AEN12]).

2.4 Modeling Formalisms

In this section we will make the reader more familiar with the most important modeling formalisms for hybrid systems. This frame of reference can be helpful when we zoom in on the formalisms in chapter 4, where we will also be discussing the analytical tools. We will not give a mathematical definition, but rather a short qualitative description of what range of dynamics hybrid models can model and in what way they can do so.

The first important distinction that can be drawn between modeling formalisms is their capacity to model only discrete behavior or continuous behavior as well. One of the best known formalisms that can only model discrete behavior is finite state machine (FSM). A finite state machine is a finite set of states linked by a finite number of edges, which are labeled with actions [SS99]. The only dynamics on the system are discrete, that is the switching between states by taking action over some edge. Generally the input is defined by a word that describes which actions (edges) ought be taken in succession. Sometimes an output is generated based on the states the system visits. FSMs are often used to describe software systems.

Timed automata (TA) are FSM where each discrete state can have continuous dynamics in the form of clocks, that is, the variables increase with fixed rate [AD94]. A discrete transition can occur if the guard of a transition is fulfilled. Any constants used in guards are integers. Along each edge some clock resets can be specified. TA are often used for analyzing real-timed behavior of computer software.

Simple multi-rate timed automata (SMTA) are timed automata that allow skewed clocks, i.e. clocks that change with a different rate than 1. They are called simple because guards must check variables against constants, not against other variables.

The formalism we discuss next is Markov chains. A Markov chain is a set of states (or modes) which the system can be in, and some way to transit between them. Markov chains have the Markov property, which means that any dynamics of the system is independent of the past, conditioned on the current state. This means that we only need to know the current state to determine the probability of future paths.

Continuous-time Markov chain (CTMC) defines over each edge between states a value describing the exponential distribution parameter [Ajm89]. This parameter describes the exponential distribution that determines when an edge is taken and which one is taken.

Discrete-time Markov chain (DTMC) consists of a set of states and a transition probability matrix describing how likely it is that a state changes to another state per time step [Daw01]. Note that each row in the matrix must add to one, and selfloops are allowed.

If one wants to model not only discrete but also continuous behavior one can use a hybrid systems formalism. Most non-stochastic hybrid formalisms that we discuss in this thesis are special cases of hybrid automata (HA). A hybrid automaton is a finite state machine that has continuous dynamics defined on each of its states [SS99]. When the system does not evolve along an edge the variables of the system evolve continuously according to the dynamics of the current state. When the system does evolve via an edge the variables might be reset, and the dynamics might be changed. This opens up a huge range of possibilities. Generally, the edges between states can be taken conditionally on the current continuous state variables. These conditions are called guards. This formalism allows a huge range of phenomena to be modeled,

often concerning a machine (that acts in a discrete way) that interacts with its continuously changing environment.

A special case of hybrid automata that limits the expressiveness of hybrid automata somewhat is linear hybrid automata (LHA). All dynamics in the LHA formalism must be linear expressions of the relevant coefficients, as must all guards and resets be linear in the input. A hybrid system which does not meet these limiting criteria is called a nonlinear hybrid automaton (NHA).

Piecewise deterministic Markov process (PDP), introduced by Davis [Dav84], is a process that follows an ordinary differential equation ODE almost all of the time. However, discrete jumps can occur either when the continuous state hits some prespecified boundary or spontaneously, after a stochastic time that can be dependent on the continuous state. A jump changes the discrete state of the system, and can also be accompanied by a probabilistic reset map for the continuous state. PDP have the strong Markov property [Dav93], which means that they have the Markov property also on specific time instants called stopping times.

In PDP events are of a probabilistic nature, whereas the continuous behavior in between these events is deterministic. If one wants to model stochastic continuous behavior there is a range of possibilities in the literature. One such option is switching diffusion processes (SDP), which have a continuous stochastic evolution until a jump occurs [PH07]. A jump changes the dynamics of the system. The probability that a jump occurs and to what dynamics it changes the system is determined by the continuous variables. Switching diffusion processes are often seen as an extension of differential equations. They make it possible to model random environment elements that ordinary differential equations cannot model.

Another option is stochastic hybrid systems (SHS), which are PDP where the dynamics includes some stochastic variation [HLS00]. The transitions, unlike in PDP, occur only when some boundary of the domain of the current discrete state is reached, i.e. not also spontaneously.

There also exists a formalism called general stochastic hybrid systems (GSHS), which includes all dynamics of SHS, but does allow spontaneous jumps [BL04]. GSHS can also be seen as PDP where the ODEs that govern the continuous dynamics have been replaced by SDEs. It is the most inclusive modeling formalism in the literature of hybrid stochastic systems. GSHS have the strong Markov property [BL04].

GSHS evolve in continuous time, because they model the behavior of real systems evolving in time. However, in certain circumstances it is advantageous to discretize time. We then speak of discrete-time stochastic hybrid systems (DTSHS). All dynamics is equivalent to GSHS, but all functions are defined for timesteps. Thus when we speak of SDE in DTSHS, we mean stochastic difference equations. One can approximate a GSHS model by letting the timesteps go to 0.

Arenas of finite state machines (FSM) are collections of FSMs that interact concurrently through a network. An 'upper' FSM determines how the 'lower' FSMs interact. The formulism allows for compositional modeling, i.e. to first make the local FSMs and afterwards connect them together.

Communicating piecewise deterministic Markov chains (CPDP) are collections of PDPs that communicate through a network. An active transition in a local PDP can trigger passive transitions in other local PDPs to fire as well. This formulism also allows for compositional modeling.

Table 2.1: Characteristics of modeling formalisms

formalism	discrete/ continuous	ODE/ SDE	linear/ nonlinear	spontaneous transitions	forced transitions
FSM	discrete	none	neither	no	no
TA	both	ODE	linear	no	yes
SMTA	both	ODE	linear	no	yes
CTMC	discrete	none	neither	yes	no
DTMC	discrete	none	neither	yes	no
HA	both	ODE	nonlinear	no	yes
LHA	both	ODE	linear	no	yes
PDP	both	ODE	nonlinear	yes	yes
SDP	both	SDE	nonlinear	yes	no
SHS	both	SDE	nonlinear	no	yes
GSHS	both	SDE	nonlinear	yes	yes
DTSHS	both	SDE*	nonlinear	yes	yes
AFSM	discrete	none	neither	no	no
CPDP	both	ODE	nonlinear	yes	yes
PN	discrete	none	neither	no	no
DCPN	both	ODE	nonlinear	yes	yes
SDCPN	both	SDE	nonlinear	yes	yes

Petri net (PN) is a discrete formalism with places and transitions, and arcs between these two [Ajm89]. A place can have tokens that signify that that place is current. For example, a place named "airplane" with two tokens in it shows that in the model there are two airplanes. Transitions can be seen as events. A transition has input places, which are all places that have an arc to that transition, and output places, which are all places that the transition has an arc to. A transition is enabled if all of its input places have at least one token. If a transition fires, one token from each input place is taken away and one token is produced in each of the output places.

Dynamically colored Petri net (DCPN) adds to PN continuous variables to each of the tokens in all of the places [Eve10]. They evolve continuously over time by ordinary differential equations (ODE). It is possible for transitions to differentiate based on these variables, such that a transition is only fired if a condition, called guard, is satisfied over the tokens. If a transition fires, the number and colors of tokens produced in the ouput places is determined by a probabilistic function. DCPN have been shown to be pobabilistically equivalent to

Stochastically and dynamically colored Petri net (SDCPN) extends DCPN by including stochastic evolution of token colors [Eve10]. The token colors follow stochastic differential equations (SDE). SDCPN have been shown to be probabilistically equivalent to GSHS.

2.5 Overview of Hybrid Systems

Table 2.4 presents a summary of the modeling formalisms discussed above. For each formalism we include whether it has continuous, discrete or hybrid dynamics, the nature of the continuous dynamics where applicable, and the types of transitions included. With spontaneous transitions

we mean transitions which occur with some probabilistic transition rate. Forced transitions are transitions which the system forces, possibly dependent on the continuous state. Where SDE is complemented with an asterisk, or SDE*, we mean to say stochastic difference equation, as opposed to stochastic differential equation.

2.6 Analysis Problems

When researchers are dealing with hybrid systems, reachability analysis is often a main concern. The informal questions are of the form; "Is it possible that event x occurs within t units of time?", or, "What percentage of time is state x occuring?" In literature there is not one clear definition of reachability, so the different perspectives are described next. Each of these prespectives is a different type of analysis problem, i.e. a type of problem that a researcher might be dealing with, and for which he would like to find answers through analysis methods.

Safety Verification

Safety verification is often researched for non-probabilistic systems such as timed automata. The goal of this kind of reachability is to see whether it is possible to reach a target state from an initial state [PJP07]. If it is possible the solution is 'yes' and the system is unsafe, if not the answer is 'no' and the system is safe. The solution to safety verification often hinges on bisimulation to a finite system, for which it is a trivial task to check this kind of reachability, for example through a breadth-first or depth-first search. The finite time version of safety verification is of course also researched, and generally an answer to one is an answer to both questions. It can be useful to us if a negative answer to this question on one of our models is found, because then we are certain that a risk-state will never be reached.

Safety verification is also interesting for model checking and verification purposes. Virtually always will our models return yes answers, because we are only interested in possible risky states in the first place. If safety verification is analyzed and found to be not true for some model, it might be evidence that the model is wrong. Thus, safety verification can be used as an inital step in our analysis. First we find whether a state can be reached at all from our initial location. If it cannot, we have either found a solution to our question or we need to redesign the model. If it can, we analyze for a different kind of reachability.

Verification

Verification is safety verification generalized to any kind of property that the system can either have or not have. A property can be any assertion about the system, such as "variable x must always be lower than value y", or "between event a and any other event is always at least t seconds", or any other assertion one can think of. Generally however, researchers look for properties that can be expressed in some kind of logic. A system is verified for such a proposition if the proposition is always true in all reachable states.

Falsification

In falsification one tries to find witness trajectories from the intial state to the target state. If a witness trajectory is found, this proves that the system is unsafe, i.e. it is possible to reach a target state. If no witness trajectory can be found, this is generally no proof that the system

is safe, only an indication that it is so. The algorithms for falsification sample only a finite number of the infinite number of possible paths.

Probabilistic Reachability

Probabilistic reachability describes how likely it is for a run, starting at an initial state within a specified region, to reach a state in the target region within either finite or infinite time. In certain cases this answers our research question. In the example explained above, the question "What is the probability that a run will exceed the 2000 meters long runway during landing" is a probabilistic reachability question.

Steady State Distribution

The steady state distribution gives to each place in $\mathcal{P}$ a percentage in $[0, 1]$ that denotes the percentage of time the system spends in that place. All of these percentages add up to one. Often, the steady state distribution is exactly what we are looking for. If we are interested in some dangerous place P_i and find in the steady state distribution that it is current p_i percent of time, we know how often the system is in a dangerous situation.

Complexity Reduction

Many algorithms that can answer the research questions above have a single- or double-exponential dependency on the dimension of the state space of the model. This limits their practicality to relatively small and simple models. One way of delaing with is is to reduce the complexity of the model. This means that the discrete state space or continuous state space is reduced, while the model still shows the same dynamics. This smaller yet equivalent model is then used to perform analysis on. Monte Carlo simulations are also faster on smaller models.

2.6.1 Decidability

A property, like the reachability properties above, is said to be decidable if it is possible to create an algorithm that, for any given system, determines in finite time whether that system has the property. Generally we talk about decidability of binary questions such as safety verification. Decidability is a valuable property because if a formalism is decidable for a given property, then that property can be effectively computed irrespective of the model we are looking at. We do not need to compute in an ad hoc manner. Often, first the decidability of a property for some formalism is determined, and later mathematical software is developed to compute the property for any input model.

When not otherwise specified, when we say decidable or undecidable in this thesis, we mean the decidability of safety verification.

Chapter 3

SDCPN

One of the most powerful formalisms used to describe hybrid systems is stochastically and dynamically colored Petri nets. This chapter first presents the basic idea of a Petri net. It proceeds by extending the original definition to dynamically colored Petri net (DCPN), which includes continuous evolution. This extended model is altered to include diffusion terms as well, as stochastically and dynamically colored Petri net (SDCPN). The chapter concludes with a discussion on the way that SDCPN-based models are used at NLR. From this discussion we try to extract what kind of analytical tools are important to realize for SDCPN.

3.1 Petri Nets

Petri nets derive their name from Carl Adam Petri, who developed them in his dissertation in 1962 [Pet62]. They are networks of places and transitions, linked by arcs. As we consider only marked Petri nets, a place can contain one or more tokens, which model that that place is current, i.e. it is true in one or more instances. Arcs link places with transitions and transitions with places, but never a place with a place or a transition with another transition. Arcs define how tokens can be moved. If all preconditions of a transition are fulfilled, that is, all incoming arcs leave from places with at least the desired number of tokens, that transition is called 'enabled'. An enabled transition can 'fire'. If it does so it removes one token from each of its input places and produces one token for each of its output places. Thus we have discrete evolution of our system.

Before we move on to a more rigorous definition of Petri net we consider Figure 3.1 on page 12. Places are symbolized by circles, transitions by rectangles, arcs by arrows and tokens by dots. This Petri net has four places, two transitions and seven arcs. Initially, P_1 and P_4 have a token, P_3 has two tokens. The only transition that is enabled is T_1, because its input place P_1 has a token. T_2 is not enabled, since its input place P_2 is empty. Assume that T_1 indeed does fire. The evolution would cause P_1 to be empty, P_2 would gain one token and P_3 has three in total. Now T_1 is enabled no longer, while T_2 does have all its preconditions for firing fulfilled. Assume T_2 fires. T_2 eats a token from both P_2 and P_3, and sends one to P_1 and one to P_4. We see that the system has returned to its initial marking, except for a gain of one token for P_4. Thus, in this system T_1 and T_2 can fire alternately, and all 'reachable' markings are those we have seen, with the addition that an arbitrary number of tokens larger than or equal to one can reside in P_4.

From the above discussion we can see that a marked Petri net is a directed bipartite graph

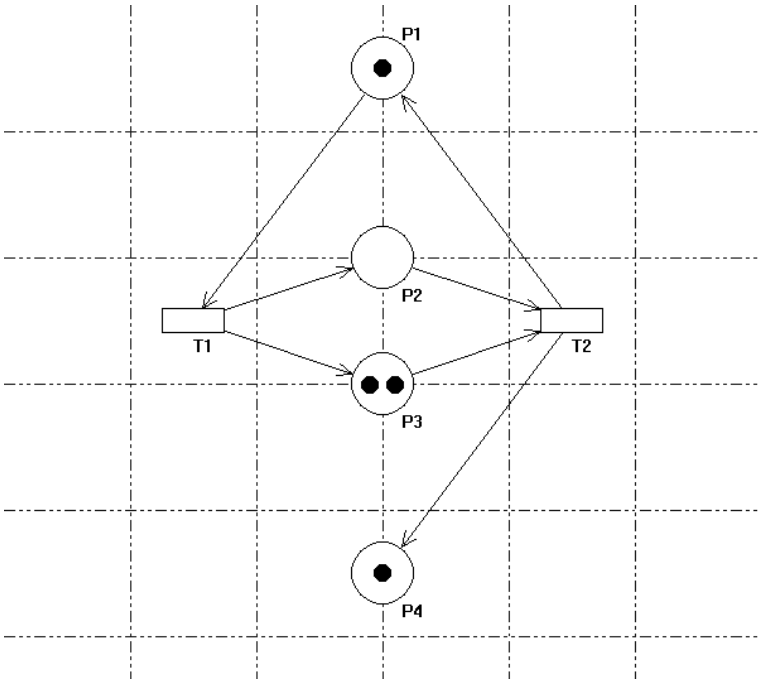


Figure 3.1: An example of a marked Petri net

with an initial marking. Mathematically, it is a quintuple $(\mathcal{P}, \mathcal{T}, \mathcal{A}, \mathcal{N}, \mathcal{M}_0)$ consisting of the following elements [Eve10]:

- $\mathcal{P} = \{P_1, P_2, ..., P_m\}$ is a finite set of places;
- $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ is a finite set of transitions, such that $\mathcal{T} \cap \mathcal{P} = \emptyset$;
- $\mathcal{A} = \{A_1, A_2, ..., A_o\}$ is a finite set of arcs such that $\mathcal{A} \cap \mathcal{T} = \mathcal{A} \cap \mathcal{P} = \emptyset$;
- $\mathcal{N} : \mathcal{A} \to (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ defines for each arc in $\mathcal{A}$ which two elements it connects and in what direction;
- $\mathcal{M}_0 = (M_1, M_2, ..., M_m)$ defines for each place from $\mathcal{P}$ how many tokens it contains initially.

Note that under the given definition of Petri net it is possible that there are multiple arcs in $\mathcal{A}$ that connect the same transition and place in the same direction. This models the situation where one transition eats multiple tokens from a place in order to fire, or that a transition gives multiple tokens to a place when it fires.

3.2 Dynamically Colored Petri Nets

In Petri nets as described above, transitions cannot differentiate based on properties of tokens, because tokens have no properties. The logical next step to extend the Petri net formalism, and to make it more expressive, is to color some tokens and thereby give them a state within each place in $\mathcal{P}$. We then talk about colored Petri nets.

A color is a property defined over a prespecified range. For instance, the color of a token may be 'Boeing 777-200', 'off' or 'red'. This would of course be dependent on the context of the model. Token colors make the model more expressive, as now we can model transitions that accept only certain tokens. For instance, a transition that determines which airplanes land on a given landing strip might accept Boeing 777-200 but not larger types.

Coloring becomes truly interesting when we accept continuous states associated with each token, that can evolve continuously. The fact that each token now represents actual data that continuously evolve according to given laws of change makes the model much more adaptive and gives it the power to model many different phenomena. In [Eve10], this class is referred to as dynamically colored Petri net, or DCPN. This type of coloring works in the following way. Each place is given a domain $\mathbb{R}^n$, for some n that differs per place. Each token color in that place lives in the given domain. The values of the tokens evolve continuously through ordinary differential equations (ODE). Effectively, each place has become a dynamical system.

Each token stays in its place unless a transition occurs. This can happen in three ways. The first is through an immediate transition ($T \in \mathcal{T}_I$). If all input places of an immediate transition have the required number of tokens, the transition will remove immediately all input tokens and create tokens in the output places according to a firing measure assigned to the transition. The second way is through a delay transition ($T \in \mathcal{T}_D$). A delay transition needs all its input places to have the required number of tokens, and besides this that an exponentially distributed amount of time has passed since, in order to be enabled and fire. The third way a transition can occur is through a guard transition ($T \in \mathcal{T}_G$). A guard transition needs all its input states to have the amount of tokens it requests, and also that some constraint on the token colors has been fulfilled. If that is the case a guard transition fires. The specific constraint for a guard transition and the exponential parameter of a delay transition are specified by the model.

DCPN also includes three types of arcs; ordinary arcs, enabling arcs and inhibitor arcs. Ordinary arcs ($A \in \mathcal{A}_O$) behave as we have seen in non-colored Petri nets. If place P_i is connected to transition T_j with an ordinary arc leaving P_i and entering T_j and transition T_j fires, place P_i loses a token. Similarly, if place P_i is connected to transition T_j with an arc entering P_i and leaving T_j and transition T_j fires, some given distribution determines the chance that place P_i gains a token, and the color of that token.

Arcs that start at a place and end at a transition can also be one of two different types of arcs. They can be enabling arcs ($A \in \mathcal{A}_E$), which means that the input place needs a token for the transition to fire, but when that transition fires it does not consume a token from that place. They can also be inhibitor arcs ($A \in \mathcal{A}_I$), which allow transitions to fire only if that input place does *not* have a token.

At this point we have defined certain aspects of dynamically colored Petri nets, but we have yet to give a more formal and complete view of a DCPN. Mathematically, a DCPN is an undecuple with elements $\{\mathcal{P}, \mathcal{T}, \mathcal{A}, \mathcal{N}, \mathcal{S}, \mathcal{C}, \mathcal{I}, \mathcal{V}, \mathcal{G}, \mathcal{D}, \mathcal{F}\}$ [Eve10], where:

- $\mathcal{P} = \{P_1, P_2, ..., P_{|\mathcal{P}|}\}$ is a finite set of places. The dimension of $\mathcal{P}$, i.e. the total number of places, is denoted by $|\mathcal{P}|$;

- $\mathcal{T} = \{T_1, T_2, ..., T_{|\mathcal{T}|}\}$ is a finite set of transitions, where $|\mathcal{T}|$ denotes the total number of transitions. A place cannot be a transition also, so $\mathcal{P} \cap \mathcal{T} = \emptyset$. As we distinguish three types of transitions, we can partition the set $\mathcal{T}$ into three sets; $\mathcal{T}_I$ for immediate transitions, $\mathcal{T}_D$ for delay transitions and $\mathcal{T}_G$ for guard transitions. They are distinct

sets, so $\mathcal{T}_I \cap \mathcal{T}_D = \mathcal{T}_G \cap \mathcal{T}_I = \mathcal{T}_G \cap \mathcal{T}_D = \emptyset$ and they constitute the entire set of $\mathcal{T}$, so $\mathcal{T} = \mathcal{T}_I \cup \mathcal{T}_D \cup \mathcal{T}_G$;

- $\mathcal{A} = \{A_1, A_2, ..., A_{|\mathcal{A}|}\}$ is a finite set of arcs never coinciding with either places or transitions, so $\mathcal{T} \cap \mathcal{A} = \mathcal{P} \cap \mathcal{A} = \emptyset$. $|\mathcal{A}|$ is the size of the set $\mathcal{A}$. Three sets of arcs are identified; ordinary arcs $\mathcal{A}_O$, enabling arcs $\mathcal{A}_E$ and inhibitor arcs $\mathcal{A}_I$. They are distinct sets, so $\mathcal{A}_O \cap \mathcal{A}_E = \mathcal{A}_O \cap \mathcal{A}_I = \mathcal{A}_E \cap \mathcal{A}_I = \emptyset$ and they constitute the entire set of $\mathcal{A}$, so $\mathcal{A} = \mathcal{A}_O \cup \mathcal{A}_E \cup \mathcal{A}_I$;

- $\mathcal{N} : \mathcal{A} \to (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ defines for each arc in $\mathcal{A}$ which two elements it connects and in what direction. Notice that both enabling and inhibitor arcs are only defined on $(\mathcal{P} \times \mathcal{T})$. For notational clarity, we define the following:

  - $A(T) = \{A \in \mathcal{A} | T(A) = T\}$ is the set of arcs connected to T.
  - $A_{out}(T) = \{A \in A(T) | \mathcal{N}(A) = (T, P(A))\}$ is the set of output arcs of T.
  - $A_{in,OE}(T) = \{A \in A(T) | \mathcal{N}(A) = (P(A), T) \cap A \in \mathcal{A}_E \cup \mathcal{A}_O\}$ is the set of input arcs of T that are either ordinary or enabling arcs;

- $\mathcal{S} \subset \{\mathbb{R}^0, \mathbb{R}^1, \mathbb{R}^2 ...\}$ is a finite set of color types. We define $\mathbb{R}^0 \equiv \emptyset$;

- $\mathcal{C} : \mathcal{P} \to \mathcal{S}$ maps each place $P \in \mathcal{P}$ to one color type in $\mathcal{S}$. The color type in $\mathcal{S}$ that a place is mapped to determines the dimension of the tokens residing in that place. So, $\mathcal{C}(P) = \mathbb{R}^{n(P)}$, where $n : \mathcal{P} \to \mathbb{N}$. If a place is mapped to $\mathbb{R}^0$ then its tokens have no color;

- $\mathcal{I} : \mathbb{N}^{|\mathcal{P}|} \times \mathcal{C}(\mathcal{P})^{\mathbb{N}} \to [0, 1]$ is a probability measure, which defines the initial marking of the net. It defines per place a non-negative number of initial tokens, and it defines their initial colors. $\mathbb{N}^{|\mathcal{P}|}$ stands for the space of finite non-negative integer vectors of dimension $|\mathcal{P}|$, or $\mathbb{N}^{|\mathcal{P}|} = \{(m_1, m_2, ..., m_{|\mathcal{P}|}) |\ m_i \in \mathbb{N}, m_i < \infty, i = 1, 2, ..., |\mathcal{P}|\}$. $\mathcal{C}(\mathcal{P})^{\mathbb{N}}$ denotes the set of Euclidian spaces defined by $\mathcal{C}$ over each P; thus constituting $\mathcal{C}(\mathcal{P})^{\mathbb{N}} = \{\mathcal{C}(P_1)^{m_1} \times ... \times \mathcal{C}(P_{|\mathcal{P}|})^{m_{|\mathcal{P}|}} |\ m_i \in \mathbb{N}, m_i < \infty, i = 1, 2, ..., |\mathcal{P}|\}$, where $\mathcal{C}(P_i)^{m_i} = \mathbb{R}^{m_i * n(P_i)}$ for all $i = 1, 2, ..., |\mathcal{P}|$, $\mathcal{C}(P) = \mathbb{R}^{n(P)}$ and $\mathcal{P} = \{P_1, P_2, ..., P_{|\mathcal{P}|}\}$.

- $\mathcal{V} = \{\mathcal{V}_P : P \in \mathcal{P} | C(P) \neq \mathbb{R}^0\}$ is the set of token color functions. Per place it defines the drift coefficient of the dynamics in that place. For each place it contains a map $\mathcal{V}_P : \mathcal{C}(P) \to \mathcal{C}(P)$. The mapping stands for the differential equations defining the dynamics in place P, i.e. $dC_t = \mathcal{V}_P(C_t)dt$. We assume that the mapping allows for a unique solution. If a place P is mapped to $\mathbb{R}^0$ according to $\mathcal{C}$ then that place has no continuous dynamics on its tokens;

- $\mathcal{G} = \{\mathcal{G}_T : T \in \mathcal{T}_G\}$ is the set of transition guards. For each $T \in \mathcal{T}_G$ it contains a transition guard which is an open subset of $C(P(A_{in,OE}(T)))$ with boundary $\partial\mathcal{G}_T$. $C(P(A_{in,OE}(T)))$ is defined as the cross product of the color types of the input places of T, that are connected to T by either an ordinary or enabling arc.

- $\mathcal{D} = \{\mathcal{D}_T : T \in \mathcal{T}_D\}$ is the set of transition delay rates. Each transition delay can be a function over the colors of the input tokens, yet it is generally exponentially distributed, independent from those colors;

- $\mathcal{F} = \{\mathcal{F}_T : T \in \mathcal{T}\}$ is a set of firing measures. For each $T \in \mathcal{T}$, it contains a firing measure $\mathcal{F}_T : \{0,1\}^{|A_{out}(T)|} \times \mathcal{C}(P(A_{out}(T))) \times \mathcal{C}(P(A_{in,OE}(T))) \to [0,1]$, which generates the number and values of the tokens produced when transition T fires, given the value of the vector in $\mathcal{C}(P(A_{in,OE}(T)))$ that collects all input tokens. Here,
  $\{0,1\}^{|A_{out}(T)|} = \{(e_1, ..., e_{|A_{out}(T)|}) |\ e_i \in \{0,1\},\ i = 1, ..., |A_{out}(T)|\}$
  and if $P(A_{out}(T)) = \{P_{O_1}, ..., P_{O_{|A_{out}(T)|}}\}$ then
  $\{0,1\}^{|A_{out}(T)|} \times \mathcal{C}(P(A_{out}(T))) = \{(e^T, a^T); e^T = (e_1, ..., e_{|A_{out}(T)|}),$
  $e_i \in \{0,1\},\ a^T \in \mathbb{R}^{n_{out}(T)},\ n_{out}(T) = \sum_{i=1}^{|A_{out}(T)|} e_i \cdot n(P_{O_i})\}$.

  In other words, for $i = 1, ..., |A_{out}(T)|$, if $e_i = 1$ then the ith output place P_{O_i} of T gets a token with value in $\mathbb{R}^{n(P_{O_i})}$ and if $e_i = 0$ then the ith output place of T does not get a token. The vector e^T thus denotes which output place get a token and the vector a^T collects all colors of tokens produced.

Now we explain how the described elements interact in order to form a run of a DCPN. To initialize the system, we determine how many tokens each place has initially through probability measures in $\mathcal{I}$. The token colors will also be determined by $\mathcal{I}$. As we do this by a probability measure, this initalization differs each run of the model.

Once initiated, the tokens will evolve according to the flow of the place they reside in. This happens until a transition is enabled. A guard transition ($T \in \mathcal{T}_G$) is enabled if each of its input places have a token and the colors of these tokens are on the boundary of the guard. A delay transition ($T \in \mathcal{T}_D$) is enabled if each of its input places have a token and an exponentially distributed time has lapsed since. An immediate transition ($T \in \mathcal{T}_I$) is enabled if all its input places have tokens. When a transition fires, it removes tokens from its input places and produces tokens and their colors according to a firing function in $\mathcal{F}$. It is possible that after a firing immediate transitions are enabled. They fire in an ordering according to rules specified here [Eve10]. When an immediate transition fires, it removes tokens from its input places and produces tokens and their colors according to a firing function in $\mathcal{F}$. Once no more immediate transitions are enabled, the tokens again follow differential equations of their respective places as specified in $\mathcal{V}$.

3.3 Stochastically and Dynamically Colored Petri Nets

Petri nets would become more powerful if they could account in more ways for random variation within the data. Besides the randomness that is included through delay transitions, SDCPN also accepts randomness in its continuous evolution [Eve10]. Instead of using ODEs to govern its continuous behavior, the solutions follow SDEs, stochastic differential equations. These are composed of a dynamic and a stochastic element, $dx_t = \mathcal{V}(x_t)dt + \mathcal{W}(x_t)dw_t$. Generally in SDCPN, the kind of stochastic diffusion present is Brownian Motion. In this way SDCPN can model more types of phenomena than DCPN, as many natural occurrences have essential unknown variation. Though this looks like a minor adjustment, it makes a vast difference in the behavior of the system. In DCPN we had completely determined the color of a token by its initial value within that place, as long as it did not jump. In SDCPN the token colors cannot be uniquely predicted.

Mathematically, an SDCPN is a duodecuple $\{\mathcal{P}, \mathcal{T}, \mathcal{A}, \mathcal{N}, \mathcal{S}, \mathcal{C}, \mathcal{I}, \mathcal{V}, \mathcal{W}, \mathcal{G}, \mathcal{D}, \mathcal{F}\}$, with the eleven elements that coincide with the DCPN defined exactly like a DCPN, and $\mathcal{W}$ defined as:

- $\mathcal{W} = \{\mathcal{W}_P : P \in \mathcal{P} |\ \mathcal{C}(P) \neq \mathbb{R}^0\}$ is a set of token color matrix functions. For each place with continuous dynamics on its tokens, it contains a measurable mapping $\mathcal{W}_P : \mathcal{C}(P) \to \mathbb{R}^{n(P)\times h(P)}$ that defines the diffusion coefficient of a stochastic differential equation, where $h : \mathcal{P} \to \mathbb{N}$ is the dimension of the Brownian motion and $n : \mathcal{P} \to \mathbb{N}$ is such that $\mathcal{C}(P) = \mathbb{R}^{n(P)}$, in other words it is the size of the dimension of the token colour. Together with $\mathcal{V}$ this function describes the SDE of each place.

3.4 SDCPN: An Example

To complete our explanation of SDCPN, and to make matters more concrete, we will describe an SDCPN model based on the example introduced earlier. The system has seven places, five transitions of which one immediate transition, two guard transitions and two delay transitions. It has 13 arcs of which three enabling arcs (indicated by the arrows that end with a black circle, instead of an arrowhead) and 10 ordinary arcs, see figure 3.2. To aid in the understanding of more complex models, we define two ways of grouping SDCPN elements. The first is in local Petri nets, or LPN, which are transitions and places that together express a particular entity of the system. If the engine can be in several modes, the places and transitions that are concerned with those modes together form an LPN. An LPN always contains exactly one token. Multiple LPN can be grouped in an agent. Agents are the natural elements of a system. In our example, we identify two agents; the pilot and the aircraft. The pilot contains only one LPN, whereas the aircraft contains two LPNs; the evolution of the aircraft and its braking performance. We see that one arc, starting at LPN braking performance and ending at the immediate transition, does not start at a place, but at the box surrounding the LPN. Following [Eve10] chapter 5, this enabling arc represents the situation where the immediate transition can use the information from the LPN braking performance no matter where in the LPN the token resides.

We start by examining LPN PF in the Pilot Flying agent. It is initiated in P_1, where the pilot has not yet initiated the braking process. The token of this place is given an initial value x, uniformly distributed between 400 and 600. This value represents the distance to be traveled by the plane before the pilot decides to brake, and it starts evolving according to $\dot{x} = 0$. The guard transition G_1 checks whether the distance traveled on the runway, which it gets from place Ac on RWY in LPN AcEvolution, exceeds the token value x in P_1. If it does, the transition removes the token from P_1, and a token is produced for P_2, without color.

Now we turn to LPN AcEvolution. This LPN is initiated with a token in P_1, with initial color $(x, v, a)'$, where v is the velocity initiated at $v = 70m/s$, x the distance on the runway equal to 0, and a the deceleration equal to 0 (the pilot has not yet started braking). There is no change in v and a in this place, and the flow on x is $\dot{x} = v$. Once the pilot decides to brake, the immediate transition fires. The firing function on I_1 creates a new token in P_2, Ac brakes on runway. The value of the token produced is equal to the value of the token removed from P_1, except a is determined based on a Gaussian distribution with parameters determined by whether LPN Braking Performance is in nominal or non-nominal mode. Next, the value of the token evolves according to $\dot{a} = 0$, $\dot{v} = a$, $\dot{x} = v$. The guard transition G_1 fires when v is below 0. It removes the token from P_2 and produces a token in P_3, with a color equal to the color of the token removed. A token entering this state represents the simulation stopping.

Lastly we look at the local Petri net Braking Performance. This LPN can be initiated in either place. It is initiated in nominal a percentage of time equal to the likelihood of an

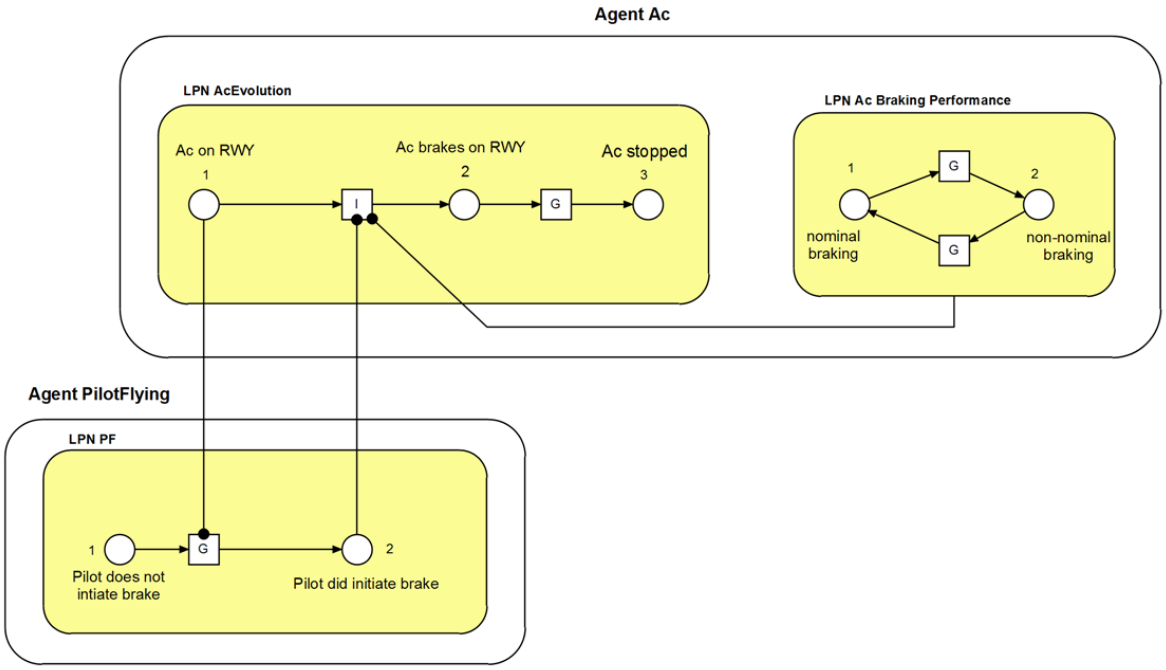


Figure 3.2: SDCPN-based model of a landing aircraft [AEN12].

aircraft being in nominal mode, and likewise for non-nominal. The delay transitions fire with an exponentially distributed delay equal to 1 divided by the chance an aircraft is in nominal mode for D_2 and 1 divided by the chance an aircraft is in non-nominal mode for D_1.

The example in a stochastic environment

The above example uses no diffusion, so can technically be called a DCPN as well. If researchers are interested in the same landing behavior in a windy situation, the model would have to be adapted. In all places of LPN AcEvolution the acceleration a can be decreased (so that the plane brakes quicker) by some number, which represents the average wind in opposite direction to the aircraft's position. A stochastic measure, such as the Weiner process, can be added to the differential equation of the flow of v to indicate the randomness of the blasts of wind.

3.5 Analysis Methods for SDCPN

In this section we discuss shortly the current analysis methods for SDCPN. These are already extensively used to analyze SDCPN-based models.

The first analysis method is Monte Carlo simulation. One runs the model, i.e. one allows the model to evolve through discrete and continuous evolution, and counts how often a particular situation occurs. To detect rare events one needs to run the model very often. This means that this analysis method can be time consuming.

The second analysis method would be to make use of the equivalence between DCPN and PDP or SDCPN and GSHS [Eve10]. PDP and GSHS have the strong Markov property,

so SDCPN-based models have this property as well. The strong Markov property makes it feasible to stop Monte Carlo simulations, and restart them without losing information. This can make Monte Carlo simulations more efficient. More on this can be found in chapter 4.11.

Thirdly, in certain situations part of an SDCPN model behaves equivalently to a model from another formalism. This is currently used when a local Petri net exclusively has delay transitions, because such sub-systems behave equivalently with CTMC models. This provides the opportunity to apply analysis methods of CTMC to parts of the SDCPN model that fulfill specific restrictions. Solving part of a model analytically through these acquired methods can significantly speed up the simulation process.

3.6 Research Question

SDCPN has much resemblance to other systems. The purpose of this thesis is to explore whether and how existing analysis methods for other types of hybrid systems are, or can be made, applicable to SDCPN.

Chapter 4

Analysis Methods for Hybrid Modeling Formalisms

There are many types of models, besides stochastically and dynamically colored Petri nets, that have been used to simulate hybrid systems. These differ in many aspects, such as how general they are, what kind of processes they allow to be modeled, what kind of analytical methods can be used on such models, etcetera. This thesis is interested in finding formalisms that allow mathematical analysis tools, which we can put to use solving the questions discussed in the previous chapter. When a formalism is said to have an analytical tool, we mean: if it is possible to fit a hybrid system into such a modeling formalism, it is possible to determine certain properties or use given analytical tools to further investigate that model.

The aim of this chapter is to give an overview of the techniques that the literature has developed in this regard. This chapter starts off with a general overview of strategies used to find analytical tools. Next we discuss the notion of abstraction and bisimulation, as these are important concepts for almost all analytical tools available. The rest of this chapter introduces a broad range of modeling formalisms and analytical tools developed for them.

4.1 General Overview

The first general method that was developed in the literature is model checking for safety verification purposes. One identifies an initial state and moves in steps to all possible future states. If eventually an undesired state is reached, the system is not safe, i.e. a dangerous state can be reached. In the case of finite automata, states can be represented by enumeration. In that case, forward reachability computation starts with the initial state and adds the one-step successors (all neighboring states) until convergence is achieved. Therefore verification is decidable for finite systems. For infinite systems this computation might fail as convergence is not guaranteed in finite time. Decidability of some specific infinite systems has been proven by showing that one can build a finite automaton that is equivalent to the original system.

In most cases it is not possible to reduce an infinite system to a finite system. For those cases two strategies have been developed. The first strategy researchers employed was to try to approximate the system to a simpler system. The dynamics of the system are made easier. This is done via over-approximation or asymptotic approximation. In over-approximate methods one gives a conservative answer to the problem, so if one finds out that verification is false in the approximated case, it is also false in the original system. However, the 'yes'

answer does not necessarily indicate a 'yes' answer in the original system. In asymptotic approximation one can choose the approximation factor such that the original system can be approximated to an arbitrary degree of accuracy, often at a cost of reduced computational ease.

The second strategy is to approximate the states that have to be propagated through the system dynamics. Here too asymptotic and over-approximative methods are possible. Over-approximation can be given by polyhedra, convex hulls, etc. Asymptotic approximation can be achieved by level-set methods and gridding.

For reachability problems beyond verification researchers generally abstract or approximate to Markov chains. Both for discrete-time and continuous-time Markov chains analysis is available to determine such properties as steady-state distributions and reach-probabilities. Most generally, one uses asymptotical convergence to Markov chains.

4.2 Abstractions

As can be seen in the discussion above, at the center of many theorems about hybrid systems is the notion of abstraction. The strategy to abstract complex systems to simpler ones in order to benefit from known analytical tools is one of the most thoroughly investigated techniques in the literature. It is important to understand the notion of abstraction in order to understand the theorems themselves, but also to get a grasp of the methodology of research concerning hybrid systems.

An abstraction is a mapping from one formalism into another, such that these systems are shown to be mathematically equivalent with respect to some property, even though some details have been left out [Pap03]. If a system is an abstraction of another, one could see this as being different perspectives of essentially the same object. A bisimulation is a partition of the state space that preserves a property of interest. It can be seen as an abstraction between models both from the same formalism.

There also exists the notion of approximate abstraction, or approximation. An approximate abstraction is an abstraction between two systems that are not equivalent, but 'close enough'. Some of the complexity of a system that does not have much effect on the property to be investigated might be left out, for example. The system that the original system is approximated into has simpler analysis, and the same answer to the investigated question, to some degree of accuracy. The fact that an abstraction is approximate enough obviously needs to be defined and proven before the approximation is accepted. Generally, a formalism is shown to approximate asymptotically in some variable that is independent of the system at hand, so that the researcher himself can choose the approximation error. However, as a rule of thumb in approximations; the better a system approximates another, the bigger its statespace gets, in which case other problems arise.

4.3 Analysis Methods for Finite State Machines

An automaton (plural: automata), is a self-operating object. It runs on an input in discrete time. At each timestep, the system is in a state, and can jump to other states in a discrete manner. Automata theory is a relatively young branch of mathematics, but it has recently, that is over the last few decades, been popularized due to its use to computers and embedded software.

4.3.1 Description

A finite state automaton, or finite state machine (FSM), is an automaton with a finite number of discrete states. Though it can be defined in slightly different ways, we opt here for the Moore definition. Mathematically, a finite state Moore machine is a hextuple $(S, s_0, \Sigma, \Lambda, G, T)$ [PPBS12] where

- $S = \{s_1, s_2, ..., s_n\}$ is a finite set of states;
- $s_0 \in S$ is the initial state;
- Σ is a finite set of input symbols, or input alphabet;
- Λ is a finite set of output symbols, or output alphabet;
- $G : S \to \Lambda$ is the output function.
- $T : S \times \Sigma \to S$ is the state transition function;

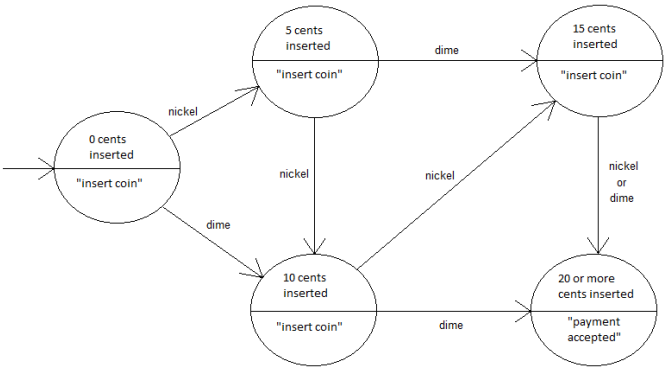


Figure 4.1: Example of a finite state machine

Evolution

Initialization: current state $\leftarrow s_0$
while *next input symbol* $\neq \emptyset$ **do**
  symbol $\leftarrow$ read next input symbol
  current state $\leftarrow T(current\ state, symbol)$
  write output $G(current\ state)$
end

In example 4.1 s_0 = "0 cents inserted" as indicated by the loose arrow entering this state. A word is a series of input symbols. Each symbol within the word describes the next transition that should be taken. The machine accepts words which consist of the symbols "dime" and "nickel" only. If one would insert a dollar bill, then the machine would simply not know what to do with this. This is expressed by the fact that dollar bill is not a possible input symbol

(on the labels). FSM models do not give information on what paths are likely to be taken, only on what paths are possible. An admissable word could look like this:

$$(nickel, dime, nickel)$$

The first symbol of the input is "nickel", therefore the FSM moves to state "5 cents inserted". The next symbol is "dime", so the FSM moves to "15 cents inserted". The last symbol is "nickel", therefore the FSM will move to "20 or more cents inserted". The output of the machine is "insert coin" as long as the total amount is below 20 cents, and "payment accepted" if it is above or on 20 cents.

4.3.2 Analysis Method 1: Breadth First Search for FSM

Breadth first search determines whether a model, starting at the initial state, could ever reach a set of unsafe nodes. This technique first determines the set of states that can be reached, and then checks whether the intersection of the reach set and the unsafe set is non-empty. Such "breadth-first" search finds all reachable states from s_0 in polynomial time with respect to the number of states in S and the number of transitions defined in T. Therefore, we say that safety verification for finite systems is decidable.

A breadth first search can be done as follows [BM06]. Start at the initial state, denote this as the current set of states at iteration 0, C_0. At each iteration we are going to find the next set by including all possible states that can be reached from the current set by at least one input symbol. It is important to note that $C_i \subset C_{i+1}, \forall i \in \mathbb{N}^+$. As the statespace is finite, we have at some iteration k: $C_k = C_{k+i}\ \forall i$. This set is called the fixpoint solution of the forward reachable set. These are all reachable states. Any state not in the fixpoint solution is not reachable from the initial state, as all input words have been considered. It is plain to see that this analysis can also be performed starting from a given initial set of states, rather than from a single initial state. The algorithm can be written thus;

Initialization: $C_0 = s_0, C_{-1} = \emptyset, i = 0$
while $C_i \neq C_{i-1}$ **do**
  $C_{i+1} \leftarrow C_i$
  for *each state* $c \in C_i / C_{i-1}$ **do**
    $C_{i+1} = C_{i+1} \cup$ (neighbors of c)
  end
  $i = i + 1$
end

In figure 4.1 we have s_0 ="0 cents inserted", so $C_0 =$ { "0 cents inserted"}. We consider all states that can be reached from C_0 and find $C_1 =$ {"0 cents inserted", "5 cents inserted", "10 cents inserted"}. C_2 comprises the entire state space, so this must be the fixpoint solution. We find that all states are reachable from "0 cents inserted".

4.3.3 Analysis Method 2: Model Checking for FSM

Model checking is a broader analysis method than reachability analysis. Its goal is to establish a model's correctness [BK08]. A researcher can identify certain properties that should always be true in the system, and use model checking to find whether they actually are true. If a model does not uphold such property, the model is flawed. A proposition could be "the airplane never gets an altitude greater than x", or "after an emergency the pilot will always contact

the Air Traffic Control Center". These properties must be translated to formulae in some property specification language, such as computational temporal logic (CTL). Computational temporal logic is a propositional logic extended with operators. It can specify precisely a large range of propositions, and is explained as follows:

A proposition is made up of atomic propositions. The proposition "a day of rest is always followed by a day of work" is made up of two atomic propositions, "day of rest" and "day of work". In order to model check an FSM for a given proposition, we need to know for each state whether the atomic propositions of interest are true. This can be done by extending the FSM definition with a labeling function $L : P \to 2^{AP}$, where AP stands for the set of all atomic propositions and 2^{AP} stands for the power set of all atomic propositions.

To illustrate the situation, consider the example of the pay machine in figure 4.1. A researcher might be interested to know whether the payment has been accepted, for an item costing 20 cents. We could determine a labeling function trivially: We create the atomic propositions $AP = \{sufficient, insufficient\}$, where all states are labeled with $insufficient$ except for "20 cents or more inserted".

Once we know in which states atomic propositions are true, we need to determine how they relate to one another in the proposition. Computational tree logic uses temporal operators and path quantifiers for this.

Temporal operators are defined over atomic propositions and are true or false on each state, dependent on the underlying graph structure. The operators are $\bigcirc$, $\Diamond$, $\Box$, $\bigcup$, along with the better-known $(true, false, \neg, \cap, \to, \Leftrightarrow)$, which are defined as usual. Given a path, $\bigcirc\phi$ states that ϕ holds after one transition from the current state. Given a path, $\Diamond\phi$ states that from the current state, ϕ will hold some time in the future. Given a path, $\Box\phi$ states that ϕ will hold globally in the future. Given a path, $\phi_1 \bigcup \phi_2$ states that at some point ϕ_2 will hold, and up until that point ϕ_1 will hold.

Temporal operators only express relations given a path. In order to define a proposition over a state one needs path qualifiers. The path qualifier $\exists\phi$ means that there exists at least one path from the current state for which ϕ holds, and $\forall\phi$ means that for all paths starting at the current state ϕ holds. Some examples of these notions can be found in figure 4.2, where $AP = \{black, white, gray\}$.

Some extended formulae expressed in CTL are:

$$\exists\bigcirc(x = 1)$$

meaning that at the current state there is a transition to a state where $x = 1$ is true.

$$\exists\bigcirc(x = 1 \cap \forall\bigcirc(x \geq 3))$$

which denotes that from some specified place, there exists a neighboring state for which $x = 1$ and which has only neighbors with property $x \geq 3$.

The proposition that ϕ_1 and ϕ_2 are mutually exclusive can be denoted by $\forall\Box(\neg\phi_1 \cup \neg\phi_2)$.

The semantics of CTL formulae is defined by two relations (both written $\models$), one for the state formulae and one for the path formulae. For state formulae, $s \models \phi$ iff ϕ holds for place s. Similarly for path formulae, $\pi \models \varphi$ iff path π satisfies φ.

The next step is to find an algorithm that determines for a given FSM whether $FSM \models \phi$. In order to be certain that all operators are well-defined, we add for all terminating states in the given FSM a selfloop.

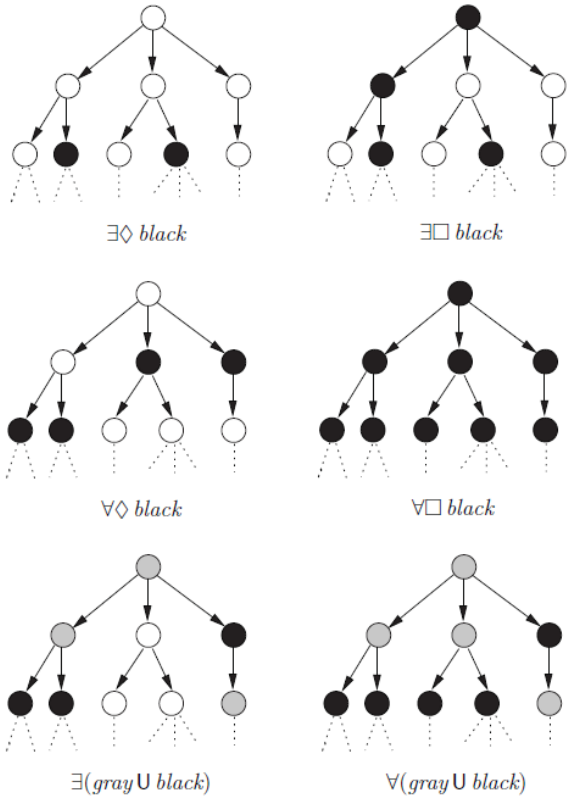


Figure 4.2: Rules of CTL visualized over a tree [BK08].

The basic procedure of model checking is very simple. The algorithm splits up the proposition ϕ into its subformulae, which it splits up again, etcetera. Then, it creates a tree in which the root is ϕ, its branches are its immediate subformulae, and so further in order of the depth of the nesting. The leaves, the loose ends of each branch, are labeled either with *true* or with an atomic proposition. The algorithm then checks for which states the leaves are true. Then, it moves up each branch simultaneously, using general rules of logic to compute for which states the current subformula is true. Eventually we have the set of states for which ϕ is true. If ϕ is true for all possible initial states, the proposition ϕ is verified for our system. If this is not the case and the proposition was supposed to be true in all runs of the model, we have discovered a flaw in our model.

Consider as example the FSM of figure 4.3 and as proposition that needs to be verified: $\phi = \exists\bigcirc a \cap \exists(b \bigcup \exists\Box\neg c)$. Figure 4.3 also specifies what atomic propositions are true for each state. We see in figure 4.4 how we could decompose ϕ, and for what steps these subformulae hold. We first evaluated the most primary subformulae (the atomic propositions at the leaves of this tree) and then moved up using the rules of logical operators. We find that ϕ holds only for P_1 and P_2. This means the verification returns false if either P_3 or P_4 was an initial state, and true otherwise. In the case that P_4 is the only inital state and ϕ is an assertion, then the model is wrong and should be altered.

If the proposition to be checked would have been $\phi = \exists\bigcirc a \cap \exists(b \bigcup \exists\Diamond\neg c)$, where $\Box$ has been changed to $\Diamond$, the model would be verified, no matter what the initial state is.

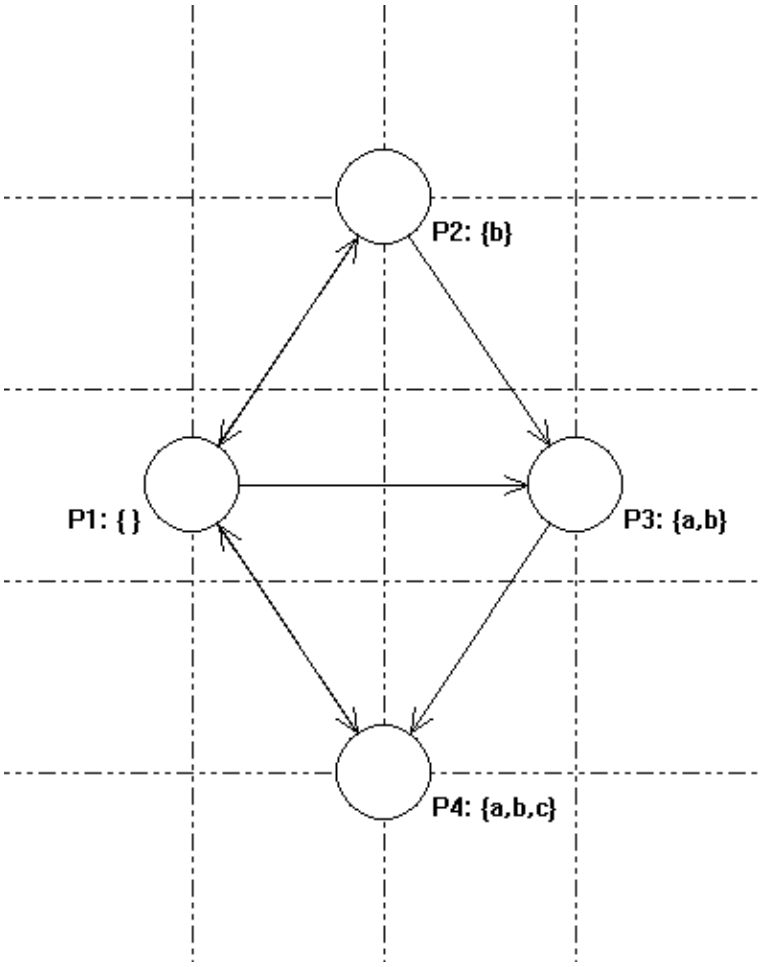


Figure 4.3: A finite state machine with labeling function L defined over $AP = \{a, b, c\}$.

4.4 Analysis Methods for Timed Automata

Timed automata theory has been used extensively over the last few decades to investigate systems that have restrictions based on time. It extends FSM by adding clocks. These clocks increase with rate 1, and can be reset when a transition occurs. They enable the use of guards to specify when a transition can occur, and the use of invariants on states to specify when a transition must occur. In other words, they put constraints on the time between two events. In timed automata, the continuous dynamics for each clock is defined as increasing with constant speed (with the speed of time). In other words,

$$\dot{x_i} = 1$$

for all clocks x_i with $i = 1, 2, ..., m$ where m is the total number of clocks. The transitions are all guard transitions, checking clocks against constants or against other clocks, and allowing the system to change its state once the transition is enabled.

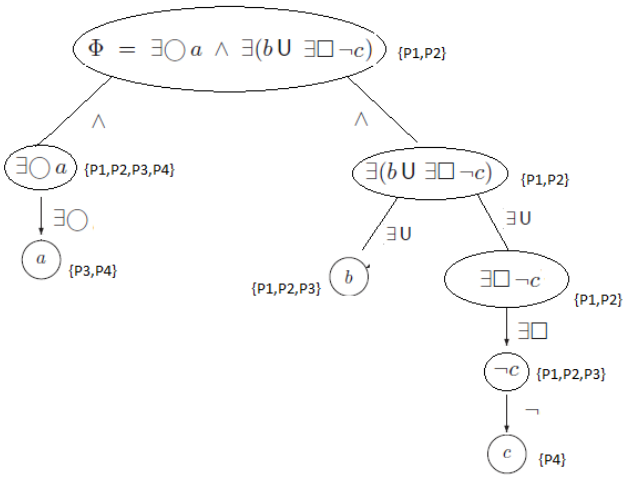


Figure 4.4: The tree decomposition of ϕ.

4.4.1 Description

To formalize, a timed automaton is an octuple $H = \{\mathcal{P}, P_0, \Sigma, E, C, Inv, Grd, Rst\}$ [AD94], where

- $\mathcal{P} = \{P_0, P_1, ..., P_{|P|}\}$ is a finite set of discrete states;
- $P_0 \in \mathcal{P}$ is the initial state;
- Σ is a finite set of labels;
- $E \in \mathcal{P} \times \Sigma \times \mathcal{P}$ is a finite set of edges;
- $C = \{c_1, c_2, ..., c_{|C|}\} \in \mathbb{R}^{|C|}$ is a finite set of clocks. These can be seen as the variables that all states share;
- Inv assigns to each discrete state an invariant. Whenever the system is in a discrete state p, the continuous state $x \in C$ must be in $Inv(p)$. A valuation is a function that, given a discrete state p, assigns to each clock a value within $Inv(p)$;
- Grd assigns to each edge a constraint on the clocks, called a guard;
- Rst defines for each edge which clocks are reset and to what value.

It is important to note that all constants that play a role in the dynamics of a timed automaton, in the guards, invariants and resets, are integers.

Evolution

Initialization: Current state $C_s \leftarrow P_0$
Set all clocks to 0
while *Next timed symbol* $\neq \emptyset$ **do**
  Increase all clocks with time given in timed symbol
  Take the edge labeled with the next symbol
  Reset clocks to 0, if any are given for the edge in *Rst*
end

In the above a timed word is a series of symbols that coincide with the labels in Σ, where each symbol is assigned a time. The time allocated to a sybol describes how long the system remains in the curren location, before it fires a transition corresponding to the symbol. It is assumed that only input words are used that satisfy all guards and invariants. To illustrate the above, a timed word might look like $(0.4a, 2.3b, 0a)$. This word would describe a run where the model stays in P_0 for 0.4 seconds, then moves over an edge labeled with a, stays there for 2.3 seconds, then travels over an edge labeled b, after which it immediately traverses an edge labeled a. That is the end of this run.

4.4.2 Analysis Method 3: Abstraction to FSM for TA

From the original timed automaton we build a region graph such that each region in the graph is a collection of hybrid states from the timed automaton that are equivalent with respect to safety verification [AD94] [BY04]. That is, all the states in a region must be reachable from the same states, and have the same set of states that they can reach. If we can show that we can create a finite region graph, we solve the timed automaton for safety verification, as we could apply FSM analysis techniques on the acquired graph.

Let M be the maximal constant appearing in the clock constraints in the automaton, $M = max\{Grd(e)|e \in E\}$. We now define the notion of reach equivalence for two clock valuations v and v', denoted by $\cong$. Two clock valuations v and v' are reach equivalent if they satisfy for any $x, y \in C$;

- $v(x) > M \Leftrightarrow v'(x) > M$;
- if $v(x) \leq M$, then $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$;
- if $v(x) \leq M$, then $\{v(x)\} = 0 \Leftrightarrow \{v'(x)\} = 0$;
- for any pair of clocks (x, y), if $v(x) \leq M$ and $v(y) \leq M$, then $\{v(x)\} \leq \{v(y)\} \Leftrightarrow \{v'(x)\} \leq \{v'(y)\}$.

where $\lfloor \alpha \rfloor$ denotes the integral part of α, whereas $\{\alpha\}$ denotes the fractional part. An example of a timed automaton and its region graph can be seen in figure 4.5. We see that discrete states have been divided into multiple regions.

The creation of a region graph of a timed automaton can be done automatically. The region graph is a finite state machine, where the states coincide with the regions of the TA and the transitions are between regions that can reach each other due to time elapsing or due to a transition being taken in the timed automaton. The important property of region graphs is that they allow for exactly the same timed words as their corresponding TA. This means that finding the reachable states in the region graph is equivalent to finding the reachable

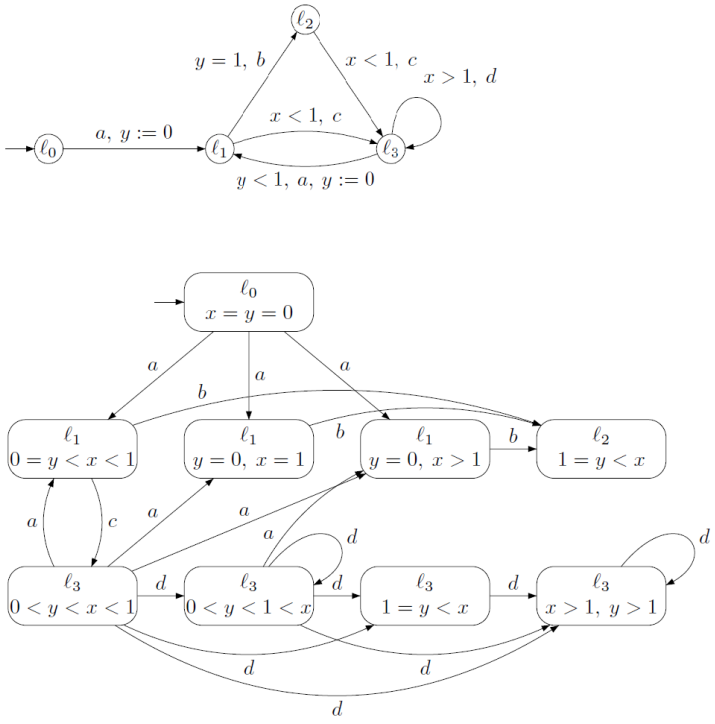


Figure 4.5: Example of a timed automaton and its region graph [AD94].

hybrid valuations in the timed automaton. Therefore, once we have the region graph we can use the analysis techniques of finite state machines to obtain our answer.

4.5 Analysis Methods for Simple Multi-rate Timed Automata

The timed automaton formalism has been developed further to include more diverse phenomena. Although there are many formalisms, often special cases of linear systems, that have been shown to be abstractions of finite state machines, we will look at only one here. Readers interested in proofs that other formalisms can be abstracted to finite state machines are referred to e.g. [LPY99], [Roo12].

4.5.1 Description

Multi-rate timed automata (SMTA), sometimes called rectangular hybrid automata, are timed automata where the clocks do not necessarily all run at the same pace. Instead, each location and each clock is assigned with an interval ρ within which the rate of that clock lies [DY95]. In other words, each location allows for skewed clocks within given limits. For each location p and each clock c, $\rho(p, c)$ is a closed interval whose endpoints belong to $\mathbb{Q}^+ - \{0\}$. We furthermore assume that whenever there is a transition between two states that allow for different intervals ρ of a variable, that variable is reset.

A (multi-rate) timed automaton is called simple if all guard transitions check variables

against constants, not against other variables. So, only transition guards of the form

$$x \leq c_1 \text{ or } x \geq c_1 \text{ or } c_1 \leq x \leq c_2$$

are allowed, where x is a clock and c_1, c_2 denote constants.

4.5.2 Analysis Method 4: Abstraction to TA for SMTA

Simple multi-rate timed automata can be transformed to timed automata. Given an SMTA $H = \{\mathcal{P}, P_0, \Sigma, E, C, Inv, Grd, Rst\}$, we make the following timed automaton H' [DY95]. Choose as set of locations $\mathcal{P}' = \mathcal{P}$, initial location $P_0' = P_0$, label function $\Sigma' = \Sigma$, transitions $E' = E$ and clocks $C' = C$. For each location p and clock c, let $\kappa_p(c, c')$ be the relation $\frac{1}{\beta}x \leq x' \leq \frac{1}{\alpha}$, where $[\alpha, \beta]$ is the interval of rates of clocks for the given location. The guard Grd and invariants Inv are obtained by applying κ to the original guards and invariants. Because the SMTA is simple, each guard is associated with only one clock, and this transformation is well-defined. Once a SMTA has been turned into a TA, it needs to be transformed to an FSM for further reachability analysis.

Example

Figure 4.6 (left) represents a robot that can pick up a package, turn, load it onto another conveyor belt and turn back. The two turning events involve turning 90 degrees, with a rate of somewhere between 15 and 18 degrees per second. The transformed TA is shown on the right.

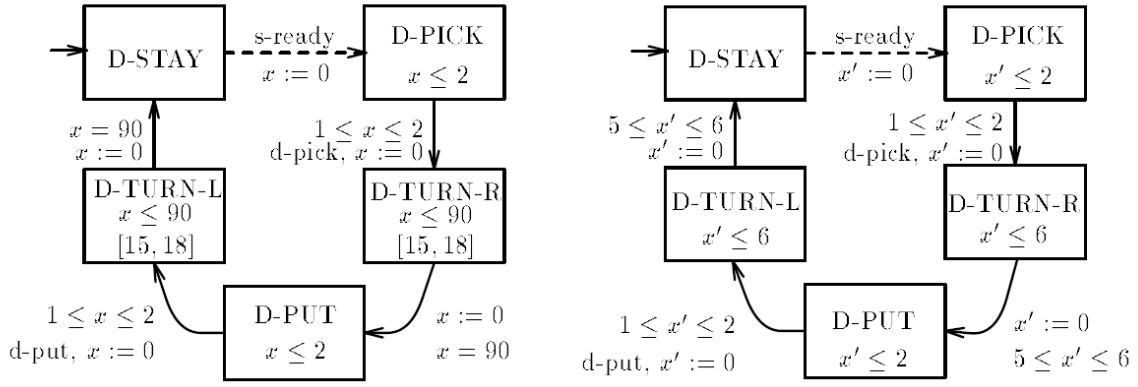


Figure 4.6: Simple multi-rate timed automaton (left) transformed into an equivalent timed automaton (right) [DY95]. Where rates are omitted, $\dot{x} = 1$ is implied.

4.6 Analysis Methods for Continuous-time Markov Chains

Continuous-time Markov chain (CTMC) is a discrete modeling formalism. Models based on CTMC consist of places and edges connecting those places. Edges are labeled with negative exponential parameters, that define the probability for a transition to fire within a given time frame.

4.6.1 Description

A continuous-time Markov chain is a graph $G = (V, E)$, where V is a finite set of nodes, and $E \subseteq V \times V$ is the set of edges, or transitions, between nodes. A function $T : E \to \mathbb{R}^+$ defines the parameters of negative exponential distributions associated with each edge, in such a way that all are greater than zero. As each transition is dependent on a negative exponential distribution, the probability of a transition being taken is independent of the time, and only dependent on the current discrete state. It is called a Markov chain because the dynamics of this system are independent of the past, conditioned on the present.

Evolution

The state of a CTMC is constant for an exponentially distributed time. Once this random time is reached the state will transition to another state. Which transition is taken is determined by the respective exponential parameters. A CTMC-based model can be graphically displayed by circles, arrows and values along those arrows indicating the exponential parameter, see for instance figure 4.7.

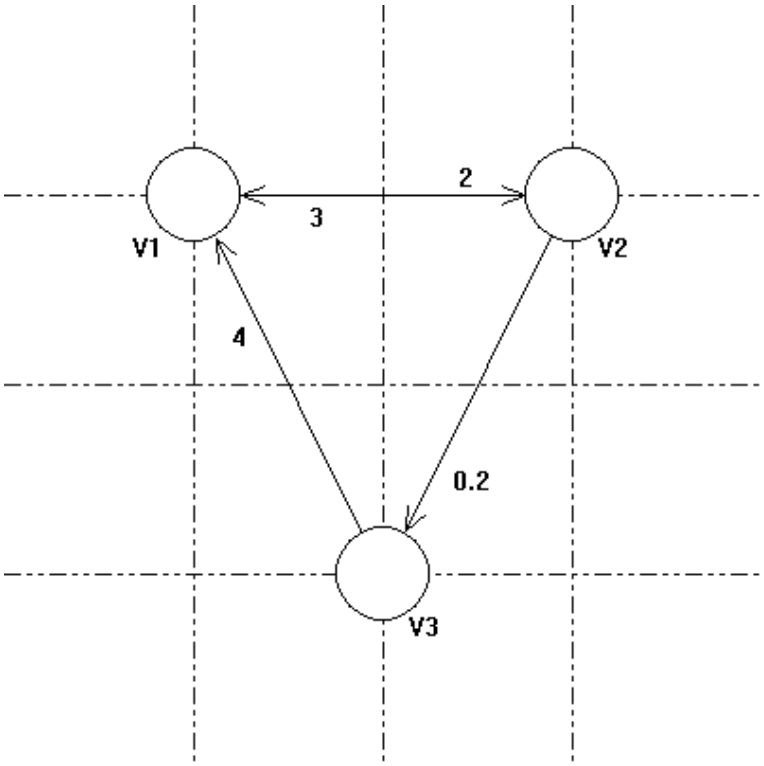


Figure 4.7: An example of a continuous-time Markov chain

4.6.2 Analysis Method 5: Steady State Distribution for CTMC

Continuous-time Markov chain steady state analysis works under the assumption that the model is ergodic. A model is ergodic if any state is reachable from any other state, not necessarily directly. This property guarantees a long-term average probability distribution

over the states that is independent of the initial marking. If a system is not ergodic the long term averages can be zero in some places, because the system of linear equations we try to solve is singular. Also, the long term averages are then dependent on the chosen initial state. Ergodicity is trivially decidable for any not overly large system, by finite state machine analysis. If the system that we are considering is not ergodic, we can decide to investigate subsystems of the model that are ergodic.

In order to find the steady state distribution of a CTMC model we use the following analysis. We first have to check by finite state machine analysis whether the model at hand is ergodic. If it is, we set up its infinitesimal generator, Q. Each diagonal element $Q_{ii}|i \in P$ describes the total flow out of i. Each element not on the diagonal, or $Q_{ij}|i, j \in P, i \neq j$, describes the transition rate from i to j. Since ergodic systems display long term converging behavior, we can then solve the system of equations

$$\pi Q = 0$$

where π denotes the steady-state distribution. Basically, this works via the principle that in the long run, the event that a system enters a state happens equally often as the event that the system leaves the state. Together with the normalization condition $\sum_i \pi_i = 1$ we have a set of equations that provide a unique solution. The ergodicity of the model guarantees non-singularity of the set of equations.

To clarify by example, assume that a system of interest has three places and four transitions, $E = \{(1,2),(2,1),(3,1),(2,3)\}$, with transition rates: $T_{1,2} = 2, T_{2,1} = 3, T_{3,1} = 4, T_{2,3} = 0.2$, see figure 4.7. The resulting infinitesimal generator is

$$Q = \begin{pmatrix} -2 & 2 & 0 \\ 3 & -3.2 & 0.2 \\ 4 & 0 & -4 \end{pmatrix}$$

Solving $\pi Q = 0$ together with $(\pi_1, \pi_2, \pi_3)(1,1,1)' = 1$ gives us, by using MATLAB's linsolve: $(\pi_1, \pi_2, \pi_3) = (0.6038, 0.3774, 0.0189)$. This steady state distribution tells us that, on average, state 1 is current about 60 percent of the time, state 2 is current 38 percent of the time, and state 3 is current only 2 percent of the time.

4.7 Analysis Methods for Discrete-time Markov Chains

For the sake of completeness we also present the steady state analysis of CTMC altered to discrete-time Markov chains (DTMC). This is included as we will look at methods of reducing stochastic formalisms to discrete time Markov chains.

4.7.1 Description

A discrete-time Markov chain is a set of locations, a set of edges between those locations, and for each location a distribution that determines, given the current state at time t, the distribution for the system at time $t+1$. Note that it is usual here to include edges from i to i for place i, to encode the chance that a state is attained for longer than one time unit. One of the locations is the initial location.

Evolution

The run starts at the initial location. Then, at each time t, the distribution determines the next location (at time $t+1$) given the current location.

4.7.2 Analysis Method 6: Steady State Distribution for DTMC

A discrete-time Markov chain has a stationary distribution if it is irriducible, i.e. any state is reachable from any other state. This notion is similar to ergodicity for continuous-time Markov chains. The infinitesimal generator is replaced by a probability transition matrix Q where $Q_{i,j} = Pr(p_k = j | p_{k-1} = i)$ with p_k denoting the state that is current at time k. The steady state behavior is found by solving the system of linear equations:

$$\pi = \pi Q$$

together with the normalizing equation $\pi_1 + \pi_2 + ... + \pi_{|P|} = 1$, where $|P|$ is the number of states. For example, we could convert figure 4.7 to the DTMC in figure 4.8 with timesteps 0.01. Using MATLAB, we find that the steady state solution is exactly the same as the steady state solution of the CTMC of 4.7.

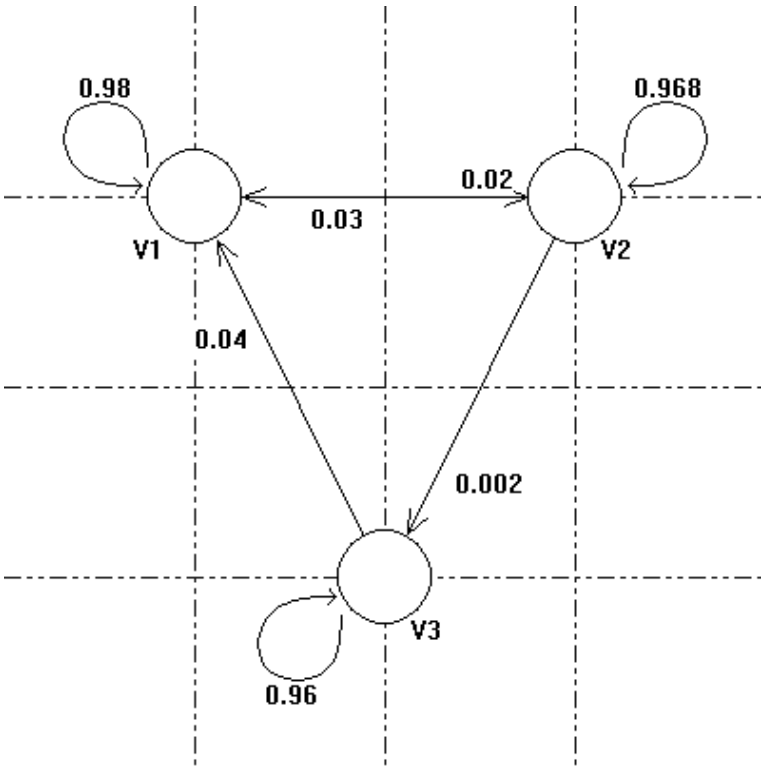


Figure 4.8: An example of a discrete-time Markov chain

4.7.3 Analysis method 7: Probabilistic Reachability for DTMC

Given a DTMC, it is easy to determine how likely it is for a set of unsafe states to be reached within a time limit. We start at the initial state with probability 1. In each time step,

the probability is propagated through the system. For example, if the initial state has two neighbors, which it can reach in one time step with probability 0.2, then the next step would have the initial state at 0.6, and both of the neighbors at 0.2. This propagation is continued until the time limit is reached. It is important that the unsafe states are all absorbing. The computation then becomes

$$\pi_f = \pi_0 Q^t \tag{4.1}$$

where π_f are the probabilities of each state occurring, π_0 is the initial state, Q is the probability transition matrix and t denotes the number of steps one would like to look at. It is important that each target state P_i is a terminal state (i.e $Q(i,i) = 1$). After the computations have been done a researcher has only got to add up the probabilities of each unsafe state to get the likelihood that the unsafe set will be reached within the time limit. This time limit can be made arbitrarily large to mimic the non-bounded case.

4.8 Analysis Methods for Hybrid Automata

As discussed above, there are many special cases of hybrid systems investigated in the literature, each with their own reason to be of mathematical interest. Many have been invented to be able to express a specific range of phenomena, and their modeling power is limited to that range. The entire branch of hybrid automata has a lot of modeling power, as it is an expansion to automata that allow the modeling of continuous events. It thus captures the interaction between a discrete machine and its continuous environment. Hybrid automata are defined as follows.

4.8.1 Description

A hybrid automaton H is a nonuple $\{L, E, X, Init, Inv, Flow, Jump, \Sigma, Upd\}$, consisting of the following elements [MS96];

- $L = \{l_1, l_2, ..., l_n\}$ is a finite set of discrete states;
- $E \subseteq L \times L$ is a finite set of edges between distinct states. Together, (L, E) form a directed simple graph;
- A finite set $X = \{x_1, x_2, ..., x_n\} \in \mathbb{R}^n$, the variables in the various states. A valuation v is equal to the hybrid state of the system at a certain time, or $v(t) = (l(t), x(t))$;
- $Init$ describes for each valuation from (L, X) whether it is a possible initial valuation or not;
- Inv describes for each state from L the possible range of X, the continuous variables;
- $Flow$ describes for each state from L the dynamics on X through differential equations;
- $Jump$ defines for each element in E the necessary conditions on X for that edge to be enabled;
- A finite set Σ that labels the edges from E;
- Upd describes how X is updated for each element from E.

Evolution

A hybrid automaton does not describe how runs occur, only which runs are possible and which are not. Any run must follow the following rules. A run starts at some location L_0 specified by $Init$. At any location its variables $x \in X$ vary according to the $Flow$. If this $Flow$ leads to $x \notin Inv$ a forced jump occurs according to $Jump$ and E. At any time that x does not leave the invariant and some edge in E, starting at the current place, is enabled according to $Jump$, that edge can be taken. If a jump occurs Upd specifies x after the jump, by taking into account the edge that is jumped over and the variable x prior to jumping.

A run of a hybrid system can then be characterized as follows:

$$\pi : \sigma_0 \mapsto^{t_0}_{f_0} \sigma_1 \mapsto^{t_1}_{f_1} \sigma_2 \mapsto^{t_2}_{f_2} ...$$

where $\sigma_i = (l_i, v_i) \in (L, X)$, $t_i \in \mathbb{R}^+$, and $f_i \in \Sigma$. The state σ_{i+1} is a successor of σ_i. The run starts at σ_0, then fires the transition labeled with f_0 at time t_0. Until that time the flow is followed to evolve the variables of X. This continues until no further actions are specified by the run. Therefore the run is:

Initialization: Current state σ_0
while *Next timed symbol* $f_i \neq \emptyset$ **do**
  let the variables in X follow the differential equation as specified by $Flow$ for t_i amount of time
  Take the edge labeled with the next symbol f_i
  Update the variables of X according to Upd
end

4.8.2 Analysis Method 8: RRT-based Falsification for HA

Falsification tries to find witness trajectories from the initial states to the target state. If this approach succeeds in finding a trajectory, it proves that the system is unsafe. If it can find no trajectories, this can be seen as evidence that the system is safe, but it does not constitute a proof.

Unlike most methods, where algorithms search in an exhaustive way for counterexamples, falsification uses some route planning mechanism to find specific runs in the system. The route planning method we will look at is rapidly-exploring random trees (RRT). The key idea is to incrementally grow a tree from the initial to the target state[Lav98]. At each iteration a new vertex is created. We try to bias the growth to unexplored areas. The general version of this model checking method is this:

Initialization: Tree of one node, the initial state
while *Target state not reached and computation limit not exceeded* **do**
  Randomly draw a valid state
  Find the closest node in the tree
  Create a new node in the tree δ distance from the closest node in the direction of the random state picked, and connect it to the closest node
end

RRT was initially used for robot motion planning. In that situation reasonable limits can be set for the size of the total space of the problem, so it becomes possible to draw a random

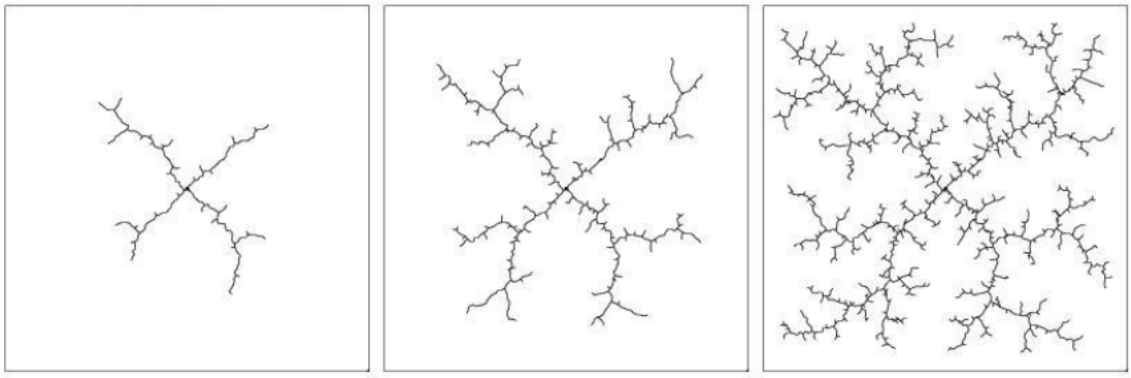


Figure 4.9: Example of a rapidly exploring random tree in a 2 dimensional non-obstructed statespace. The figures show that the search is biased towards unexplored areas [Lav98].

valid state. Also, the Euclidean metric can be used to define the closest neighbor, and in what direction the new vertex should be. If we deal with unconstrained movement, i. e. the robot can travel in all directions, we simply move in the direction of the random state that was chosen. RRT then biases exploration to unexplored areas, as vertices close to empty space have more chance of being the closest neighbor to a random point than are vertices with many close neighbors. This insures that RRT is a fast method of finding a valid route.

RRT implementation is more complicated in hybrid systems. What is the state space from which we can pick a random point? How do we define the distance between two points if they are in different discrete states? How can we move in a certain direction, if we are limited in our movement by differential equations? [PKV09] provides a strategy to deal with these matters, depicted in figure 4.10. The paper proposes a multilayered approach called HyDICE, Hybrid DIscrete Continuous Exploration. First, the algorithm determines a set of transitions that leads from the initial to the target state. This is the macro-path. For each discrete state that this global path encounters, we have an initial state and a target state. The approach then uses RRT to connect these points. This means that we initiate with the continuous state where the discrete state has been entered. In each iteration, we draw a random continuous state from within the statespace of the location. If the statespace of the discrete state is infinite, we limit our scope to a finite region which is sufficiently large to include all reasonably interesting paths. The closest node in the tree is determined by Euclidean distance. Now we should determine which direction we should move from the closest node. However, as hybrid systems have deterministic continuous dynamics, there is only one possiblity. From the closest node we follow the flow for δ distance. If a 'maximum time' has been reached without the algorithm having found a path, the algorithm concludes that there is no path that connects the transitions proposed before, so it updates the macro-path to a new path that does not need the same transitions in succession, see figure 4.10. If applicable, the found tree can be used as initialization of the new RRT search. If during some update no new path can be found, or when an overall time limit has been reached, the algorithm can conclude that it was unable to find a path. This is no proof that the target state is unreachable, but it is an indication that that is true.

It should be noted that when one works with controlled hybrid systems, the above can be done in a more effective way. Controlled hybrid systems provide the user with a limited input, that he can use to steer the current state towards a desired direction. In that case,

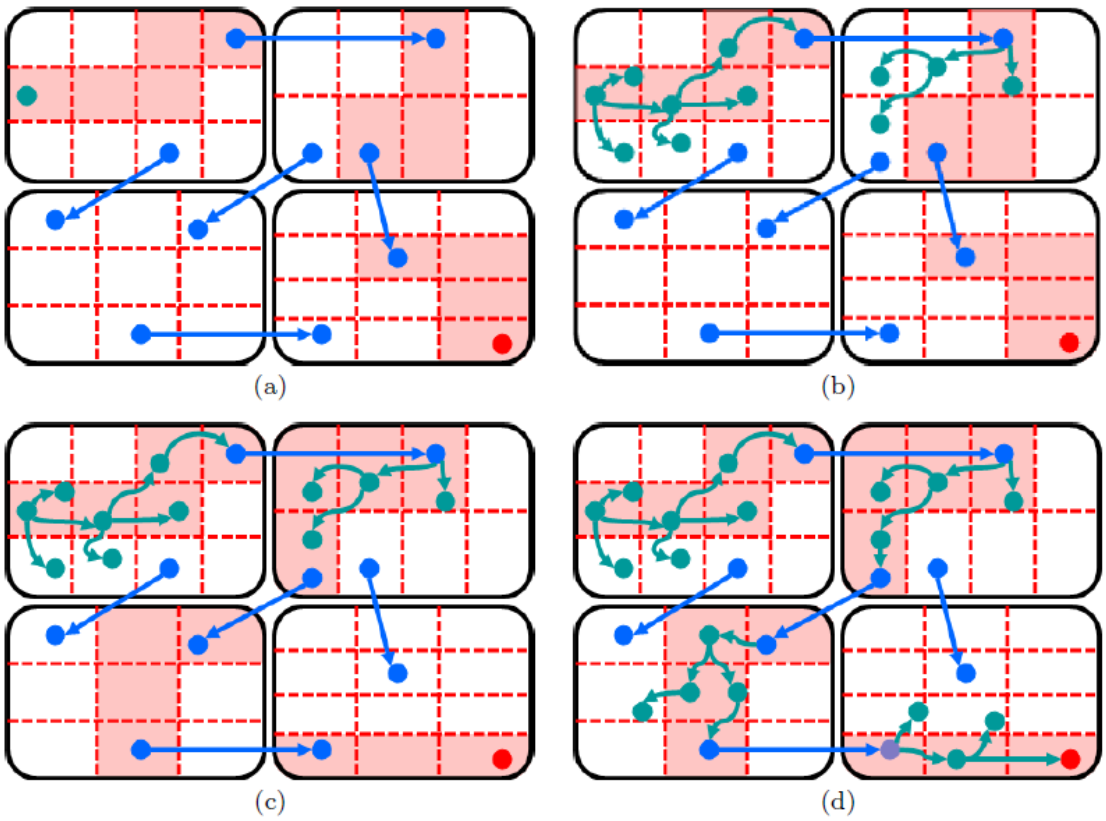


Figure 4.10: Example of the HyDICE approach from [PKV09]. This is a controlled hybrid system, as there are multiple paths possible from a given hybrid state. In (a) a global path is calculated. In (b) no path is found to connect transitions in the upper right corner place. In (c) the global path is recalculated. In (d) a witness trajectory is found.

the direction from the closest state does not necessarily have to be the flow, but can be any direction available when applying the input on the flow for δ distance. Naturally, one can choose the control in such a way that the direction brings us closest to the randomly chosen point.

It should furthermore be noted that this algorithm can also be performed like a backward reachability analysis, starting from the goal state and moving towards the initial set. Even better, if we perform each individual RRT search as both forward as well as backward search, the algorithm becomes more efficient. Moreover, generally the starting node is actually a collection of nodes. The initialization of the system and the reset maps over transitions are not deterministic, which means that we start with a range of possible starting points.

Summarizing, the algorithm would look like:

Input: Hybrid system, t_{global}: upper bound on computational time for the global algorithm, t_{indv}: upper bound on computational time for the individual searches
Output: Witness trajectory or Failure
Initialize: Start clock $t = 0$
Initialize: Establish a global path
while *no witness trajectory and* $t < t_{global}$ **do**
  for *each place visited by global path* **do**
    Perform RRT search on entering state to goal state within that place:
    Initialize: create tree with set of initial nodes, the states where the place can be entered
    while *no valid path found and* $t < t_{indv}$ **do**
      Select a random state from the state space of the place
      Determine the closest node of the tree to the picked random state
      From this closest node, follow the flow for distance δ (if there is diffusion, use this to get closer to the picked state)
      Add this state to the nodes of the tree
      Add an edge from the new node to the closest node
    end
    if *one of the RRT searches comes up empty* **then**
      Update global path
    end
  end
end
Return: failure or witness trajectory.

4.9 Analysis Methods for Linear Hybrid Automata

4.9.1 Description

A linear hybrid automaton is a hybrid automaton as described in 4.8, where the $Flow$, Inv and $Jump$ are rational linear mappings over the relevant terms.

Evolution

A linear hybrid automaton evolves exactly as a hybrid automaton does.

Preliminaries

Before we look at the different ways to analyze linear systems, we define some vital notions. Let $P \subseteq (l_i, K),\ l_i \in L$ and $K \subseteq inv(l_i)$, be a set of acceptable hybrid states in some location. The set of states that can be reached from this set P through the flow is called the forward time closure of P, denoted by $\langle P \rangle_l^{\nearrow}$. Similarly the backward time closure of a set of states P in one place is defined as the set of states for which there exists a run that ends in some $v \in P$. It is denoted by $\langle P \rangle_l^{\swarrow}$.

A region $P = (M, K)$, where $M \subseteq L$ and $K \subseteq X$, is a subset of all possible hybrid states. As can be seen from the definition it can span multiple discrete states, and need not contain all possible valuations from such a state. The forward respectively backward time closure of a region are defined similarly to those of a hybrid state.

A postcondition, given $e \in E$ and $P \subseteq (L_i, X),\ L_i \in L$, describes what states can be reached if the system is in P and transition e fires. The postcondition of e on P is written $post_e[P]$. Thus, $x' \in post_e[P]$ iff $\exists x \in P$ s.t. $(l, x) \to (l', x')$. Analogously, the precondition defined on $e \in E$ and $P \subseteq X$ describes the set of hybrid states from which it is possible to reach P by transitioning through e. It is denoted by $pre_e[P]$. A postcondition or precondition can also be defined only on a region, not on a specific edge, in which case we do not specify which edge should be taken, but instead accept hybrid states from any transition that can reach or be reached from the region.

4.9.2 Analysis Method 9: Forward Analysis for LHA

In terms of linear hybrid automata, the question concerning safety verification asks, "is it possible to reach from state $\sigma \in (L, X)$ another state σ'?" It is natural then to investigate what the range of states is that simulation runs starting at σ can enter. This is called forward analysis as we travel forward in time. The basic idea is very similar to safety verification for finite systems, and is described in [ACH$^+$95].

One determines analytically in each step what range of hybrid states can be reached from the set of current states, starting at the initial state. Each of those reached states forms the set of current states of the next step. Once the algorithm reaches a fixpoint, i.e. the range of states we compute is equal to the former, we know the reachable scope. We see that this is very similar to the analysis of finite systems. We still update a current set of states in iterations by expanding the current set by states that can be reached from the current set. The difference, however, is that the reachability problem for linear hybrid automata is undecidable in general, so this algorithm will not necessarily converge. We call an algorithm with these characteristics a semi-decision algorithm. As a general algorithm it would look like this:

Initialization: ReachedSet(0) := initial set
$i = 0$
while *not stopping criteria* **do**
  $i = i + 1$
  ReachedSet(i) = ReachedSet(i-1) expanded to all valuations it can reach within its places, and expanded to all valuations it can reach via edges.
end

The idea comes from the notion of a symbolic run of a system. A symbolic run is a finite sequence

$$g : (l_0, P_0)(l_1, P_1)...(l_i, P_i)...$$

of regions such that for all $i \geq 0$ there exists a transition e_i from l_i to l_{i+1} and $P_{i+1} = post_{e_i}[\langle P_i \rangle^{\nearrow}_{l_i}]$.

The reachable region for an initial region $I \subseteq (L, X)$, written $(I \mapsto^*)$, is the smallest subset of the state space that includes the initial region itself and each state that can be reached through a combination of transitions and continuous evolution. That reachable region is the least fixpoint of the equation

$$X_I = \langle I \cup post[X_I] \rangle^{\nearrow}.$$

For automated computing this method presents specific problems. It might cost an exorbitant amount of time and memory, as the expressions for each step in the fixpoint calculation become increasingly complex. However, if an answer is found the technique is

exact, and the accuracy guarantee is 100 percent. This means that if this technique gives a yes or no answer, that is the correct answer.

Example

We will show how forward analysis works for the system of Figure 4.11. A gas burner can leak, but if it does so it is detected and resolved within one second, after which it takes at least 30 seconds before it could start leaking again. In this example, x is used to record the time taken for the leak to be detected and resolved, y records the total amount of time the system has been simulated, and z records how much of that time has been spent leaking. Inside the circles are written the flows and invariants. On the transitions are written the guards and resets. We take as initial setup $\phi_I : pc = 1 \cap x = y = z = 0$. The set $(I \mapsto^*)$ of reachable states from I is the fixpoint solution of these two equations:

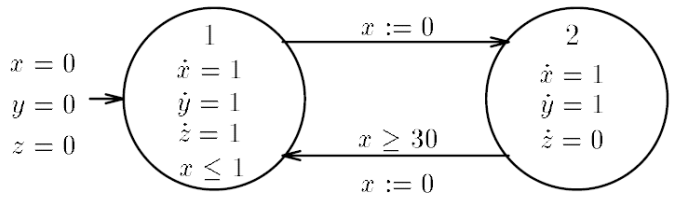


Figure 4.11: An example of a linear hybrid system [ACH+95].

$$\begin{aligned} \phi_1 &= \langle x = y = z = 0 \cup post_{(2,1)}[\phi_2] \rangle_1^{\nearrow} \\ \phi_2 &= \langle false \cup post_{(1,2)}[\phi_1] \rangle_2^{\nearrow} \end{aligned}$$

which becomes an iterative process:

$$\begin{aligned} \phi_{1,0} &= \langle x = y = z = 0 \rangle_1^{\nearrow} = (x \leq 1 \cap y = x = z) \\ \phi_{2,0} &= false \\ \phi_{1,i} &= \phi_{1,i-1} \cup \langle post_{(2,1)}[\phi_{2,i-1}] \rangle_1^{\nearrow} \\ \phi_{2,i} &= \phi_{2,i-1} \cup \langle post_{(1,2)}[\phi_{1,i-1}] \rangle_2^{\nearrow} \end{aligned}$$

For $i = 1$ we have

$$\begin{aligned} \phi_{1,1} &= \phi_{1,0} \cup \langle post_{(2,1)}[\phi_{2,0}] \rangle_1^{\nearrow} = \phi_{1,0} \\ \phi_{2,1} &= \phi_{2,0} \cup \langle post_{(1,2)}[\phi_{1,1}] \rangle_2^{\nearrow} \\ &= \langle post_{(1,2)}[x \leq 1 \cap y = x = z] \rangle_2^{\nearrow} \\ &= \langle (x = 0 \cap y \leq 1 \cap z = y) \rangle_2^{\nearrow} \\ &= (z \leq 1 \cap y = z + x) \end{aligned}$$

We get from induction on i that for all $i \geq 2$

$$\begin{aligned} \phi_{1,i} &= \phi_{1,i-1} \cup x \leq 1 \cap 0 \leq z - x \leq i \cap 30i + z \leq y \\ \phi_{2,i} &= \phi_{2,i-1} \cup y \leq i + 1 \cap 30i + x + z \leq y \end{aligned}$$

So the least fixpoint is

$$\phi_R = (pc = 1 \cap \phi_1) \cup (pc = 2 \cap \phi_2)$$

where

$$\begin{aligned} \phi_1 &= x \leq 1 \cap x = y = z \cup \exists i \geq 1 s.t. (x \leq 1 \cap 0 \leq z - x \leq i \cap 30i + z \leq y) \\ &= (x \leq 1 \cap x = y = z) \cup (x \leq 1 \cap x \leq z \cap y + 30x \geq 31z) \\ \phi_2 &= z \leq 1 \cap y = x + z \cap x \geq 0 \cup \exists i \geq 1 s.t. (z \leq i + 1 \cap 30i + x + z \leq y) \\ &= (z \leq 1 \cap y = x + z \cap x \geq 0) \cup y \geq x + 31z - 30 \end{aligned}$$

4.9.3 Analysis Method 10: Backward Analysis for LHA

This technique works in a analogous way to the forward reachability analysis. The difference is that here we start with the risk zones, and successively move backwards in time.

Given a target region $I \subseteq (L, X)$, the initial region is the least fixpoint of the equation

$$X_I = \langle I \cup pre[X_I] \rangle^{\swarrow}$$

The reachability of the leaking gasburner, upon which we tested the forward analysis technique, can also be examined using backward analysis. This can be found here [ACH+95].

4.9.4 Analysis Method 11: Approximate Analysis for LHA

There are two difficulties in the computation of forward and backward analysis. The first is that it might run forever, as it is only a semi-decision procedure, because there are an infinite number of possible states. There is no guarantee of finding a fixpoint, because every iteration might find new states to add to the current solution. The second is that in each iteration we obtain multiple polyhedra, or sets bounded in each variable by a linear constraint, one from each transition we can take in that iteration. Many of these polyhedra intersect or are equivalent. However, there is no easy method for determining whether a union of polyhedra is included in another union of polyhedra. Thus the number of polyhedra keeps rising, and a computational program would have to keep track of it all. Therefore, the forward or backward reachability analysis are generally not used directly. Instead, an approximate solution to these problems is provided by abstract interpolation techniques [ACH+95]. The sets propagated through the dynamics of the system are first abstracted to other sets, which make the computation easier, and then propagated through the system. We introduce two operators to create a fully automated procedure that will always terminate and overcome the polyhedron computational difficulty.

In order to deal with the polyhedron problem we approximate polyhedra by their convex hull. A convex hull is the least convex polyhedron that contains the operands of the union. We denote this operation by $\bigcup$:

$$P \bigcup P' = \{\lambda x + (1-\lambda)x' | x \in P, x' \in P', \lambda \in [0,1]\} \tag{4.2}$$

Therefore, when we iterate the forward analysis step, using the convex hull approximation, the equation will become:

$$X_I = \left\langle I \bigcup post[X_I] \right\rangle^{\nearrow} \tag{4.3}$$

At each iteration, we take the convex hull of all polyhedra within the same place, and in this way approximate what the reachable set is within that place. The upside of this is that we have only one set per place, instead of possibly any number, so this costs much less computational time.

The next problem to be fixed is that of non-convergence. This is solved by looking at a widening technique (∇). This technique must satisfy:

- $P \bigcup P' \subseteq P \nabla P'$
- For each infinite series of polyhedra $(P_0, P_1, ..., P_n, ...)$, the series defined by $Q_0 = P_0$, $Q_{n+1} = Q_n \nabla P_{n+1}$ must not be strictly increasing. It must remain constant after a while.

This can be done by consecutively removing constraints from P. P, the convex polyhedron, is defined by constraints on the variables of the state-space of a place, so removing constraints consistently makes the polyhedron larger. As only finitely many constraints can be removed, the operation must remain constant after finite iterations. The convex hull approximation operator and the widening operator can be seen on two examples in figure 4.12.

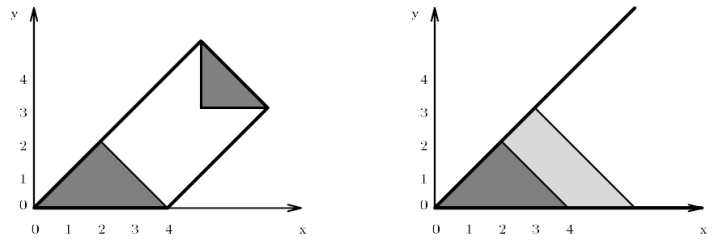


Figure 4.12: The convex hull approximation operator and the widening operator [ACH+95].

This analysis technique is an over-approximation. That means that if the algorithm generates a yes answer, this was not necessarily the case in the original system. However, if the algorithm generates a no answer for a given initial and target state, then within the original system the target state was not reachable from the initial state. The algorithm operates on the 'safe side'.

In order to present a complete overview of available literature, we mention that ellipsoidal approximation techniques can also be used in place of convex hull approximations. One

creates an outer ellipsoid or an inner ellipsoid for the current set, and propagates this through the system. These ellipsoidal approximations can be made in such a way that they are overapproximations. We refer the interested reader to [KV00].

4.9.5 Analysis Method 12: Minimization for LHA

The approximation techniques for forward or backward reachability analysis can still include a large number of difficult computations. Minimization can aid in this respect. The principle behind this technique is to define the forward (respectively backward) time closure on two regions, instead of between a region and a specific state. We define $R \mapsto R'$ if there exists a state $\sigma' \in R'$ that can be reached from some state $\sigma \in R$.

Let π be a partition of the entire state space. Then $R \in \pi$ is stable if for all $R' \in \pi$:

$$R \mapsto R' \text{ implies } \forall \sigma \in R : \{\sigma\} \mapsto R' \tag{4.4}$$

Further, a partition π is a bisimulation if every region in π is stable. A partition π respects some region R_F if for every region $R \in \pi$, either $R \subseteq R_F$ or $R \cap R_F = \emptyset$. If a partition is a bisimulation and respects a region R_F then it can be used to compute reachability. Thus, our objective is to find the coarsest (partition with fewest sets) bisimulation that respects R_F, given that such partition exist.

In cases where we have an initial and a final region, we would like to create an algorithm that performs a simultaneous reachability and minimization analysis. This can be done in the following way. Let R_F be the set of risky states, I be the set of initial states and denote by Λ the entire state space, then

$\pi := \{R_F, \Lambda - R_F\}; \alpha := \{R \mid R \cap I \neq \emptyset\}; \beta := \emptyset$
while $\alpha \neq \beta$ **do**
  choose $R \in (\alpha - \beta)$
  let $\alpha' := split[\pi](R)$
  if $\alpha' = R$ **then**
    $\beta := \beta \cup \{R\}$
    $\alpha := \alpha \cup \{R' \in \pi | R \mapsto R'\}$
  else
    $\alpha := \alpha - \{R\}$
    if $\exists R' \in \alpha'$ *such that* $R' \cap I \neq \emptyset$ **then**
      $\alpha := \alpha \cup \{R'\}$
    end
    $\beta := \beta - \{R' \in \pi | R' \mapsto R\}$
    $\pi := (\pi - \{R\}) \cup \alpha'$
  end
end
return There is $R \in \alpha$ such that $r \subseteq R_F$

Here $split[\pi](R)$ is a function that splits the region $R \in \pi$ into subsets that are more stable with respect to π:

$$split[\pi](R) := \{R', R - R'\} \text{ if } \exists R'' \in \pi, R' - pre[\langle R'' \rangle^{\swarrow}] \cap R \cap R' \subseteq R$$

In the case that the if condition cannot be fulfilled, $split[\pi](R)$ returns R, the input.

This algorithm terminates if there exists a partition π that respects the final region R_F, is stable and is finite. If this partition exists the algorithm provides the set α which includes all sets of states R that are reachable from the initial state.

4.10 Analysis Methods for Switching Diffusion Processes

A switching diffusion process (SDP) is a hybrid system where the continuous dynamics are governed by SDEs dependent on the discrete state, whereas the discrete evolution is a Markov process where the transition rates are dependent on the continuous state. When a discrete transition occurs, the continuous state is not changed.

4.10.1 Description

A switching diffusion process is a tuple $\{\mathcal{X}, L, \mu_0, f_l, g_l, \lambda_W\}$[PJP07], such that

- $\mathcal{X} \subset \mathbb{R}^n$ is the bounded continuous state space;
- L is the finite set of locations;
- μ_0 is the distribution that determines the initial (hybrid) state;
- $f_l : \mathcal{X} \to \mathbb{R}^n, l \in L$ is a set of vector fields;
- $g_l : \mathcal{X} \to \mathbb{R}^{n\times m}, l \in L$ is a set of diffusion coefficients, where the ith column of g_l corresponds to the ith component of the $\mathbb{R}^m$-valued Wiener process $w(T)$;
- $\lambda_W : \mathcal{X} \to \mathbb{R}, (l, l') \in L^2$ determines the transition rates.

Evolution

The run of an SDP starts by drawing an initial location by μ_0. The continuous dynamics are given by

$$dx(t) = f_l(x(t))dt + g_l(x(t))dw(t)$$

where l is the current location. The discrete dynamics are determined by λ_W.

Whenever the system is in location l, the system evolves according to the flow f_l. A discrete transition occurs according to λ, which is dependent on both l and the continuous state x. If a transition occurs, the discrete state is changed, whereas the continuous state remains unchanged.

4.10.2 Analysis Method 13: Barrier Certificates for SDP

In many reachability analysis methods one tries to propagate a set through the dynamics of the system in order to find what states are reachable. Specifically, one wants to find whether it is possible to go from an initial set to a target set. There is however a different way of looking at this problem. Instead of propagating sets through the dynamics one can also try to find some barrier with a zero level set through which the initial set could not propagate, and which blocks the initial set from access to the target set. A barrier containing such a zero level set is called a certificate as it certifies that the system is safe [PJP07].

In an n-dimensional continuous dynamical system ($\mathcal{X} \in \mathbb{R}^n$) a zero level set of a barrier certificate can be seen as an $(n-1)$-dimensional plane, that separates the initial ($\mathcal{X}_0$) and target ($\mathcal{X}_u$) region. In an SDP, each location must have a barrier set. Let $\mathcal{D}$ be the set of disturbance inputs. The barrier (B) must have the following charactersistics $\forall l \in L$:

- $B_l(x) \leq 0 \ \forall x \in \mathcal{X}_0$;

- $B_l(x) > 0 \; \forall x \in \mathcal{X}_u$;
- $\frac{\mathrm{d}B_l}{\mathrm{d}x}(x) f_l(x,d) \leq 0 \; \forall (x,d) \in \mathcal{X} \times \mathcal{D}$ such that $B(x) = 0$.

If there exists a level set with the above characteristics, it is clear that no path exists from $\mathcal{X}_0$ to $\mathcal{X}_u$. The reason is that starting at the initial set the function is negative, and will never be able to cross to a positive value, whichever path it chooses. It can therefore never end up in the target region, which is positive. The zero-level set forms a boundary on what can be reached from the initial set. No trajectory starting from the initial set can cross this boundary.

The computation of a barrier certificate is no trivial matter. For general processes there is no standardized algorithm that can always find a certificate, if one exists. It is possible if one restricts to a system for which:

- Vector fields are polynomial;
- The initial and target sets can be described by polynomial inequalities;
- The barrier certificate itself is polynomial.

If the above is the case the problem can be transformed to a sum of squares problem. A MatLab toolbox such as SOSTOOLS can be used to solve such problems [PPP02]. If this toolbox cannot find a barrier certificate from within the range of candidates this does not prove that the system is safe, as there might exist non-polynomial barrier certificates.

Example

Assume that we have a hybrid system in which one location has two dimensional dynamics: $\dot{x}_1 = x_2$ and $\dot{x}_2 = -x_1 + \frac{p}{3}x_1^3 - x_2$, where p is an uncertain time-invariant parameter that lies in $[0.9, 1.1]$. The location can be entered with values $\{x \in \mathbb{R}^2 : (x_1 - 1.5)^2 + x_2^2 \leq 0.25\}$ and a transition can occur from $\{x \in \mathbb{R}^2 : (x_1 + 1)^2 + (x_2 + 1)^2 \leq 0.16\}$. All this information is presented in figure 4.13. A barrier certificate can be found, represented by the dashed curves in figure 4.13, by methods presented in [PJ04]. This means that the global path from the initial to the target state cannot use this location. If barrier certificates can be found for enough locations, such that there is no global path from the initial set to the target set, the target set cannot be reached. This would certify that the system is safe.

4.11 Analysis Methods for General Stochastic Hybrid Systems

General stochastic hybrid systems allow for a very broad range of dynamics. It includes diffusion processes in the continuous dynamics. It contains both spontaneous and forced transitions, and the reset after a transition can be probabilistic.

4.11.1 Description

A general stochastic hybrid system is an octuple $H = \{\mathcal{Q}, D, \mathcal{X}, b, \sigma, Init, \lambda, R\}$ [BL04], [BL06] where

- $\mathcal{Q}$ is a countable set representing the discrete states;

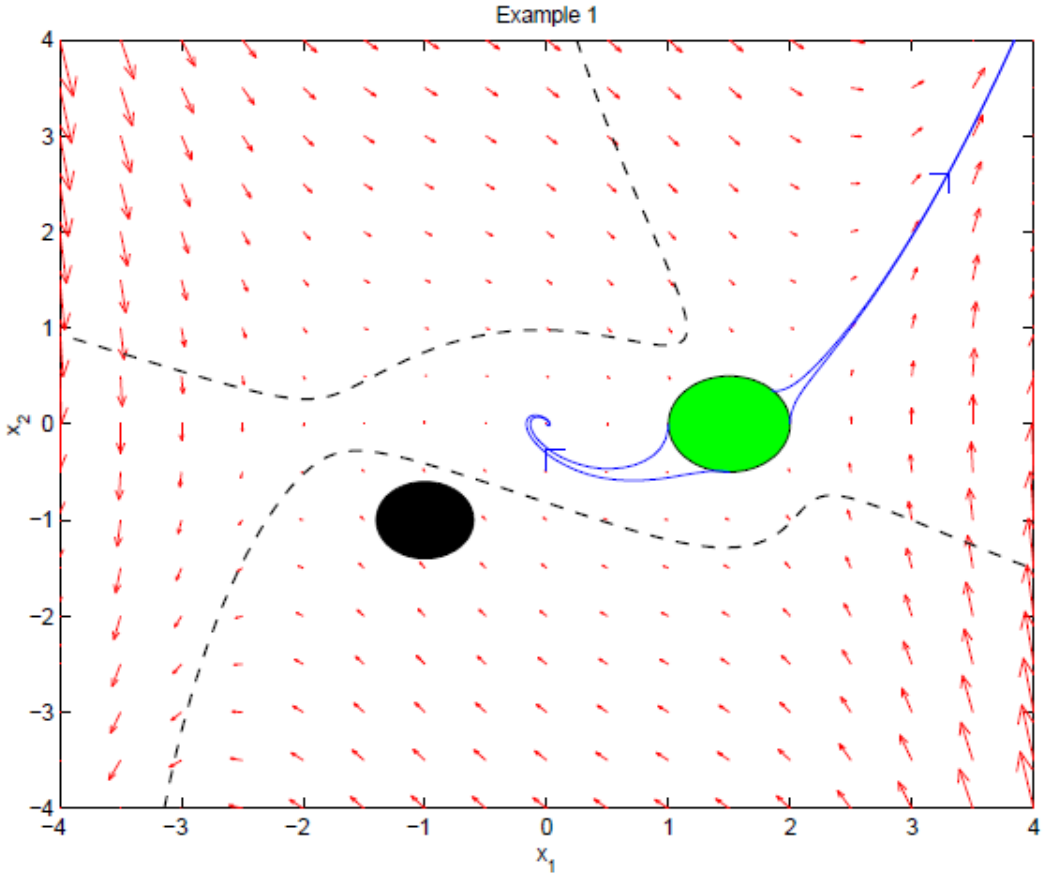


Figure 4.13: A location in an SDP, which will never allow for a run that entered in the solid path on the right to reach the solid patch on the left, due to the dashed barrier certificate [PJ04].

- $d : \mathcal{Q} \to \mathcal{N}$ defines for each discrete state the dimension of the continuous state;
- $\mathcal{X} : \mathcal{Q} \to \mathbb{R}^{d(.)}$ maps each $q \in \mathcal{Q}$ to an open subset $X^q \subseteq \mathbb{R}^{d(.)}$. This is the continuous domain associated to each discrete state;
- $b : X(\mathcal{Q}, d, \mathcal{X}) \to \mathbb{R}^{d(.)}$ represents the flow;
- $\sigma : X(\mathcal{Q}, d, \mathcal{X}) \to \mathbb{R}^{d(.)\times m}$ represents the diffusion;
- $Init : \mathcal{Q} \times \mathcal{X} \to [0, 1]$ is the initial probability measure;
- $\lambda : X(\mathcal{Q}, d, \mathcal{X}) \to \mathbb{R}^+$ is the transition rate function;
- $R : X(\mathcal{Q}, d, \mathcal{X}) \times \mathcal{B}(X) \to [0, 1]$ is the reset function.

Evolution

A stochastic process $x_t = (Q(t), x(t))$ is called a GSHS execution if there exists a series of stopping times $\tau_0, \tau_1, \tau_2, ...$ such that $\tau_0 = 0$ and $\tau_i \leq \tau_{i+1}$, and

- x_0 is a random variable in $\mathcal{Q} \times X$ extracted according to $Init$;
- for $t \in [\tau_i, \tau_{i+1})$, $q_t = q_{\tau_i}$ is constant and $x(t)$ is the solution of

$$dx(t) = b(q_{\tau_i}, x(t))dt + \sigma(q_{\tau_i}, x(t))dW_t$$

  where W_t is the m-dimensional Wiener process;
- $\tau_{i+1} = \tau_i + \sigma_i$ where σ_i is chosen according to a survivor function given by $F(t) = I_{(t<t_*(\omega_i))} exp(\int_0^t \lambda(i, x_s^i(\omega_i)))ds$;
- The probability distribution of $x(\tau_{i+1})$, the value of a token right after a jump, is governed by $R(q_t, x(\tau_{k+1}^-), \cdot)$.

4.11.2 Analysis method 14: Factorization of Reach Probabilities for GSHS

In models where one tries to detect rare events, it can be advantageous to factor the simulations into multiple smaller simulations. Through the use of methods from large deviation and importance sampling theories one can achieve significant computational speedup [BBK09].

Given are an $\mathbb{R}^n$-valued strong Markov process s_t and the research problem "what is the probability that a run hits a small target set $D \in \mathbb{R}^n$ before a maximum time T_{final}. The idea of factorization of reach probabilities is to define a series of nested subsets $D = D_n \subset D_{n-1} \subset ... \subset D_2 \subset D_1$, and in each iteration of simulations determine the likelihood that the next subset is reached [BBK+05]. The first simulation would start outside of D_1. The moment at which some D_i is reached is called a stopping time. The strong Markov property states that at stopping times, the probability distribution of the future states is independent of the past, conditioned on the current state (i.e. at the stopping time). The probability that a dangerous situation occurs, i.e. D is reached, is the product of the conditional probabilities that each D_i is reached, given that D_{i-1} has already been reached. This means that we can use the final states of such smaller simulations, at the moment when the runs enter D_i, as the initial states of the simulations where the goal is state D_{i+1}. In many cases there is a clear way of defining such nested subsets. For example, in collision risk models each set D_i is defined as the first time that two aircraft are less than a given distance apart.

The estimation of the probability to reach the next set is recursively characterized as

$$\begin{array}{ccccc} \pi_{k-1}(\cdot) & \overset{prediction}{\rightarrow} & p_k(\cdot) & \overset{conditioning}{\rightarrow} & \pi_k(\cdot) \\ & & \downarrow & & \\ & & \gamma_k & & \end{array}$$

where $\gamma_k = P(\tau_k < T_{final} | \tau_{k-1} < T_{final})$. We define the stopping times $\tau_k, k = 1, 2, 3, ...$ as the first time that the Markov process $\{s_t\}$ hits subset D_k, i.e. $\tau_k = inf\{t > 0 \cap s_t \in D_k\}$. The probability of hitting target set D before some final time T_{final} can then be given as $P(\tau_m < T_{final}) = \prod_{k=1}^m \gamma_k$.

Factorization can speed up simulations in two ways. The first is through using an interacting particle system (IPS) [BBK09]. Here, importance sampling is used such that the same accuracy

can be attained with fewer runs. The second way to use stopping times is to use the fact that some elements might be irrelevant after a given stopping time has been reached. For example, in a model of two aircraft that fly parallel, wind and the possibility of vital instruments breaking are included. But once the aircraft are less than 100 meters apart, a collision will occur or be avoided within 10 seconds. The effect that wind has and the chance that the engine fails within these 10 seconds might be negligible. We see that the model might be simpler at certain crucial moments. It may be possible in some cases to use other analysis techniques on this simplified system, that could not be used on the original model.

4.12 Analysis Methods for Discrete-time Stochastic Hybrid Systems

In this section we will explain how steady state analysis can be applied to discrete-time stochastic hybrid systems (DtSHS). We do this by showing that DtSHS can be approximated by DTMC to any degree of accuracy. Once DtSHS is abstracted to DTMC, steady state analysis can be performed.

4.12.1 Description

In a Discrete-time stochastic hybrid system, at each discrete time step the system is in a state, and determines probabilistically, through kernels, what event will occur [AKLP10]. These events are continuous evolution both to itself and neighboring states, and discrete transitions. The likelihood of these events depends upon the current state. Formally, a DtSHS-based model is given by:

- $Q = \{q_1, q_2, ..., q_m\}$ with $m \in \mathbb{N}$ the discrete state space;
- $n : Q \to \mathbb{N}$ is a function giving to each element from Q its dimension of the continuous state space $\mathbb{R}^{n(q)}$. The hybrid state space is then $S = \bigcup_{q \in Q} \{q\} \times \mathbb{R}^{n(q)}$, with elements $s = (q, x)$, where $q \in Q$ and $x \in \mathbb{R}^{n(q)}$;
- Init: $S \to [0, 1]$ is a probability measure on S that defines the initial state;
- $T_x : \mathcal{B}(\mathbb{R}^{n(\cdot)}) \times S \to [0, 1]$ is a conditional stochastic kernel on $\mathbb{R}^{n(\cdot)}$ given S. It assigns to each $s = (q, x) \in S$ a probability measure, $T_x(\cdot|s)$, on the Borel space $(\mathbb{R}^{n(q)}, \mathcal{B}(\mathbb{R}^{n(q)}))$. The function $T_x(A|(q, \cdot))$ is assumed to be Borel measurable, for all $q \in Q$ and all $A \in \mathcal{B}(\mathbb{R}^{n(q)})$. T_x determines the next continuous state, given that the system does not jump;
- $T_q : Q \times S \to [0, 1]$ is a conditional discrete stochastic kernel on Q given S, which assigns to each $s \in S$ a probability distribution, $T_q(\cdot|s)$, over Q. This function determines the likelihood of a jump;
- $R : \mathcal{B}(\mathbb{R}^{n(\cdot)}) \times S \times Q \to [0, 1]$ is a conditional stochastic kernel on $\mathbb{R}^{n(\cdot)}$ given $S \times Q$, that assigns to each $s \in S$ and $q' \in Q$, a probability measure, $R(\cdot|s, q')$, on the Borel space $(\mathbb{R}^{n(q')}, \mathcal{B}(\mathbb{R}^{n(q')}))$. The function $R(A|(q, \cdot), q')$ is assumed to be Borel measurable, for all $q, q' \in Q$ and all $A \in \mathcal{B}(\mathbb{R}^{n(q')})$. This is the reset function, which determines the continuous variables, given that a discrete jump occurs.

Evolution

The execution of a run of a discrete time stochastic hybrid system is as follows.
Find initial state by using $Init$;
repeat
  use $T_q(\cdot|(q_k, x_k))$ to find $q_{k+1} \in Q$ given q_k;
  if $q_{k+1} = q_k$ **then**
    use $T_x(\cdot|(q_k, x_k))$ to find $x_{k+1} \in \mathbb{R}^{n(q_{k+1})}$;
  else
    use $R(\cdot|(q_k, x_k), q_{k+1})$ to find $x_{k+1} \in \mathbb{R}^{n(q_{k+1})}$;
  end
until *stopping criteria*;

Instead of two distinct conditional stochastic kernels, one for discrete transitions and one for continuous evolution, we can also define them in one kernel;

$$\begin{aligned} T_s(\{q'\} \times A_{q'}|(q,x)) &= T_x(A_{q'}|(q,x))T_q(q'|(q,x)), \text{ if } q' = q \\ &= R(A_{q'}|(q,x)T_q(q'|(q,x))), \text{ if } q' \neq q. \end{aligned}$$

When, in a given hybrid state (p, x), it is possible to pass to a new state by a spontaneous jump, the transition kernel for that state is $T_p(p'|(p, x)) > 0$. When there are also forced jumps, this is done by setting the transition kernel for all hybrid states outside of the domain towards the same place to zero: $T_p(p|(p, x)) = 0$.

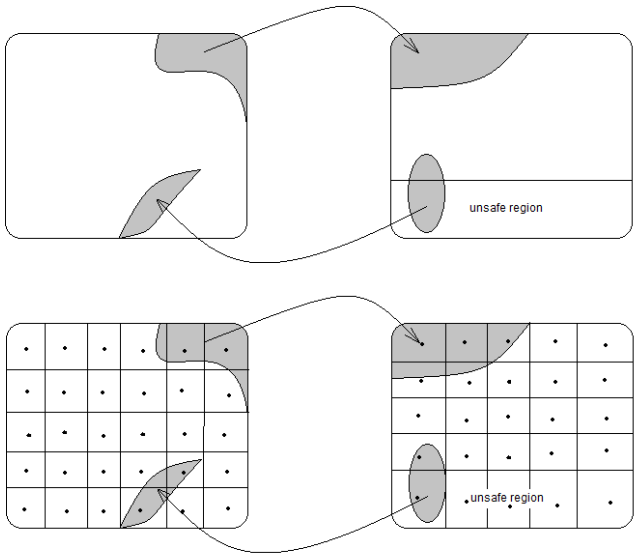


Figure 4.14: A discrete-time stochastic hybrid system and how to discretize it. States will be neighbors if one can be reached from the other either through continuous evolution or discrete evolution in the original DtSHS. The probability of transitions and selfloops will also be determined by the original probability of discrete/continuous evolution.

4.12.2 Analysis Method 15: Approximation to DTMC for DTSHS

We discretize the entire state space by partitioning it into separate (equal) volumes, see figure 4.14. For each volume we choose a point to be the representative of that region. These will be the states of our DTMC. We assume that the created DTMC has a finite number of states. This is only true if the DtSHS was compact. If it was not, we would have to make it compact by dropping a part of the state space. As these are often regions we were not interested in in the first place, we can generally make a model bounded without loss of its expressiveness. Next we define the transition kernel between any two regions. As we use points to represent their region, we define $\Xi(z)$ to be the region that is represented by the point z. we then get the following transition kernel for the DTMC:

$$T_\delta(z'|z) \quad = \quad T_\delta(\Xi(z')|z)$$

The transition kernel between states in the DTMC are determined by both the continuous transition kernel and the discrete transition kernel of the original DtSHS. The edge between two states i, j represents the probability of getting from the region that is represented by i to the region represented by j in one timestep, either by continuous evolution or by an event.

In this way we have approximated the original system by a discrete time Markov chain. Now all we would have to do is perform steady state analysis on this DTMC. Then we would add up the percentages of risky states, and we would thereby get the probability that the system is in a risky state. We should note here that this strategy can be implemented with arbitrary accuracy, as we can make the discrete states small enough such that the representative point really does represent its region.

4.13 Analysis Methods for Arenas of Finite State Machines

Finite state systems are discrete systems, as explained in section 4.3. Arena of finite state machines (AFSM) is a formalism in which the model is a composition of such FSMs. This means that each state of the AFSM is in itself an FSM. For example, each state described in figure 4.1 might itself be a finite state machine looking like figure 4.1. Each edge in the AFSM describes communication from one FSM to another. Of course, the way in which these lower FSMs communicate with each other has to be specified. For a given system we are interested in detecting when the system is in a state for which some critical relation holds. A critical relation might be "system overheating", "aircraft flying too close to each other" or "critical amount of fuel left". A critical relation might be true in multiple states. We define $\mathcal{R}_c \subseteq S$ to be the set of all states for which this critical relation holds.

4.13.1 Description

An AFSM is defined as a directed graph $\mathbb{A} = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V}$ is a set of FSMs $M_i = (S_i, s_{0,i}, \Sigma_i, \Lambda_i, T_i, G_i)$, for $i = 1, 2, ...N$, and $\mathbb{E} \in \mathbb{V} \times \mathbb{V}$ describes which FSMs can communicate with each other [PPBS12]. For each $(M_i, M_j) \in \mathbb{E}, M_j$ can use the output of the current state of M_i (before the jump) as its input.

An AFSM can be folded out to an FSM. This is defined as: $\mathbb{M}(\mathbb{A}) = (S, s_0, \Sigma, \Lambda, T, G)$, where

- $S = S_1 \times S_2 \times ... \times S_N$;

- $s_0 = (s_{0,1}, s_{0,2}, ..., s_{0,N})$;
- $\Sigma = \bigcup_{M_i \in \mathbb{V}} \Sigma_i$
- $\Lambda = \bigcup_{M_i \in \mathbb{V}} \Lambda_i$
- $T \subseteq S \times 2^{\Sigma} \times S$ such that

$$(s_1, s_2, ..., s_N) \xrightarrow{\sigma} (s'_1, s'_2, ..., s'_N) \tag{4.5}$$

where $s_i \xrightarrow{\sigma_i} s'_i$ is a transition in M_i. The label σ of this transition is defined as

$$\sigma = \bigcup_{M_i \in \mathbb{V}} (\sigma_i \backslash (\bigcup_{M_j \in Pre(\mathbb{A}, M_i)} G_j(s_j))) \tag{4.6}$$

where $Pre(\mathbb{A}, M_i) = \{M_j \in \mathbb{V} | (M_j, M_i) \in \mathbb{E}\}$.

- $G(s_1, s_2, .., s_N) = \bigcup_{M_j \in \mathbb{V}} G_j(s_j)$

An important concept in the work concerning AFSMs is that of observers. An observer of an FSM $M = (S, s_0, \Sigma, \Lambda, T, G)$ is defined itself as an FSM with $\mathcal{O} = (\tilde{S}, \tilde{s}_0, \tilde{\Sigma}, \tilde{\Lambda}, \tilde{T}, \tilde{G})$ where $\tilde{S} \subseteq 2^S$ is a set of states, $\tilde{s}_0 \subseteq 2^S$ is the set of initial states, $\tilde{\Sigma}$ is the input alphabet which corresponds to the output alphabet Λ of M, $\tilde{\Lambda}$ is the output alphabet, $\tilde{T}$ is the transition relations, $\tilde{G}$ defines per place the output.

A critical observer $\mathcal{O}_{\mathcal{R}_c}$ of an FSM M and a set of critical states $\mathcal{R}_c$ is an observer whose input is defined as the output of M, and whose output function $\tilde{G}$ gives $\tilde{G}(s) = 1$ iff $s \in \mathcal{R}_c$ and $\tilde{G}(s) = 0$ otherwise. An FSM M is said to be $\mathcal{R}_c$-critically observable if an $\mathcal{R}_c$-critical observer exists. This means intuitively, that if we monitor the output of the system, we know when the system reaches the critical states. It is impossible for the system to reach a critical state without the observer noticing, and likewise it is impossible for the observer to think that the system is in a critical state even though it actually is not.

4.13.2 Analysis Method 15: Maximal Bisimulation Relation for AFSM

Bisimulation is an equivalence relation between two models of the same modeling formalism. If two models are bisimilar they display the same dynamics. This can be used to transform a large model to a smaller one. We will first show how bisimulation works for FSMs, and then show how that definition is extended to AFSMs [PDBDS11].

We now define the term bisimulation equivalence of FSMs. Given two FSMs M_1 and M_2, a relation $R \subseteq X_1 \times X_2$ is a simulation relation from M_1 to M_2 if

- for any $s_0^1 \in S_0^1$ there exists $s_0^2 \in S_0^2$ such that $(x_0^1, x_0^2) \in R$.
- for any $(x^1, x^2) \in R$, $G(x^1) = G(x^2)$.
- for $(x^1, x^2) \in R$, existence of $x^1 \xrightarrow{u_1(\Delta_1)} x_+^1$ implies existence of $x^2 \xrightarrow{u_2(\Delta_2)} x_+^2$ and $u_1 = u_2$ and $(x_+^1, x_+^2) \in R$, where $x^i \xrightarrow{u_i(\Delta_i)} x_+^i$ is a transition in FSM i.

Two FSMs M_1, M_2 are bisimilar if there exists a simulation relation from M_1 to M_2 and one from M_2 to M_1. If two FSMs M_1, M_2 are bisimilar, and M_1 is $\mathcal{R}_c$-critically observable, then M_2 is $\mathcal{R}_c$-critically observable.

The maximal bisimulation relation $\mathbb{R}^*(M_1, M_2)$ is a bisimulation such that $\mathcal{R} \subseteq \mathcal{R}^*(M_1, M_2)$ for any bisimulation relation. The quotient of an FSM M induced by $\mathcal{R}^*(M, M)$ is the minimal (lowest number of states) compositionally bisimilar FSM to M.

This main result can be extended to AFSMs. First define what similarity between two AFSMs means. Two AFSMs $\mathbb{A}^1, \mathbb{A}^2$ are compositionally bisimilar if there exists a relation $\mathbb{R} \in \mathbb{V}^1 \times \mathbb{V}^2$ for which holds that any $(M_i^1, M_j^2) \in \mathbb{R}$ satisfies

- $M_i^1 \cong M_j^2$
- existence of $(M_i^1, M_i^{1,+}) \in \mathbb{E}^1$ implies existence of $(M_j^2, M_j^{2,+}) \in \mathbb{E}^2$ such that $(M_i^{1,+}, M_j^{2,+}) \in \mathbb{R}$
- existence of $(M_j^2, M_j^{2,+}) \in \mathbb{E}^2$ implies existence of $(M_i^1, M_i^{1,+}) \in \mathbb{E}^1$ such that $(M_i^{1,+}, M_j^{2,+}) \in \mathbb{R}$

It was shown in [PBS11] that if two AFSMs $\mathbb{A}_1$ and $\mathbb{A}_2$ are bisimilar then the FSMs generated by unfolding the AFSMs, $\mathbb{M}(\mathbb{A}_1)$ and $\mathbb{M}(\mathbb{A}_1)$ are also bisimilar. This notion of bisimilarity can be used to reduce the size of AFSMs, by looking for a bisimulation with a model with fewer states. Futhermore, the maximal bisimulation relation $\mathbb{R}^*(\mathbb{A}_1, \mathbb{A}_2)$ is a bisimulation such that $\mathbb{R} \subseteq \mathbb{R}^*(\mathbb{A}_1, \mathbb{A}_2)$ for any bisimulation relation. The quotient of an AFSM, denoted by $A_{min}(\mathbb{A})$ induced by $\mathbb{R}^*(\mathbb{A}, \mathbb{A})$ is the minimal (lowest number of states) compositionally bisimilar AFSM to $\mathbb{A}$.

The above results provide tools to develop a complexity reduction scheme for an AFSM $\mathbb{A}$ through finding the maximal bisimulation relation:

- compute the maximal bisimulation relation $\mathbb{R}^*(\mathbb{A}, \mathbb{A})$;
- compute the quotient $A_{min}(\mathbb{A})$;
- expand this quotient to the corresponding FSM, $\mathbb{M}(A_{min}(\mathbb{A}))$;
- compute the maximal bisimulation relation $\mathbb{R}^*(\mathbb{M}(A_{min}(\mathbb{A})), \mathbb{M}(A_{min}(\mathbb{A})))$;
- compute the quotient $A_{min}(\mathbb{M}(A_{min}(\mathbb{A})))$.

Using the above technique one is able to obtain an AFSM model which contains exactly the same paths as does the original model, and it is the smallest possible AFSM which does so.

4.14 Analysis Methods for Communicating Piecewise Deterministic Markov Processes

Communicating Piecewise Deterministic Markov Process (CPCP) is an extension of the PDP formalism [Dav84]. It combines the existing rules for piecewise deterministic Markov processes (PDP) with methods to specify PDP in a compositional way. In order to model a CPDP then, a researcher can first specify the region behavior, and then use predescribed rules to combine them and form the total of the dynamics in an accurate manner. This is much easier for complex systems than having to model the global dynamics immediately.

4.14.1 Description

PDP is defined as a tuple $\{L, Inv, Flow, \lambda, Q\}$ [Str05], where

- $L = \{l_1, l_2, ..., l_m\}$ is the set of locations;
- V is a set of variables. $d(x)$ with $x \in V$ denotes the dimension of the variable x;
- $v : L \rightarrow 2^V$ assigns to each location a subset of V;
- Inv : assigns to each $l \in L$ an open subset of $v(l)$. The hybrid state space of the PDP is defined as $E = \{(l, x) | l \in L, x \in Inv(l)\}$. $(l_0, x_0) \in E$ is the initial state;
- $Flow$ assigns to each location a locally Lipschitz vector field that determines a flow $\phi(t, \eta) \forall \eta \in E$;
- $\lambda : \eta \rightarrow \mathbb{R}_+, \eta \in E$ determines the jump rate from any hybrid state;
- $Q : E \cup \delta E \rightarrow Prob(E)$ is the transition measure. Given the hybrid state at the moment of a transition (either forced or spontaneous), $Q(l, x)$ determines the probability measure of the next state.

Evolution of PDP

At time $t = 0$ the PDP is intitialized at (l_0, x_0). Until a jump occurs the discrete state is constant, the continuous state evolves continuously, following the vector field defined by $flow$. A jump occurs whenever (a) the continuous state hits the boundary of the state space, or (b) a spontaneous jump occurs with jump rate $\lambda(l, x)$ where (l, x) is the current state. The reset, i.e. the state the system is in right after a jump, is determined by $Q(l, x)$.

Definition of CPDP

In order to be able to build the PDP compositionally, we add passive transitions. CPDP is a tuple $(L, V, v, Inv, Flow, W, w, \Sigma, B, P, S, C, G)$ [SVDS05a], where

- $L, V, v, Inv, Flow$ are defined like in PDP;
- W is a set of output variables;
- w maps each location in L to a subset of W;
- Σ is a set of communication labels;
- B is the set of boundary transitions, which consist of (l, a, l', R), where l is the location where the transition can occur, a is the label of the transition, l' is the target location, and the reset map R defines over each boundary state in l a probability measure on the invariant of l';
- P is the set of passive transitions, where (l, a, l', R) is defined as it is in B except that R is defined over the entire invariant of l;
- S is the set of spontaneous transitions, with (l, λ, a, l', R). Here, $\lambda : Inv(l) \rightarrow \mathbb{R}_+$ is the jump rate function, the other elements are defined like in P;

- C is the choice function. C assigns to each boundary point a probability on the outgoing boundary hit transitions, in order to define what happens when a forced transition occurs, and multiple candidate transitions are possible. Likewise, for each passive transition α from l, C assigns to each triplet (l, α, x), with $x \in Inv(l)$, a probability measure;
- $G(l, w)$ determines for each location the output equation.

Evolution of CPDP

The evolution of a CPDP without passive transitions is completely analogous to PDP evolution. The only organizational difference is that the reset map (Q in PDP) is now described in S and B. If there are passive transitions, then these transitions are triggered by the external environment. When these triggers occur is not described by the CPDP model. However, when we use CPDP to build compositional models, active transitions will trigger passive transitions, and no external environment is necessary to complete the model.

Composition of CPDPs

The composition of two CPDPs X and Y can be given by $X|A|Y$, where A is a set of labels. Such composition means that the two models run simultaneously, and transitions with labels in A are synchronized, i.e. can only fire at the same time if both are enabled. In these composed models active transitions (spontaneous and forced transitions) force passive transitions to fire. [Str05] shows that under reasonable conditions the resulting model is again a CPDP.

4.14.2 Analysis Method 17: Bisimulation for CPDP

A negative side effect of compositional modeling is the state space explosion problem. Every combination of locations from subsystems is a location in the composed model. One way to deal with this is to reduce the state-space by bisimulation.

A relation $\mathcal{R}$ on the hybrid state space E of CPDP X is called a bisimulation if for any $(e_1, e_2) \in \mathcal{R}$,

- $w(loc(e_1)) = w(loc(e_2))$, meaning that the variables that matter for output are the same for any bisimilar states;
- $\forall w_i \in w(loc(e_1)) : G(loc(e_1), w_i)(e_1) = G(loc(e_2), w_i)(e_2)$, meaning that the outputs are equal for two bisimilar states;
- $Flow(t, e_1) = Flow(t, e_2) \forall t > 0$ assures that two bisimilar states remain bisimilar;
- $\sum_{\alpha \in S(loc(e_1))\rightarrow} \lambda_\alpha(e_1) = \sum_{\alpha \in S(loc(e_2))\rightarrow} \lambda_\alpha(e_2)$, which means that the total jump rate is equal for bisimilar locations;
- $\forall \sigma \in \Sigma$ we have that if a σ-transition is enabled at e_1 with reset measure R, then there exists a σ-transition enabled from e_2 with equivalent reset measure.

The above states that if a model has two locations which are similar in the output, flow, and transitions, then there exists a smaller model with the same behavior. When we consider output-similar places, the task becomes similar to AFSM bisimulation. [Str05] furthermore shows that this analysis holds true under composition. This means that we can find the

maximal bisimulations for all component CPDPs, then use composition to make it one model, after which we can again reduce the statespace by bisimulation.

We now give an algorithm which tries to find the maximal bisimulation relation for a CPDP-based model. The algorithm does not necessarily find a bisimulation if one exists, and if it finds a bisimulation this is not necessarily the maximal bisimulation relation.

The algorithm consists of three steps [SvdS05b]. In the first step we find a partition of the locations L such that all locations with bisimilar dynamics are in the same class. For each two locations l, l' in the same class the maximal continuous bisimulation $\mathcal{R}_{l,l'}$ is determined, where a continuous bisimulation is a relation on the state spaces such that the first two items of the definition of bisimulation are upheld. Two reset measures r, r' are equivalent with respect to the partition made if their target locations lie in the same class and they are equivalent with respect to $\mathcal{R}_{l,l'}$. In step two a partition of all reset measures is made, in such way that resets that are equivalent with respect to the partition of step one are in the same class. The third step is a recursive step. In each iteration the two partitions of step one and two are refined if two elements in all class can be differentiated, i.e. they are not equivalent. Once a fixpoint is reached a bisimulation is found.

4.15 Summary

Table 4.1 presents the results of this chapter. It shows for which formalisms we found analytical methods, and what questions these methods try to answer. One interesting thing we notice is that automata answer the analysis problems of safety verification, verification and falsification, whereas hybrid systems that are not automata answer probabilistic verification, steady state distribution and complexity reduction. Automata do not define the likelihood of paths, which means that different analysis problems are interesting for researchers than those in systems that do determine the likelihood of paths.

Table 4.1: Overview of Analysis Methods

Formalism	**Method**	**Analysis Problem**
FSM	Breadth first search	Safety verification
FSM	Model checking	Verification
TA	Abstract to FSM, then use breadth first search	Safety verification
TA	Abstract to FSM, then use model checking	Verification
SMTA	Abstract to TA, then abstract to FSM, then use forward analysis	Safety verification
SMTA	Abstract to TA, then abstract to FSM, then use model checking	Verification
CTMC	Steady state analysis	Steady state distribution
DTMC	Steady state analysis	Steady state distribution
DTMC	Probabilistic reachability	Probabilistic reachability
HA	RRT-based falsification	Falsification
LHA	Forward analysis	Safety verification
LHA	Backward analysis	Safety verification
LHA	Approximate analysis	Safety verification
LHA	Minimization	Safety verification
SDP	Barrier certificates	Probabilistic reachability
GSHS	Factorization of reach probabilities	Probabilistic reachability
DTSHS	Approximate to DTMC, then use steady state analysis	Steady state distribution
DTSHS	Approximate to DTMC, then use probabilistic reachability	Probabilistic reachability
AFSM	Bisimulation complexity reduction	Complexity reduction
CPDP	Bisimulation complexity reduction	Complexity reduction

Chapter 5

Applicability to SDCPN

The previous chapter discussed various kinds of modeling formalisms and some of their important properties or analysis tools. This chapter shows how these tools might be applied to SDCPN. There are always three aspects for each such tool that should be examined. Firstly, we explain why the analysis might be useful. What are the particular strengths of this analysis method? Secondly we look at the limits of this method. What conditions have to be assumed for an SDCPN to fit the modeling requirements of the analysis method? How often do subsystems display such behavior? Are there other factors that limit the use of this tool? Thirdly we look at the opportunities that the method presents to SDCPN analysis. Given the strengths and limitations, in what way could the method be used to analyze SDCPN? Is it useful to apply the technique to sub-systems within SDCPN models, such as LPN, directly? Is there some indirect way that this analysis could be used for SDCPN? This chapter follows the same ordering as chapter 4.

Language Equivalence

For each modeling formalism we will discuss what limits must be put on an SDCPN-based model, in order for them to be mathematically equivalent. However, depending on the formalism we mean different things with equivalence. Normally, with equivalence we mean it is possible to create a model that displays the exact same behavior, the same range of possible runs and the same distribution of runs (in probabilistic terms). We do this because when they are equivalent, the method could be used on both systems, the original hybrid system and the restricted SDCPN. In some cases we say 'language equivalence'. The language of a system is the total collection of possible runs. Where we say language equivalence, the models still portray the same behavior and possible runs, but the distribution of runs is neglected. In those cases, language equivalence is enough for the method to work on both systems.

5.1 Breadth First Search for FSM

Strengths

Reachability for finite state machines is a decidable property. There are relatively straightforward algorithms that can check which states are reachable in a given model. Breadth first search can be fully automated, is fast and can handle large FSMs. Furthermore, it is exact, i.e. we do not need to approximate the answer.

Equivalence. *An SDCPN model is language equivalent to an FSM if*

- $\forall P \in \mathcal{P} : \mathcal{C}(P) = \emptyset$;
- $\forall T \in \mathcal{T} : T \in \mathcal{T}_I$;
- *The number of tokens remains finite for* $t \to \infty$.

Limitations

For SDCPN models to behave equivalently to FSMs, no continuous behavior and no delay transitions are allowed. A local Petri net would have to consist of only immmediate transitions and places with $\mathcal{C}(P) = \emptyset$. Time could not progress in such system, as the token would take immediate transitions endlessly, never allowing any continuous evolution to occur.

If there are multiple tokens in the SDCPN model, the model has to be converted to one that has only one token.

Breadth first search can handle large FSMs, but there is a limit. This limit is due to memory size and computation time. Thanks to new research into large FSM reachability analysis these bounds have increased to 10^7 nodes [CCQ96]. However, advanced techniques have to be implemented to reach such results.

Opportunities

There are two ways in which we could apply FSM reachability analysis to tools for the analysis of SDCPN. Firstly, we could try to abstract SDCPN to finite state machines. This is not possible directly, because of the big difference between these two formalisms as explained in the limitations, but it might be possible to do this via an intermediate formalism. This we will discuss at timed automata analysis and simple multi-rate timed automata later in this chapter.

The second opportunity that FSM reachability analysis presents to SDCPN researchers is in verification. When dealing with very large SDCPN models, it is very difficult to have a good overview of all dynamics. For example, the graphical representation of a large SDCPN model might cover ten to twenty A-4 standard size papers. To follow which transitions and places are connected, especially if they are in different agents or local Petri nets, can be difficult. It might occur that an arc is omitted unintentionally. Often, this leads to unwanted unreachable markings, i.e. markings that should be reachable, but are not. The FSM breadth first search analysis method can inspire us to create a verification tool that can check for such simple errors:

Input: SDCPN, markings that should be reachable.
Create the reachability graph (RG) of the SDCPN [Eve10].
Check whether each marking that should be reachable coincides with a node in the RG.

In [Eve10] it is explained how to construct a reachability graph from an SDCPN. This makes it possible to determine which markings are impossible to reach. If any of the markings that should be reachable do not correspond to a reachable state in the RG, the researcher has made a mistake in the modeling process and should determine where the mistake is and fix it.

This analysis is an overapproximation. Many states that can be reached in this analysis cannot be reached in the actual SDCPN. This is because the breadth first search acts as though

any transition in the RG can be taken. However, it might be possible that a transition in the RG is based on a guard transition that is never enabled (due to the underlying dynamics). However, due to human error it is very possible that a marking that should be reachable is not even in this overapproximation, which this analysis will capture. This analysis technique can therefore be used as an automatic checker for some human errors.

Example

To illustrate, consider the original example introduced in chapter 2 and 3, also shown in figure 5.1. We have created the reachability graph and can use reachability analysis to find that all states shown in this graph are reachable. That means that if the modeler needed to have both $P_2P_5P_6$ and $P_2P_5P_7$ to be reachable, then he made no mistakes (at least none that this method can find). However, if the researcher had wanted that the plane had stopped (P_5), while the pilot had not initiated braking yet (P_1), then this method tells him that he has made a mistake in his design.

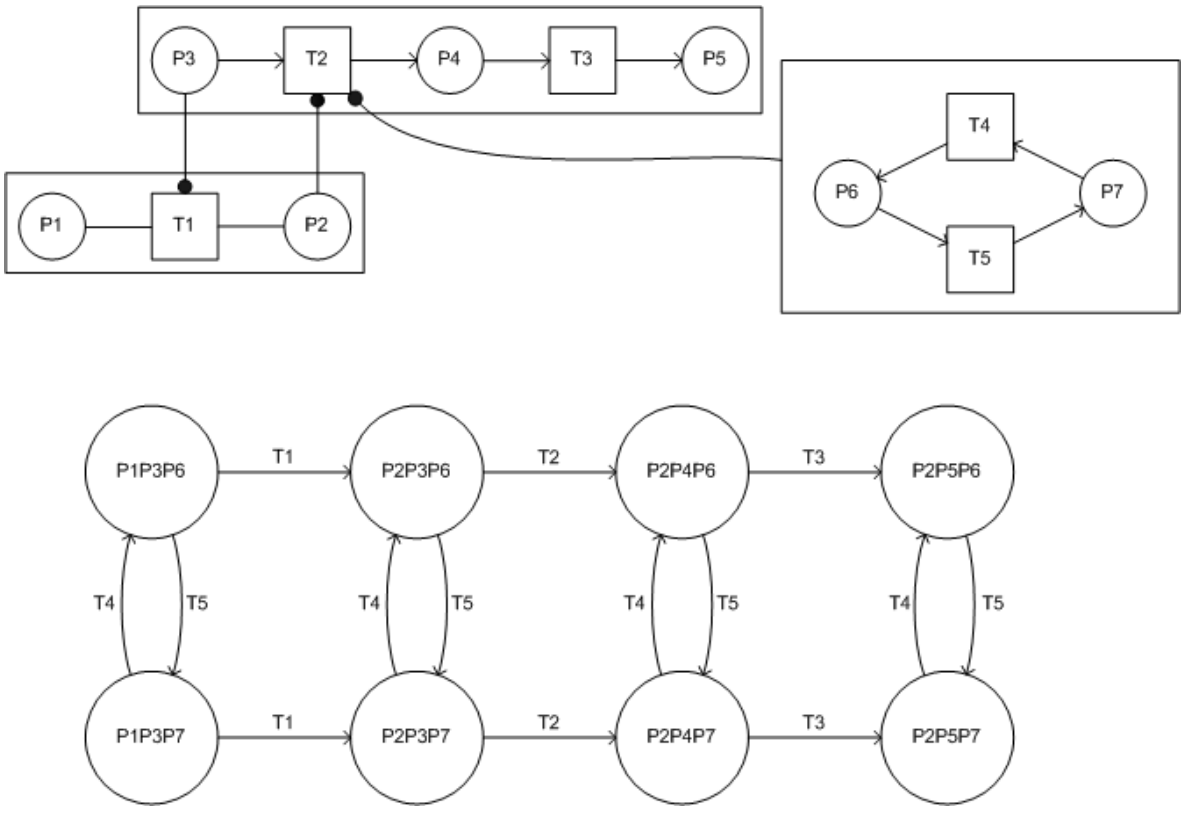


Figure 5.1: Above: the original SDCPN. Below: the corresponding reachability graph.

5.2 Model Checking for FSM

Strengths

Model checking goes one step further than reachability analysis. Checking that asserted properties are true is one of the primal ways of validating a model. It is important to be able

to validate a model, as it can increase our confidence that the model does indeed reflect reality. This in turn increases our confidence in the obtained data from the model. Model checking FSMs has all other strengths of reachability analysis for FSMs.

The restrictions under which SDCPN is equivalent to a finite state machine are given in the previous section.

Limitations

The limitations of reachability analysis for FSMs apply to model checking as well. For equivalence, the same restrictions on SDCPN apply. Furthermore, in order to use this technique it must be possible to determine for each marking whether an atomic proposition holds. This puts a limit on what kinds of propositions can be used. One kind of propositions of interest, those about the continuous variables of places, will be out of scope. Lastly, we must remember that looking at the discrete dynamics in the FSM obtained through the reachability graph is an overapproximation in comparison to the hybrid dynamics of SDCPN models. Therefore, propositions should be chosen of the form $\neg\exists \diamond \neg\phi$ or $\forall\Box\phi$. If the analysis returns that the proposition is true then the SDCPN is also safe. If the proposition is not true then witness trajectories can be formulated where the violation occurs. A researcher can decide whether the witness trajectories were false inclusions due to overapproximation or whether the model contains a mistake.

Opportunities

The main opportunity for model checking lies with model checking that includes the continuous dynamics. However, it is also possible to perform model checking on the associated RG of an SDCPN, similar to reachability analysis in the section above. Model checking can then be used as a means of checking for human errors. To verify a model, a researcher should then formulate a set of propositions, and check whether these hold in the model. A yes answer means the system is verified. A no answer gives rise to a set of witness trajectories, which should be studied by the researcher to check whether the system was modeled correctly.

Example

Consider again the SDCPN and its associated RG in figure 5.1. Assume we want to model check the statement "the airplane can never be in non-nominal braking mode before the pilot has started braking", $\phi := \forall\Box\neg(\mathit{not\_yet\_braking} \cap \mathit{non\_nominal\_braking})$. There are two atomic propositions; $\mathit{not\_yet\_braking}$ which is true in $\{P_1P_3P_6, P_1P_3P_7\}$ and $\mathit{non\_nominal\_braking}$ which is true in $\{P_1P_3P_7, P_2P_3P_7, P_2P_4P_7, P_2P_5P_7\}$. Moving up the logic tree, we find that $(\mathit{not\_yet\_braking} \cap \mathit{non\_nominal\_braking})$ is false everywhere except in $\{P_1P_3P_7\}$. Therefore, $\neg(\mathit{not\_yet\_braking} \cap \mathit{non\_nominal\_braking})$ is true everywhere except in $\{P_1P_3P_7\}$. The last step is applying $\forall\Box$, thus finding that ϕ holds for all states except $\{P_1P_3P_6, P_1P_3P_7\}$. When we intersect this set with the set of initial states $\{P_1P_3P_6, P_1P_3P_7\}$ we find that the model is not verified. Therefore, if ϕ needed to hold for the model, a modeling error has been made. This entire procedure can be done fully automated.

5.3 Abstraction to FSM for TA

Strengths

Timed automata can be abstracted into finite state machines, because of which the analysis techniques that apply to FSM can also be used to these more complex models. As timed automata are closer to nonlinear dynamics than FSM there is less loss of modeling power.

Equivalence. *An SDCPN model is language equivalent to a timed automaton if*

- $\mathcal{C}(P_i) = \mathcal{C}(P_j)\ \forall P_i, P_j \in \mathcal{P}$*;*
- $\forall P \in \mathcal{P} : \mathcal{V}(P) = \{1, 1, ..., 1\}$*;*
- *The firing functions are deterministic, and set each token value either to 0 or to the value of the token removed;*
- $\mathcal{W} = 0$*;*
- *Constants in guards are integers;*
- *The number of tokens remains finite for* $t \to \infty$*.*

Note however that the timed automaton formalism does not provide a mechanism how runs are generated. Equivalence therefore means that for any SDCPN that fits the given restrictions one could build a timed automaton that allows the same range of paths as the SDCPN does.

Limitations

When a timed automaton is abstracted to a finite state machine, the state space explosion problem can occur. For each state in the TA, many states in the FSM are created to preserve all possible paths. This is more likely to happen if the TA has many states, or if the TA has large constants in its guards or invariants.

Opportunities

This analysis method can be applied to SDCPN sub-systems that are equivalent to TA, see the equivalence above. First we create the reachability graph (RG) of the SDCPN as in [Eve10]. Our TA $H = \{\mathcal{P}, P_0, \Sigma, E, C, Inv, Grd, Rst\}$ will then be built as follows: $\mathcal{P}$ is set equal to the set of nodes in RG. P_0 is set equal to the starting configuration of our SDCPN. If the SDCPN allows for a range of starting positions, a range of TA can be built each with different P_0. E is set equal to the edges in RG, and each edge gets a label in Σ equal to the label in the RG for that transition, which is the name of the transition in the SDCPN. C is equal to $\mathcal{C}(P_0)$, which is the set of clocks which is the same for each place of the SDCPN. Inv is built for each place by the guard transitions that have that place as their input place. Grd is placed on edges that correspond with guard transitions in the SDCPN, with the same guard as before. Grd is placed on edges corresponding to delay transition with a guard that each variable be greater than zero, as a delay transition cannot fire immediately. Note that delay transitions thus become unguarded transitions, and immediate transitions have vanished in the RG. Rst is equal to $\mathcal{F}$, and as specified by the equivalence relation can model either no

change in the variable or a reset to 0. Once this TA is created, we can perform abstraction to FSM and the corresponding reachability analysis.

Note that it is common for a TA obtained in the above manner to have unbounded locations, meaning locations that have one or more clocks not affected by any invariant. The FSM we get from abstracting such TA would have an infinite number of states. The solution could be forcing an invariant on such locations, in such a way that no interesting behavior is cut from the model.

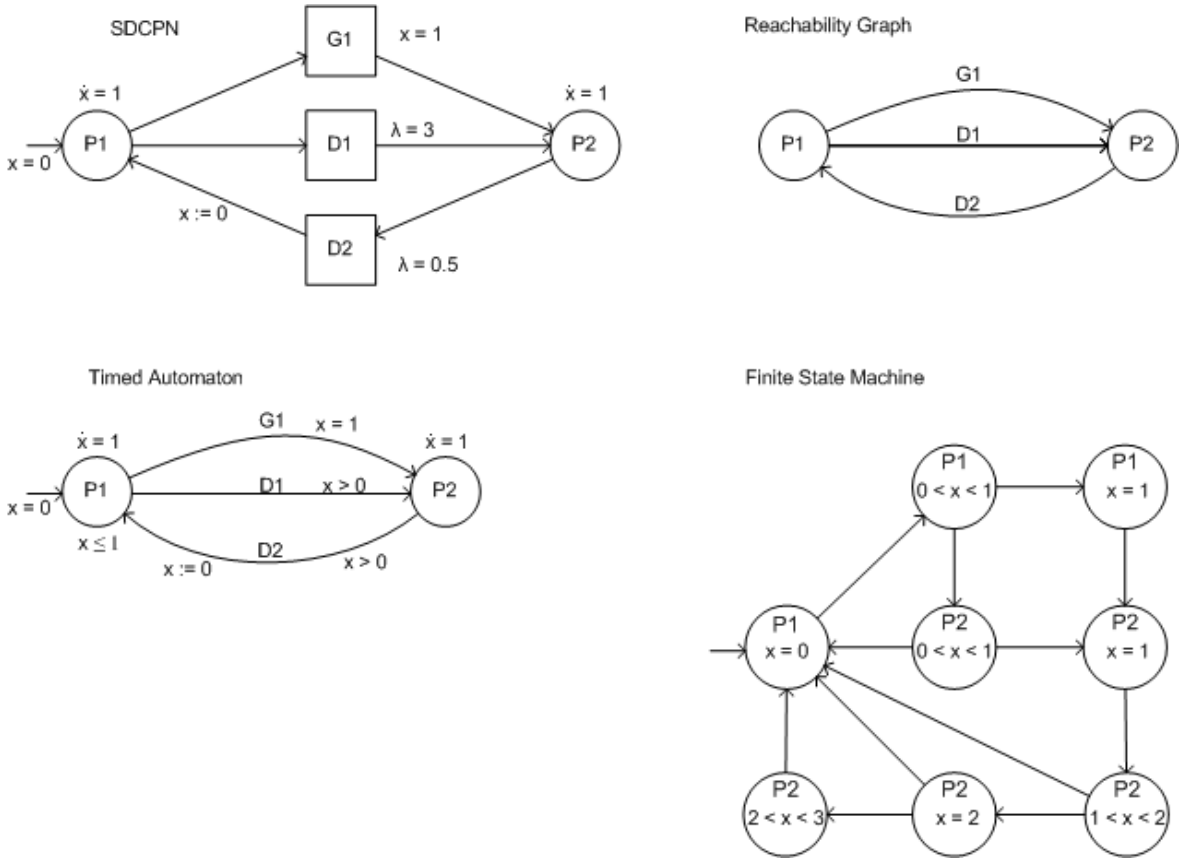


Figure 5.2: Abstracting an SDCPN into a timed automaton.

Example

Assume we have a sub-system of an SDCPN model like in figure 5.2. We would like to determine whether $x \geq 2$ is possible. The arrow entering in each model indicates that the model starts there, with $x = 0$. We first check that the SDCPN behaves like a TA and see that it is indeed so. Therefore, we create the reachability graph (above right) and then the timed automaton (below left). Notice that location P_2 has no invariant constraining x. Therefore we set the invariant for this location as $x < 3$, since we know any values above 3 do not change our investigation. This TA we abstract to the FSM (below right). Breadth-first search finds the state of the FSM $P2, 2 < x < 3$ to be reachable, so we know that $x \geq 2$ is possible.

5.4 Abstraction to TA for SMTA

Strengths

SMTA extend timed automata by allowing clocks with rates within given margins. The strength is the greater ease with which an SDCPN model can be approximated by an SMTA. The biggest strength of SMTA analysis compared to TA analysis is that we make weaker assumptions on the flow of the SDCPN. The idea will be to choose the range such that it includes the flow and diffusion for a given variable.

Equivalence. *The language of an SDCPN model can be approximated by a simple multi-rate timed automaton under the following conditions:*

- $\mathcal{C}(P_i) = \mathcal{C}(P_j)\ \forall P_i, P_j \in \mathcal{P}$*;*
- *The firing functions are deterministic, and set each token value either to 0 or to the value of the token removed;*
- *Between two states that have different boundaries on the changes of a variable x, x is reset to 0 along that edge;*
- *Guards can only check variables against constants, not against other variables;*
- *Constants in the guards are integers;*
- *The number of tokens remains finite for $t \to \infty$;*
- *For any place, the flow $\mathcal{V}$ and diffusion terms $\mathcal{W}$ combined are positive for any valid valuation.*

Limitations

In many cases we can try to approximate an SDCPN with a SMTA while losing some modeling power. For example, due to approximating non-integer constants are not integers. In those cases we round these constants in such way that more paths are included (not fewer). Secondly, the SMTA almost always overapproximates the original model as it acts as though any flow within the margins given by ρ is possible. This is not necessarily true in an SDCPN. The loss of accuracy will be dependent on the model, and as of yet there is no way to give an indication of the magnitude of lost accuracy.

Opportunities

We can transform the SDCPN into an SMTA in the following way. First we assume that the conditions outlined above hold. Then we transform the SDCPN into a SMTA in the same way as was done for timed automata in section 5.3. The only thing different in this case is that we now use the flow $\mathcal{V}$ and diffusion terms $\mathcal{W}$ to determine ρ. ρ should be chosen in such a way that it includes all possible continuous dynamics. The minimum and maximum of ρ will be determined by the minimum and maximum combined effects of $\mathcal{V}$ and $\mathcal{W}$.

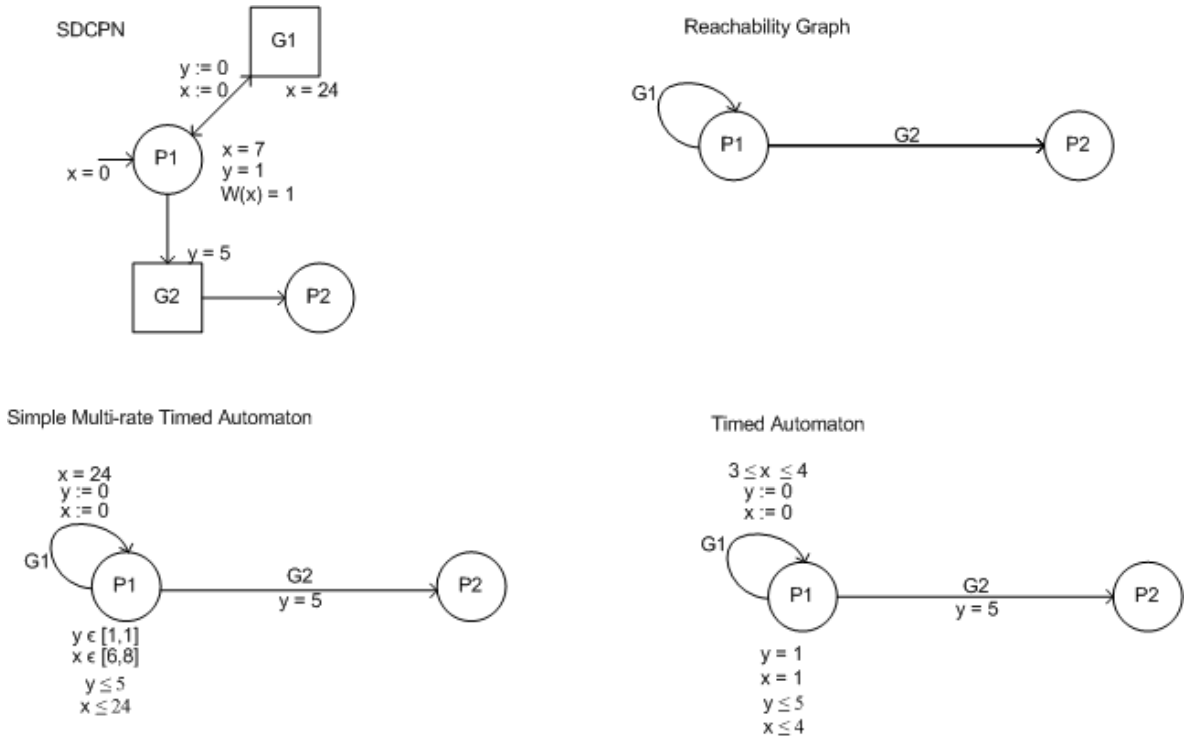


Figure 5.3: Abstracting an SDCPN into a simple multi-rate timed automaton.

Example

The example is depicted in figure 5.3. Place $P1$ sends a signal to other parts of the SDCPN (not depicted here) via transition $G1$. What is important is that this signal is sent at least every 5 seconds. This is what transition $G2$ tests. We want to check whether it is ever possible for $P2$ to be current, i.e. that for 5 seconds no signal has been sent. The difficulty is that the flow of x is influenced by a Wiener process of size 1, denoted in the figure by $W(x) = 1$. We first create a rectangular hybrid automaton (below left), where we form $\rho(P1, x)$ by looking both at the flow and diffusion of x. The creation of the timed automaton (below right) is straightforward. The further steps for evaluating the TA are not shown, but one can implicitly see that $P2$ is not reachable. x and y grow at the same pace, and the invariant on x: $x \leq 4$ prevents y from increasing to 5.

5.5 Steady state analysis for CTMC

For any two states $i, j \in V$, if in the underlying directed graph $G = (V, E)$, there is a path from $i \in V$ to $j \in V$, then it is possible to reach j from i in the original system. Thus the safety verification problem is solved using the finite state machine analysis on the underlying graph of the CTMC. However, we would like to go one step further and investigate the steady state behavior, or long time averages, of a CTMC-based model.

Strengths

Certain elements of SDCPN correspond exactly with the dynamics of continuous-time Markov chains. A sub-system with no continuous dynamics and only delay transitions can be modeled by CTMC without loss of modeling power. The steady state distribution we find with this analysis can be used as further input for the larger model.

Equivalence. *An SDCPN model behaves equivalently to a continuous-time Markov chain if*

- $\mathcal{C}(P) = \emptyset \ \forall \ P \in \mathcal{P}$*;*
- *the number of tokens remains finite for* $t \to \infty$*.*

Limitations

Vital elements of SDCPN, such as continuous evolution, are not implemented in CTMC. CTMC analysis can therefore not be used for the entire SDCPN. Furthermore the sub-systems must satisfy a couple of constraints; it may only have delay transitions, and the places may not have continuous evolution.

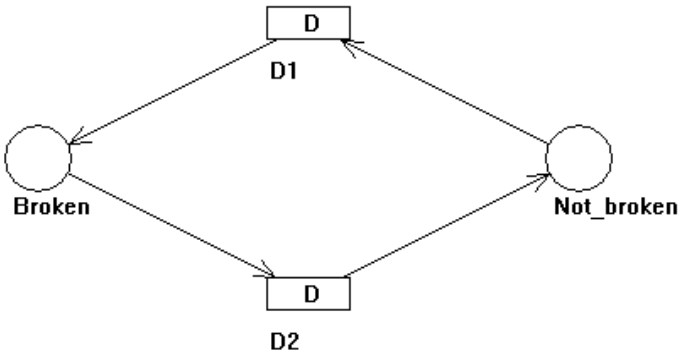


Figure 5.4: Example of an LPN that can be analyzed using CTMC steady state analysis analysis.

Opportunities

As luck would have it, some sub-systems can be isolated that have precisely these properties. These are often LPN that describe failure and operational modes of technical systems, e.g. as in 5.4. For example, if we are considering an altometer, we model the chance of its breaking as a delay transition. The time it takes to repair the altometer is also a negative exponentially distributed parameter represented by a delay transition. Moreover, such sub-systems tend not to be influenced by other LPNs. It generally has only one token as it is an LPN, and it tends to have little or no continuous evolution. An analyst should solve this sub-system analytically first, and then put the information back in the larger system.

Example

This approach is already often used for SDCPN. When we look at the example used in chapter 3, see figure 3.2, we can perform this analysis on local Petri net Braking Performance. Suppose we are interested in the average distance until the aircraft has come to a stop. First, we find the probability of a non-nominal landing analytically. Next, instead of running Monte Carlo simulations of the entire SDCPN, we can run a number of simulations where we set the landing to be non-nominal, and a number of times with nominal landing conditions. In this example, on average 1 in every 10000 runs occurs under non-nominal conditions, which we can find thanks to steady state distribution analysis. In order to observe 1000 non-nominal landings (to get significant data about landing distance during non-nominal landings) we would normally need 1000×10000 landings, which is ten million landings. However, once we know the steady state distribution, we can perform 1000 Monte Carlo simulations conditioned on non-nominal landing and 1000 Monte Carlo simulations conditioned on nominal landing, and use the fact that average landing distance is equal to average landing distance under nominal braking performance $*\frac{9999}{10000}+$ average landing distance under non-nominal braking performance $*\frac{1}{10000}$. This requires only 2000 Monte Carlo simulation runs instead of 10 million, without loss of accuracy of the result.

5.6 Steady State Distribution for DTMC

Strengths

The steady state of a sub-system of a model is important information to a researcher. In many cases the set of places of a sub-system can be split into two sets; a set with dangerous places and a set with safe places. Once we know the steady state distribution, we know how often the token of the sub-system is in the dangerous set, and how often it is in the safe set.

Equivalence. *An SDCPN model never behaves exactly equivalent to a discrete-time Markov chain, because of the difference between continuous-time and discrete-time modeling. However, we can get arbitrarily close to the behavior of an DTMC if*

- $\mathcal{C}(P) = \emptyset \forall P \in \mathcal{P}$*;*
- *the number of tokens remains finite for* $t \to \infty$*.*

Limitations

In cases where we would like to apply this technique directly to sub-systems of SDCPN models, CTMC steady state analysis is the more natural choice. It provides us with the same information, while staying in the same reference frame (continuous-time dynamics) in contrast to DTMC analysis (discrete-time dynamics).

Opportunities

The main opportunity is to first abstract a formalism that includes continuous dynamics to DTMC, and then perform this analysis. This is what is done in discrete-time stochastic hybrid systems analysis.

5.7 Probabilistic Reachability for DTMC

Strengths

This analysis can be used to calculate the probability that a sub-system will enter a dangerous state, given that the system is equivalent with a DTMC model, see section 5.6. This is important in cases where we are not interested in the steady state distribution, but rather in the likelihood that a target state is reached. Furthermore, while steady state analysis only works on ergodic systems, probabilistic reachability has no such limitation.

When creating Monte Carlo simulations, one necessarily transforms continuous-time dynamics to discrete-time dynamics. These discrete-time dynamics can be the basis upon which this method is used.

Limitations

SDCPN is a continuous-time hybrid system, whereas DTMC is discrete-time. Converting SDCPn to DTMC therfore will always include losing some degree of accuracy.

Opportunities

The main opportunity is again to abstract a formalism like DtSHS to DTMC, and then perform this analysis. This creates a much greater range of situations for which this analysis can be used.

Example

Assume we have an SDCPN model like in figure 5.5. All transitions with a number written in them are delay transitions, with that number denoting the delay rate λ. Transition $I1$ is an immediate transition, which fires with 0.5 probability a token to $P4$, and otherwise fires a token to $P1$. In this model $P1$ denotes the nominal state and also the starting place, $P4$ and $P5$ are the target places. Figure 5.6 shows the DTMC we obtain. The equation $\pi_f = \pi_0 Q^t$ thus becomes

$$\begin{pmatrix}1 & 0 & 0 & 0 & 0 & 0\end{pmatrix}\begin{pmatrix}0.98 & 0.01 & 0.01 & 0 & 0 & 0\\ 0.5 & 0 & 0 & 0.5 & 0 & 0\\ 0.1 & 0 & 07 & 0 & 0.1 & 0.1\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 1\end{pmatrix}^{10000} = \begin{pmatrix}0 & 0 & 0 & 0.4286 & 0.2857 & 0.2857\end{pmatrix}$$

From this we obtain that the likelihood that either $P4$ or $P5$ becomes current is 0.7143.

5.8 RRT-based Falsification for HA

Strengths

Falsification has been researched because of the high computational complexity of general model checking. A definite strength of this method is that it can be performed on hybrid models with all kinds of stochastic influence, such as GSHS. Although the technique does not

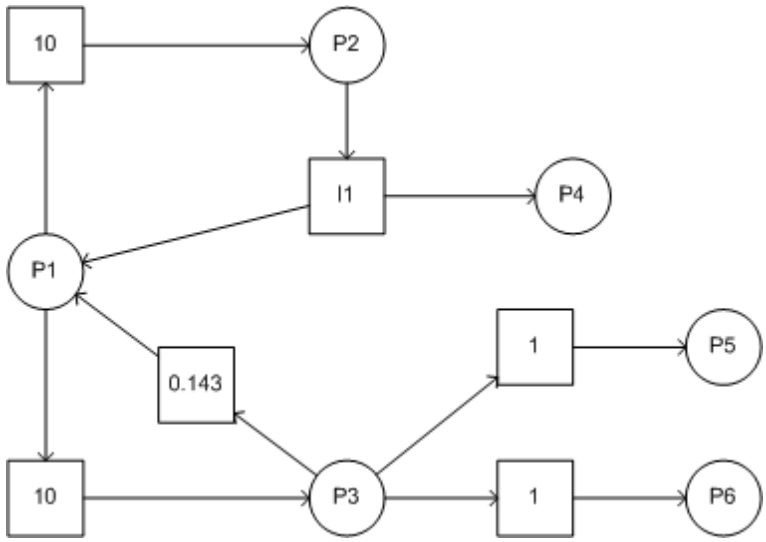


Figure 5.5: Example of an SDCPN model that can be analyzed using DTMC probabilistic reachability analysis.

have the ability to prove correctness of a model, in many cases when a model is incorrect it finds a counterexample, i.e. a path from the intial state to the target state. This also means that if the procedure comes up empty, it is likely that no counterexample exists, so the system is probably safe for that target state. This can give us some indication that a system is safe, which is especially important for systems which are too complex to prove safety for.

Equivalence. *An SDCPN model is language equivalent to a hybrid automaton if [Eve10]*

- *The initial marking does not enable a transition;*
- *No transition immediately enables a guard transition;*
- *The number of tokens remains finite for* $t \to \infty$*;*
- $\mathcal{W} = 0$.

Limitations

What we would like to do is find an implementation for falsification for SDCPN. Without using a transformation to GSHS, this seems to be too difficult. On the highest level, it is very difficult to find a global path in an SDCPN. In hybrid systems a global path is simply a series of connected states, in SDCPN a global path finder would have to make sure that different tokens get to desired positions at the same time to make transitions possible.

These problems are evaded if the SDCPN model is transformed to a GSHS model. In that case, the only minor difficulty in implementation is to determine how to make compact spaces of all the states spaces of the different locations. However, this could be done by hand. The GSHS foldout of an SDCPN can be rather large, which means that the tree that we create with the RRT search gets huge. So computational time is no longer an issue, but memory management might be a new one.

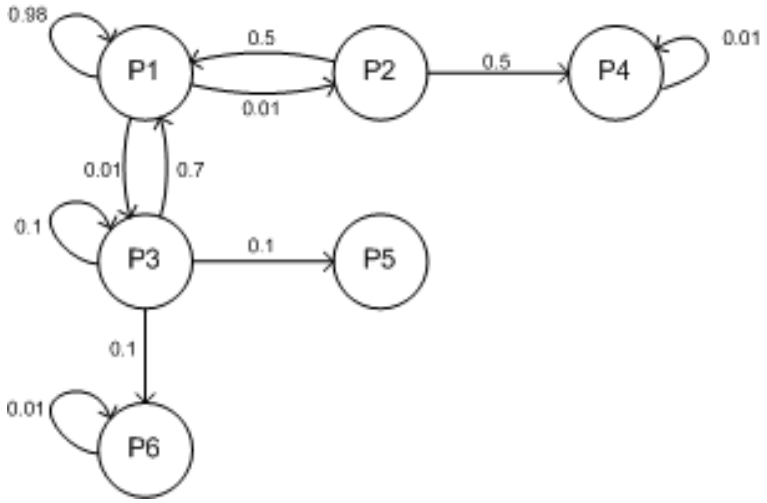


Figure 5.6: Discrete time Markov chain.

Opportunities

As seen under 'Limitations', there is no straightforward way to use falsification on SDCPN. For this reason we will focus on the other possibility, to first transform the SDCPN model into a GSHS model, and then apply falsification. In order to have a system that behaves like a hybrid automaton, we must negate the stochastic variance, i.e. set $\mathcal{W} = 0$. However, we can also see the stochastic variance as an input like that in controlled hybrid automata. There is a one-to-one correspondence between SDCPN and GSHS, as decribed in [Eve10]. The method as described in section 4.8.2 can then be used.

Example

To illustrate this technique we use the SDCPN model of figure 5.7. The initial state is $P1$ with $x = 0$ and $y = 0$. The first step of the falsification process is to create a GSHS, which would look like figure 5.8 (above left), unspecified. The initial state is denoted by a dot in the lower left corner of $P1$. The global path finder finds a path using breadth-first search like in FSM reachability analysis. The shortest path is from $P1$ to $P3$, so the local path finder will start searching for a path in $P1$. As the state space of this location in unbounded, we initially look at the state space of $P1$ within $x \in [0, 4], y \in [0, 4]$. In the next subfigure (above center) we see the first three random states picked from the state space and the tree that these random picks induce. The next subfigure shows the entire rapidly-exploring random tree search in $P1$. The local goal, a transition to $P3$ when $x \geq 2$ and $y \leq 1$ has not been reached. Therefore, a new global route is found, namely from $P1$ through $P2$ to $P3$. The local search is first performed in $P1$, where we can use the created tree and find a path to transition $G1$ to $P2$ immediately. The next subfigure (below left) shows the RRT search in P2, which is successful. Therefore, we have found a witness trajectory to $P3$. This trajectory is shown in the last subfigure (below right).

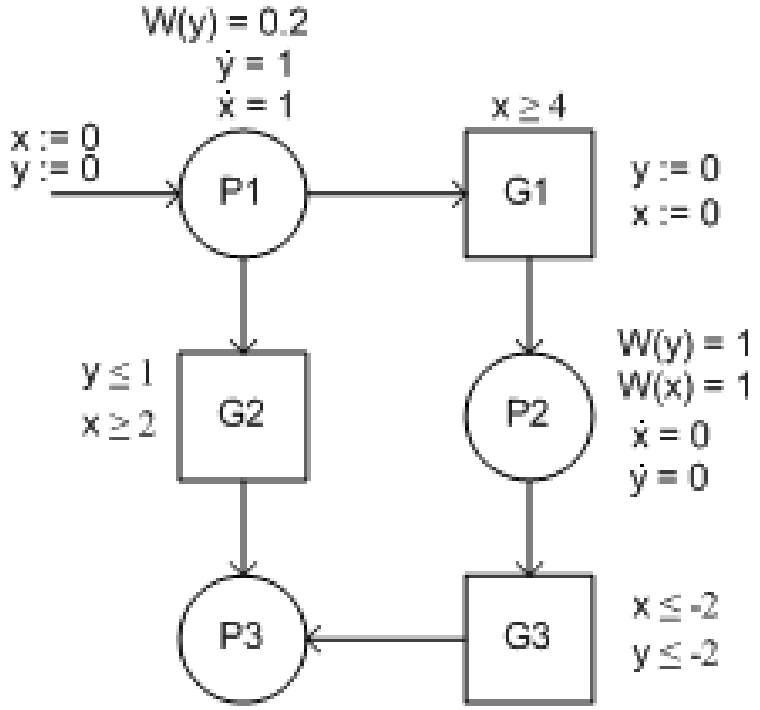


Figure 5.7: The SDCPN model we use to illustrate falsification.

5.9 Forward and Backward Reachability Analysis for LHA

Strengths

An important property of forward or backward reachability analysis is that if it terminates, it provides the correct answer, i.e. it does not approximate the answer like many other algorithms do. An algorithm could combine forward analysis from the initial states and backward analysis from the target states to have faster convergence. If the algorithm has not terminated yet after a bound on the computation time has been reached, this can be seen as an indication that the target states are not reachable, though it is no proof.

Equivalence. *An SDCPN model is language equivalent to a linear hybrid automaton if*

- *in each location the flows are linear;*
- *All guards are linear;*
- $\mathcal{W} = 0$*;*
- *The initial marking does not enable a transition;*
- *No transition immediately enables a guard transition;*
- *The number of tokens remains finite for* $t \to \infty$*;*
- *All firing measures in* $\mathcal{F}$ *are linear;*
- *the number of tokens remains finite for* $t \to \infty$*.*

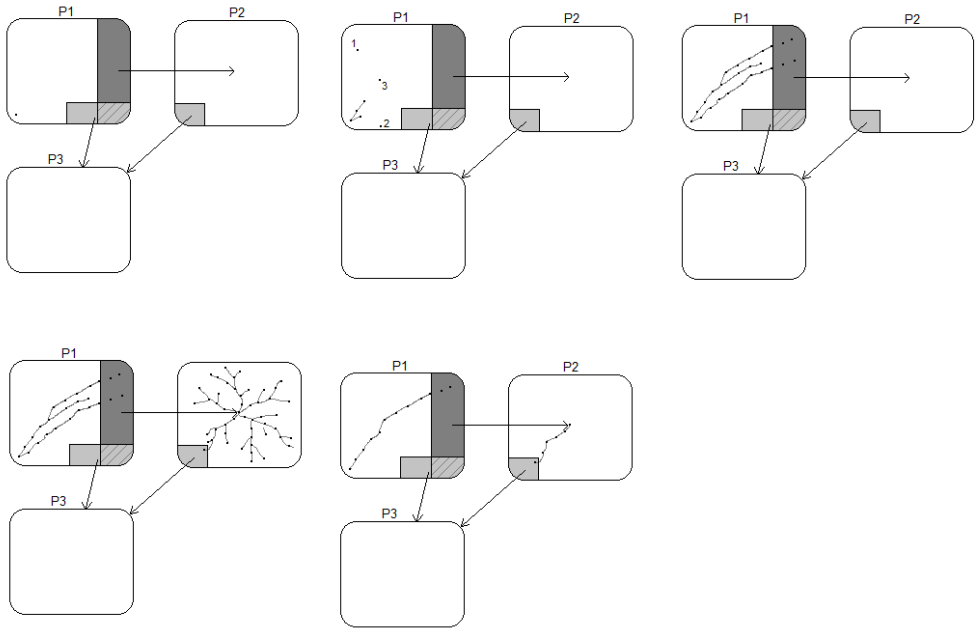


Figure 5.8: Five stages of the falsification method.

Note however that the linear hybrid automaton formalism does not provide a mechanism how runs are generated. Equivalence therefore means that for any SDCPN that fits the given restrictions one could build a linear hybrid automaton that allows the same range of paths as the SDCPN does.

Limitations

These algorithms are semi-decision procedures. This means that the computations do not necessarily converge so the algorithm might not terminate. This was to be expected, as the reachability problem is undecidable for linear systems. Moreover, if a system is complex the computations grow in complexity. It is difficult for a computer to test whether a set is contained in another set, so the number of sets that we get grows rapidly. This would cause many of these computations to exceed the time or memory limit. In most papers discussing these methods, examples solved with computational tools such as HyTech have very few discrete states.

Furthermore, the algorithm can only be performed on linear systems. This is a severe restriction to place on an entire model, but simple sub-systems of an SDCPN might show simpler dynamics, so this restricts the algorithm to simpler sub-systems of the entire model.

Opportunities

It is very difficult to calculate the forward or backward time closure and the postconditions of SDCPN models if they have multiple tokens. The literature has not focused on those situations, so we opt for the approach using a transformation to a system with only one token, i.e. based on the reachability graph. This is not necessary if we are looking at one LPN, which has only one token. Assume that a sub-system that we would like to analyze has only linear flows and guards. We can perform reachability analysis in the same way as was done in chapter 4.9.2. If the SDCPN model has stochastic flow (Wiener process) on the continuous evolution or nonlinear flow or guard, we would have to approximate the dynamics with linear dynamics. A series of methods that explain how to do this can be found in [HHWt96]. Once we have a linear hybrid automaton, we can use freely available hybrid model checkers, such as HyTech [HHWt97]. These model checkers make use of reachability analysis. The disadvantage is that these model checkers are meant for systems with very few nodes, and often come up empty for bigger models.

Example

The example will start from the SDCPN model in figure 5.9. We first change it to a linear hybrid system. Then, in each iteration of the forward reachability algorithm we compute all states reachable through continuous evolution and which transitions can be taken from the current set. We get

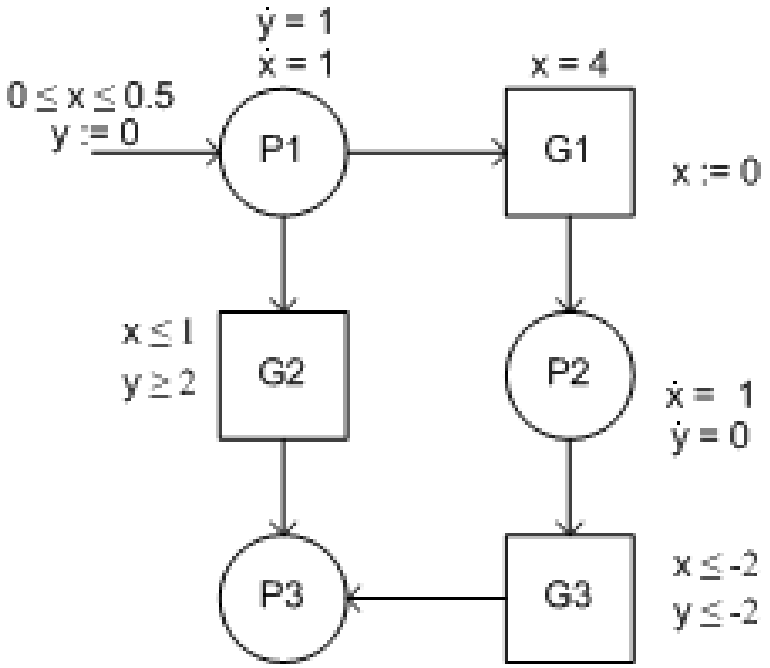


Figure 5.9: SDCPN model on which we perform forward reachability analysis.

$$
\begin{aligned}
I_1 &= \{y = 0 \cap 0 \leq x \leq 0.5 \cap P1\} \\
I_2 &= \{y \geq 0 \cap y \leq x \leq y + 0.5 \cap P1\} \\
I_3 &= \{(y \geq 3.5 \cap x = 0 \cap P2) \cup I_2\} \\
I_4 &= \{(y \geq 3.5 \cap x \geq 0 \cap P2) \cup I_2\} \\
I_5 &= \{I_4\}
\end{aligned}
$$

We see in the last line that the fixpoint has been found. As it does not include states in the target state $P3$, the system is safe. The fixpoint solution has been visualized in figure 5.10.

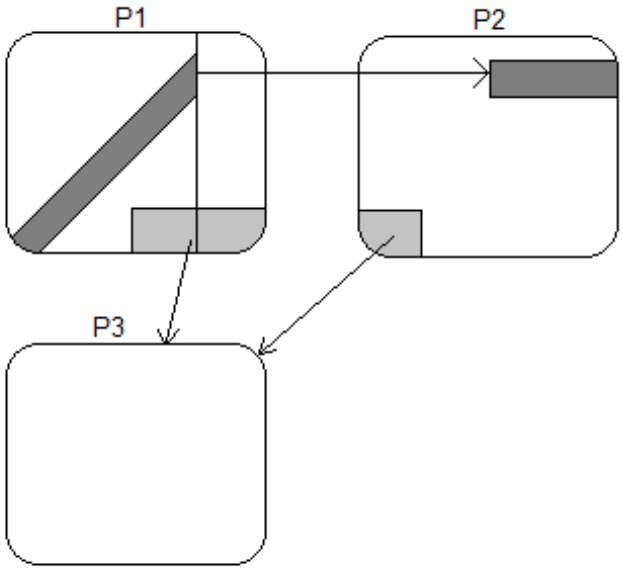


Figure 5.10: Visualization of a forward reachability computation. The dark gray set is I_5.

5.10 Minimization and Approximation for LHA

Strengths

Minimization and approximation techniques have been designed to overcome the limits of forward and backward reachability analysis. The first limit was computational complexity, because of which the procedure is slow. The polyhedra approximation technique transforms the reached sets into polyhedra, with which it is much easier to compute. A further method to speed up the reachability analysis is minimization. By bisimulating the model to an easier but equivalent model, the reachability analysis can be done more efficiently.

The termination problem is overcome by the widening technique. By consecutively removing constraints from polyhedral sets we force termination at some point, namely when the last constraint has been removed.

The restrictions under which SDCPN is equivalent to a linear hybrid automaton are given in the previous section.

Limitations

The cost of doing these approximations is that the computations are no longer exact. The solutions found by these methods are overapproximations, so a yes answer (we can reach the target state) is no longer proof of the target state's reachability, but rather evidence that it might be so. A no answer (the system is safe with respect to the target state) is still proof of the fact that the target region is not reachable.

Opportunities

These techniques are less accurate than pure reachability analysis. However, if forward and backward analysis fail to provide a conclusive answer, these techniques can be used to approximate the reachability question. There are no available online tools that use these techniques, so one would have to implement them oneself.

5.11 Barrier Certificates for SDP

Strengths

This method does not rely on the propagation of sets through the system dynamics. Instead, it searches for a function with certain characteristics. This is a big advantage in cases where the model contains difficult dynamics, through which it is difficult to propagate sets.

Equivalence. *An SDCPN model behaves equivalent to a switching diffusion process if*

- $\mathcal{T}_G = \emptyset$*;*
- *Every transition reset sets the token color of produced tokens to be equal to the token color of the token removed;*
- *The number of tokens remains finite for* $t \to \infty$*;*
- *The initial marking may not enable a transition.*

Limitations

This method is a semi-decision procedure; if no function can be found, that is no proof that the target set is reachable. On the other hand, if a barrier function is found, it does certify that the target set is not reachable.

Opportunities

This analysis can be used once an SDCPN has been converted to a GSHS model. Then, there are multiple methods of trying to find barrier certificates. One wants to find such certificate for each location that is apart of a global path between the initial and the target state, much like in falsification. A barrier certificate for a location separates the range of values with which that state can be entered from the range of values that allows the system to move to some other location. If we find barrier certificates for enough locations such that no global path can be taken to the target state, then the system is safe. It is also possible to search for barrier certificates which certify that a target is reached less than a given percentage of runs. This is especially helpful for stochastic systems such as SDP, because traditional barrier certificates are difficult to find. These probabilistic barrier certificates then allow the researcher to determine the likelihood of the target state to be reached.

Example

Assume that the SDP model created from an SDCPN model has a location in which the dynamics are equal to the example shown in figure 4.13. Assume furthermore that the location can be entered at the right solid patch, and that it can be exited only when the state is at the left solid patch. Then this location cannot be part of the global path leading from the initial set to the target set, as shown by the dashed barrier certificate.

5.12 Reach Probabilities for GSHS

Strengths

There are very minimal restrictions on SDCPN in order to use this method. It has already been used extensively to reduce the computation time of Monte Carlo simulation.

Equivalence. *An SDCPN-based model behaves probabilistically equivalent to a general stochastic hybrid system if*

- *The number of tokens remains finite for $t \to \infty$;*
- *The initial marking may not enable a transition;*
- *No transition may immediately enable a guard transition.*

Limitations

In large models a lot of data has to be stored at once, posing a high requirement on dynamic computer memory.

Opportunities

SDCPN inherits the strong Markov property of GSHS. This strong Markov property makes the analysis as outlined in section 4.11 possible. In order to use this opportunity then, we only have to use the method exactly as described in that section. Vital is the choice of stopping times, which in the case of collision risk modeling of air traffic tends to be the distance between two planes.

Example

For an example where reach factorization is used on an SDCPN-based model we refer the reader to [BBEVdP03].

5.13 Approximation to DTMC for DTSHS

Strengths

The main strength of DtSHS analysis when we consider its application to SDCPN, is the fact that one can approximate SDCPN to arbitrary degree with DTSHS, by making the gridding and timesteps snall enough.

Equivalence. *An SDCPN model never behaves exactly equivalent to a discrete-time stochastic hybrid system, because of the difference between continuous-time and discrete-time modeling. However, we can get arbitrarily close to the behavior of an DTSHS if*

- *The number of tokens remains finite for $t \to \infty$;*
- *The initial marking may not enable a transition;*
- *No transition may immediately enable a guard transition.*

Limitations

This analysis method becomes more accurate if more DTMC states are used, i.e. we cut up the state space into smaller cubes. This is subject to the state space explosion problem, meaning that computation time will be too great for models with too many states. Therefore there is a limit in accuracy that we can achieve, given the original complexity and size of the SDCPN.

Opportunities

SDCPN is not discrete time. Nonetheless, this analysis technique can be adapted to suit SDCPN models. Assume that we have an SDCPN model which has a local Petri net that is not influenced by other local Petri nets. We mean that it does have outgoing edges, but no incoming edges. Furthermore, assume that it does not have a form suitable for CTMC analysis, i.e. it has guard transitions and continuous dynamics. We could then discretize the LPN in a similar way as is done in DtSHS analysis.

We discretize over two factors, time and state space. The state space can be represented by one point if a place has dimension 0. Otherwise, the state space is always infinite. Each color is defined over $[-\infty, +\infty]$, so we would need an infinite number of points to cover this dimension. Therefore, the researcher should put reasonable boundaries in each dimension. Generally, the guards and resets of transitions give indications as to what boundaries preserve most paths. As a next step, the researcher should determine gridding sizes. A small gridding size, which results in many states for the DTMC, will be more accurate, but also more demanding in computational complexity. The gridding size may differ per parameter in each state space, which means that parameters more prone to errors can be gridded more tightly.

The last step is determining the transition probabilities between DTMC states. This must be done by looking at the flow, diffusion, transitions and resets of transitions. For each pair of

states from the DTMC, the researcher must determine the probability that within a single timestep, the token will go from one place to the other. This probability is both continuous evolution, looking at the flow and diffusion, and discrete evolution, looking at transitions and their resets. We will examine this method with an example.

Example

Assume we are dealing with the model described in chapter two and three, about a landing airplane. The LPN braking performance has no incoming arcs, relatively simple dynamics and a large influence on the simulation time. If the situation non-nominal braking is very rare, say one in ten thousand landings, it takes a long time before we see these events occurring in our simulations. If we would like to observe one thousand non-nominal landings, we would need to simulate ten million landings. This can be improved by doing CTMC analysis on LPN braking performance, reducing the simulation time of one thousand non-nominal landings to under one second. We now explore what would happen if LPN braking performance did not fit the CTMC requirements perfectly, but instead looked like figure 5.11.

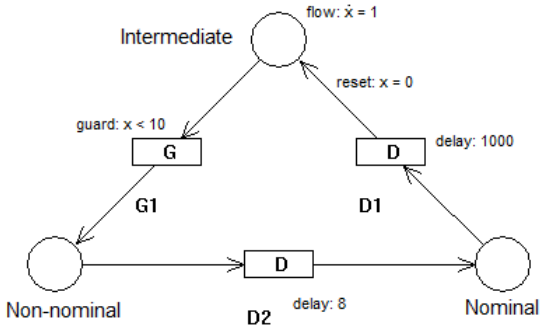


Figure 5.11: Example of an LPN that cannot be analyzed using CTMC analysis.

The steady state distribution of this LPN can be analytically found using the following method. First we determine the size of the relevant state spaces of each place. In places Nominal and Non-nominal, the state space is trivial, only one value is possible. The state space of place Intermediate could be set to $\{x : x \geq 0 \cap x \leq 10\}$. Next we decide on grid sizes. We could set time to be equal to 0.1 second and the grid on x to be 0.1. In this case the transition rates between states would become very simple, as it is certain that a token moves from one gridpoint to another in a timestep. We get a DTMC like in figure 5.12. This DTMC can be analyzed by DTMC steady state distribution analysis. Using MatLab we got that place Non-nominal and Intermediate are current about 0.01 percent of time, and place Nominal is current about 99.99 percent of time.

We can see that we have a much broader range of LPN that we can analytically determine the steady state distribution of than is the case for simple CTMC analysis. One limitation of this method is that it is difficult to determine the validity of the steady state distribution. We know that it is an approximation.

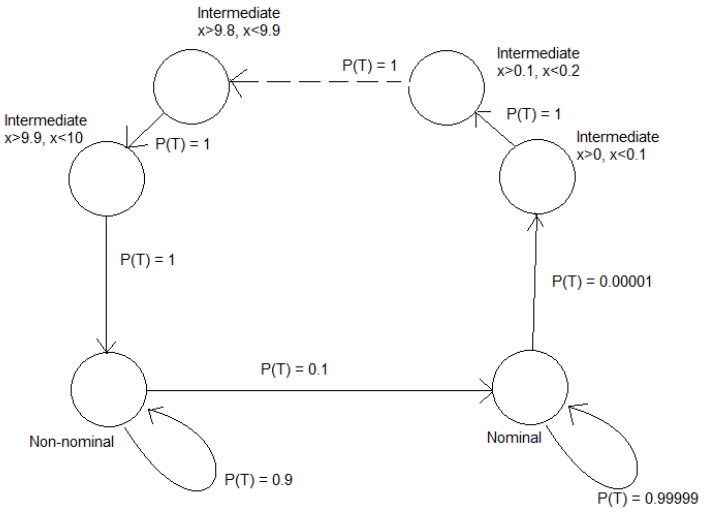


Figure 5.12: DTMC obtained from the LPN braking performance.

5.14 Bisimulation analysis for AFSM and CPDP

In this section we will discuss how bisimulation, as discussed in the previous chapter when dealing with AFSM and CPDP, can be applied to SDCPN.

Strengths

Bisimulation can be used to decrease the state space of a model. It can either reduce the continuous space of variables within a location, or decrease the number of locations in a model. This can be advantageous as a smaller model takes less time to analyze. There are no instances when bisimulation techniques cannot be used. Howevere, in many cases the model is already maximally bisimulated, and cannot be reduced in complexity any further.

Equivalence. *An SDCPN-based model is language equivalent to an arena of finite state machines if*

- $\forall P \in \mathcal{P} : \mathcal{C}(P) = \emptyset$;
- $\forall T \in \mathcal{T} : T \in \mathcal{T}_I$;
- *The number of tokens remains finite for* $t \to \infty$.

An SDCPN-based model behaves probabilistically equivalent to a communicating piecewise determinstic Markov chain if

- *The initial marking is determinstic;*
- *The initial marking does not enable a transition;*

- *None of the transition firings enable a guard transition;*
- *The number of tokens remains finite for* $t \to \infty$*;*
- $\mathcal{W} = 0$.

Limitations

Bisimulation might make an SDCPN model less clear, as it might blend two places into one which did have a different meaning to the developer. A great strength of SDCPN, that the model's structure resembles the structure of the system to be modeled, is thereby weakened.

Opportunities

We can define bisimulation relation of SDCPN-based models in a similar way as CPDP or AFSM bisimulation. We can also use the bisimulation algorithm presented in section 4.14.1. It is also thinkable that a bisimulation relation be defined for SDCPN including stochastic influence, i.e. not necessarily $\mathcal{W} = 0$. Two places could be stochastically equivalent if they, besides the other requirements described in section 4.14.1, have equal stochastic influence.

5.15 Overview

Table 5.1 gives an overview of the results of this chapter. We shortly discuss interesting features of this data.

The kind of equivalence one gets if the restrictions of column 3 are placed on SDCPN can differ between different systems. For example, when a restricted SDCPN behaves equivalently to an automaton, it does so only in the possible paths it allows. This is enough to apply the methods of automata on SDCPN.

In some cases, namely DTMC, DTSHS and SMTA, although it states equivalency this should be read as asymptotically equivalent. That is, we can get arbitrarily close to a restricted SDCPN. For example, GSHS and DTSHS have the same restrictions placed on SDCPN in order to be equivalent. In the case of GSHS this means that the systems are actually probabilistically equivalent. In the case of DTSHS the equivalency is an approximation. If timesteps are chosen very small, the SDCPN model can be approximated to an arbitrary degree of accuracy.

Not only the restrictions on SDCPN determine when to use which method, also the type of analysis problem one is dealing with is important. If one is dealing with a probablistic verification problem, i.e. what is the likelihood a target set is reached, methods based on automata are of limited use. One should look at methods based on GSHS, DTSHS, SDP and DTMC. Once a reseacher has determined which methods might be of interest, the third column of table 5.1 is useful for the next step. He should look at his SDCPN-based model and determine if it fits all the requirements for some method. If it does for one method, that is the method that can be used. If it does for multiple methods, the method that is most restrictive for SDCPN should generally be used. Methods that are more restrictive have stronger assumptions on the underlying system, and are therefore generally more adapted to give a suitable answer. For example, if an SDCPN-based model would fit the requirements of SMTA and LHA, methods based on SMTA should be used. It guarantees an answer and is much faster in finding that answer computationally than methods for LHA.

Formalism	**Method**	**Restrictions on SDCPN**
FSM	- Breadth first search (safety verification) - Model checking (verification)	- $\forall P \in \mathcal{P} : \mathcal{C}(P) = \emptyset$ - $\forall T \in \mathcal{T} : T \in \mathcal{T}_I$ - The number of tokens remains finite for $t \to \infty$
TA	- Abstract to FSM, then use forward analysis (safety verification) - Abstract to FSM, then use model checking (verification)	- $\mathcal{C}(P_i) = \mathcal{C}(P_j)\ \forall P_i, P_j \in \mathcal{P}$ - $\forall P \in \mathcal{P} : \mathcal{V}(P) = \{1, 1, ..., 1\}$ - The firing functions are deterministic, and set each token value either to 0 or to the value of the token removed - $\mathcal{W} = 0$ - Constants in guards are integers - The number of tokens remains finite for $t \to \infty$
SMTA	- Abstract to TA, then abstract to FSM, then use forward analysis (safety verification) - Abstract to TA, then abstract to FSM, then use model checking (verification)	- $\mathcal{C}(P_i) = \mathcal{C}(P_j)\ \forall P_i, P_j \in \mathcal{P}$ - The firing functions are deterministic, and set each token value either to 0 or to the value of the token removed - Between two states that have different boundaries on the changes of a variable x, x is reset to 0 along that edge - Guards can only check variables against constants, not against other variables - Constants in the guards are integers - The number of tokens remains finite for $t \to \infty$ - For any place, the flow $\mathcal{V}$ and diffusion terms $\mathcal{W}$ combined are positive for any valid valuation
CTMC	- Steady state analysis (steady state distribution)	- $\mathcal{C}(P) = \emptyset\ \forall P \in \mathcal{P}$ - The number of tokens remains finite for $t \to \infty$
DTMC	- Steady state analysis (steady state distribution) - Probabilistic reachability (probabilistic reachability)	- $\mathcal{C}(P) = \emptyset\ \forall P \in \mathcal{P}$ - The number of tokens remains finite for $t \to \infty$
LHA	- Forward analysis (safety verification) - Backward analysis (safety verification) - Approximate analysis (safety verification) - Minimization (safety verification)	- In each discrete state the flows are linear - All guards are linear - $\mathcal{W} = 0$ - All firing measures in $\mathcal{F}$ are linear - The initial marking does not enable a transition - No transition immediately enables a guard transition - The number of tokens remains finite for $t \to \infty$
HA	- RRT-based falsification (falsification)	- The initial marking does not enable a transition - No transition immediately enables a guard transition - The number of tokens remains finite for $t \to \infty$ - $\mathcal{W} = 0$

SDP	- Barrier certificates (probabilistic reachability)	- $\mathcal{T}_G = \emptyset$ - Every transition reset sets the token color of produced tokens to be equal to the token color of the token removed - The number of tokens remains finite for $t \to \infty$ - The initial marking may not enable a transition
GSHS	- Factorization of reach probabilities	- The number of tokens remains finite for $t \to \infty$ - The initial marking may not enable a transition - No transition may immediately enable a guard transition
DTSHS	- Approximate to DTMC, then use steady state analysis (steady state distribution) approximate to DTMC, then use probabilistic reachability (probabilistic reachability)	- The number of tokens remains finite for $t \to \infty$ - The initial marking may not enable a transition - No transition may immediately enable a guard transition
AFSM	- Bisimulation complexity reduction (complexity reduction)	- $\forall P \in \mathcal{P} : \mathcal{C}(P) = \emptyset$ - $\forall T \in \mathcal{T} : T \in \mathcal{T}_I$ - The number of tokens remains finite for $t \to \infty$
CPDP	- Bisimulation complexity reduction (complexity reduction)	- The initial marking is determinstic - The initial marking does not enable a transition - None of the transition firings enable a guard transition - The number of tokens remains finite for $t \to \infty$ - $\mathcal{W} = 0$

Table 5.1: Overview of the Applicability of Analysis Methods

Chapter 6

Conclusion

Many phenomena that are worthy of studying and simulating, air transport operations for example, comprise of discrete, continuous and stochastic elements. Stochastically and dynamically colored Petri net is a powerful modeling formalism for such systems. Improvements can be made in the arsenal of analysis techniques available for SDCPN. Researchers dealing with SDCPN are interested in the likelihood that systems enter risky areas, what percentage of time the system resides in risky states or the distribution of a given variable. The analysis techniques they use include Monte Carlo simulation and Markov chain analysis. This thesis identifies several ways to broaden that toolkit to include methods already known for similar, but less powerful modeling formalisms. Through investigating ways in which we can make these methods applicable to SDCPN, we come to the following recommendations.

Which analysis method is the best choice to use greatly depends on the type of analysis problem a researcher is facing. In general, we notice that methods based on automata tend to answer different kinds of questions than methods based on other hybrid systems. For example, RRT-based falsification could be a nice additional tool for SDCPN. It can aid a reseacher in finding events which rarely ever happen. However, the analysis problem it answers is falsification. Therefore it finds paths, without knowing the likelihood such path is taken. If we were able to use rapidly-exploring random trees to answer the probabilistic verification problem, this would be greatly useful. This remains an open problem.

We have distinguished the different analysis problems for a reason. A reseacher can identify with which analysis problem he is dealing, and then look at methods that deal with the same problem. For each of these methods he should determine whether the SDCPN-based model satisfies the restrictions as outlined in chapter 5. We recommend he start with the methods that require the tightest restrictions. Because the system is more restricted, the method more ably provides an answer. In general, however, the tighter restrictions will be too tight for an SDCPN-based model to satisfy.

Some of the more restrictive hybrid systems offer methods which might be used in analyzing errors. Modeling errors in SDCPN, such as accidental arc placement or removal, might be detected by analyzing the underlying graph through model checking or safety verification of finite state machines. What kinds of errors can be traced by using model checking and similar methods is a topic for further research.

Automated analysis is an interesting research topic. Many analysis techniques are difficult to do by hand, but could be done by computers. At this moment most of the computing

tools (such as HyTech for forward search of linear hybrid automata, SOSTOOLS for barrier certificates of switching diffusion processes) are of limited practical use as they only work on restricted models, generally small models that include straight-forward dynamics. Although headway is being made to speed up these tools and use new techniques, it remains an open problem to make these tools truly applicable to practical real-life models.

Further work needs to be done concerning the accuracy loss of approximation techniques. If one would use DTSHS approximation to DTMC for SDCPN, some accuracy is always lost. However, one has to choose a timestep and a gridding factor, which influence that accuracy loss. It would be interesting to know how significant the accurcay loss is for a series of SDCPN models and a range of timestep values and gridding factors.

Lastly, this thesis can be seen as an inventory of current existing methods. Many methods are relatively new, and it is to be expected that some headway can still be made. Algorithms might be improved and sped up, or perhaps new abstractions might be found that allow methods to be used on systems that are different from the systems for which they are used today. It is therefore recommended to check up on new research every couple of years.

Bibliography

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[Adl01] M. Adler. *An Introduction to Mathematical Modelling*. HeavenForBooks.com, 2001.

[AEN12] AENA and DFS-NLR and NATMIG. Initial guidelines for dynamic risk modelling application. SESAR 16.1.3 project Deliverable D07, November 2012.

[Ajm89] M. Ajmone Marsan. Stochastic Petri nets: An elementary introduction. In *Advances in Petri Nets*, pages 1–29. Springer, 1989.

[AKLP10] A. Abate, J.-P. Katoen, J. Lygeros, and M. Prandini. Approximate model checking of stochastic hybrid systems. *Eur. J. Control*, 16(6):624–641, 2010.

[BBEVdP03] H.A.P. Blom, G.J. Bakker, M.H.C. Everdij, and M.N.J. Van der Park. Collision risk modeling of air traffic. In *Proceedings of European Control Conference*, 2003.

[BBK+05] H.A.P. Blom, G.J. Bakker, J. Krystul, M.H.C. Everdij, B. Klein Obbink, and M.B. Klompstra. Sequential monte carlo simulation of collision risk in free flight air traffic. *Hybridge Report D*, 2005.

[BBK09] H.A.P. Blom, G.J. Bakker, and J. Krystul. Rare event estimation for a large-scale stochastic hybrid system with air traffic application. *Rare Event Simulation using Monte Carlo Methods*, page 193, 2009.

[BK08] C. Baier and J. Katoen. *Principles of Model Checking*. MIT press, 2008.

[BL04] M.L. Bujorianu and J. Lygeros. General stochastic hybrid systems. *Proceedings 12th IEEE Mediteranean conference on control and automation (MED)*, 2004.

[BL06] M.L. Bujorianu and J. Lygeros. Toward a general theory of stochastic hybrid systems. *Lecture Notes in Control and Information Sciences (LNCIS)*, 337:3–30, 2006.

[BM06] D. Bader and K. Madduri. Designing multithreaded algorithms for breadth-first search and st-connectivity on the Cray MTA-2. In *ICPP '06: Proceedings of the 2006 International Conference on Parallel Processing*, pages 523–530, Washington, DC, USA, 2006. IEEE Computer Society.

[BY04] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, pages 87–124. Springer, 2004.

[CCQ96] G. Cabodi, P. Camurati, and S. Quer. Improved reachability analysis of large finite state machines. In *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pages 354–360, 1996.

[Dav84] M.H.A. Davis. Piecewise deterministic Markov processes: a general class of non-diffusion stochastic models. *Journal Royal Statistical Society (B), 46(3)*, pages 353–388, 1984.

[Dav93] M.H.A. Davis. *Markov models and optimization*, volume 49 of *Monographs on statistics and applied probability*. Chapman and Hall, London, 1993.

[Daw01] C. Daws. Symbolic and parametric model checking of discrete-time Markov chains. 2001.

[DY95] C. Daws and S. Yovine. Two examples of verification of multirate timed automata with kronos. In *Proc. 1995 IEEE Real-Time Systems Symposium, RTSS'95*, pages 66–75. IEEE Computer Society Press, 1995.

[Eve10] M.H.C. Everdij. *Compositional Modelling Using Petri Nets with the Analysis Power of Stochastic Hybrid Processes*. PhD thesis, Universiteit Twente, 2010.

[HHWt96] T.A. Henzinger, P.-H. Ho, and H. Wong-toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:225–238, 1996.

[HHWt97] T.A. Henzinger, P.-H. Ho, and H. Wong-toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:460–463, 1997.

[HLS00] J. Hu, J. Lygeros, and S. Sastry. Towards a theory of stochastic hybrid systems. *N. Lynch and B. Krogh (Eds.): HSCC 2000, LNCS 1790*, 2000.

[KV00] A.B. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In *Hybrid Systems: Computation and Control*, pages 202–214. Springer, 2000.

[Lav98] S.M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[LPY99] G. Lafferriere, G.J. Pappas, and S. Yovine. A new class of decidable hybrid systems. In *Hybrid Systems : Computation and Control*, pages 137–151. Springer, 1999.

[MS96] O. Müller and T. Stauner. Modelling and verification using linear hybrid automata – a case study, 1996.

[nlr] NLR website. http://www.nlr.nl/who-we-are/index.html. Accessed: 14/12/2012.

[Pap03] George J Pappas. Bisimilar linear systems. *Automatica*, 39(12):2035–2047, 2003.

[PBS11] G. Pola, M. D. Di Benedetto, and E. De Santis. Arenas of finite state machines. *CoRR*, abs/1106.0342, 2011.

[PDBDS11] G. Pola, M.D. Di Benedetto, and E. De Santis. A compositional approach to bisimulation of arenas of finite state machines. In *18th IFAC World Congress, Milan*, 2011.

[Pet62] C.A. Petri. *Communication with Automata*. PhD dissertation, Darmstadt University of Technology, 1962.

[PH07] M. Prandini and J. Hu. A numerical approximation scheme for reachability analysis of stochastic hybrid systems with state-dependent switching. pages 4662–4667. New Orleans, LA, 2007.

[PJ04] S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *Hybrid Systems: Computation and Control*, pages 477–492. Springer, 2004.

[PJP07] S. Prajna, A. Jadbabaie, and G.J. Pappas. A framework for worst-case and stochastic safety verification using barrier certificates. *IEEE Transactions on Automatic Control*, 52(8):1415–1429, 2007.

[PKV09] E. Plaku, L. Kavraki, and M. Vardi. Hybrid systems: from verification to falsification by combining motion planning and discrete search. *Formal Methods in System Design*, 2009.

[PPBS12] A. Petriccone, G. Pola, M.D. Di Benedetto, and E. De Santis. Safety criticality analysis of complex air traffic management systems via compositional bisimulation. *4th IFAC conference on Analysis and Design of Hybrid Systems*, 2012.

[PPP02] S. Prajna, A. Papachristodoulou, and P.A. Parrilo. Introducing SOSTOOLS: A general purpose sum of squares programming solver. In *CDC*, pages 741–746, 2002.

[Roo12] N. Roohi. Decidability for the reachability problem in initialized almost rectangular hybrid automata. 2012.

[SS99] A. van der Schaft and H. Schumacher. *An Introduction to Hybrid Dynamical Systems*. 1999.

[Str05] S.N. Strubbe. *Compositional Modelling of Stochastic Hybrid Systems*. PhD thesis, University of Twente, The Netherlands, December 2005.

[SVDS05a] S. Strubbe and A. Van Der Schaft. Bisimulation for communicating piecewise deterministic Markov processes (cpdps). In *Hybrid Systems: Computation and Control*, pages 623–639. Springer, 2005.

[SvdS05b] Stefan Strubbe and Arjan van der Schaft. Algorithmic bisimulation for communicating piecewise deterministic Markov processes. In *44th IEEE Conference on Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05.*, pages 6109–6114. IEEE, 2005.