The development of autonomous agents, such as mobile robots or software agents has generated considerable research in recent years. Robotic systems, which are usually built from a mixture of continuous (analog) and discrete (digital) components, are often referred to as hybrid dynamical systems.

The modeling and analysis of hybrid dynamical systems is becoming more and more important as such systems are now widely used to reason about complex physical systems. Ying Zhang and Alan Mackworth developed a semantic model for dynamical systems, called Constraint Nets (CN) [ZM95a]. CN introduces an abstraction and unitary framework to model hybrid systems. Furthermore, specification and verification methods were introduced for deterministic system.

Traditional approaches to real-time hybrid systems usually define behaviors purely in terms of determinism or sometimes non-determinism. The CN framework was developed to model and verify deterministic systems, with the capability to model non-determinism. However, real-time dynamical systems very often behave unpredictably and thus exhibit (structured) uncertainty. It is therefore important to be able to model and analyze real-time probabilistic systems. Hence, a formal framework to model systems with unpredictable behaviors is essential.

We extend the work previously done on Constraint Nets by developing a new framework that we call "Probabilistic Constraint Nets" (PCN). The PCN framework allows for the modeling and simulation of any dynamical system, whether it is deterministic, non-deterministic or probabilistic. We introduce formal syntax and semantics for the framework that ensure the correctness of the models. We also provide a graphical representation that simplifies the task of modeling complex systems. Moreover, we show that our framework is a generalization of many commonly used frameworks such as Markov processes and Markov Decision Processes (MDP). This allows the user to take advantage of a unified framework encompassing most popular modeling paradigms. We have also developed two specification languages (average-time timed $\forall$-automata and PATTL) along with verification algorithms that allow us specify some behavioural constraints on the system and enables us to proceed to on average and to probabilistic verification of these requirements.

Finally, we also provide, for a subclass of PCN models algorithms for control synthesis. Moreover, we investigate the use of stochastic and robust control for handling the control synthesis task within PCN. With such control synthesis techniques, a designer can automatically construct an optimal controller for his system, hence greatly facilitating his task.
## Contents

Abstract ........................................... ii

Contents ....................................... iii

List of Tables .................................... vii

List of Figures .................................... viii

Acknowledgements ................................ xi

Dedication ...................................... xii

1 Introduction and Motivation .................. 1

1.1 Considered Problems .......................... 3

1.2 Proposed Solutions to the Considered Problems .... 10

1.3 Related Work ................................ 14

1.3.1 Review of Constraint Nets ................. 14

1.3.2 Modeling Probabilistic Systems .......... 19

1.3.3 Requirement Specifications and Verification Techniques .... 21

2 Measure-Theoretical and Topological Structure of Dynamics .... 26

2.1 General Topology, Partial Order, Metric Space and Measure Theory ...... 26

2.1.1 Partial order ................................ 30

2.1.2 Metric space .............................. 33

2.1.3 Random Variables ......................... 36

2.1.4 White Noise, Brownian Motion and Stochastic Integrals .............. 40

2.2 Time Structures .............................. 43

2.3 Domain Structures ........................... 46

2.4 Stochastic Traces and Events ................. 49

2.5 Transductions ............................... 53

2.5.1 Transductions: general concepts ........... 53

2.5.2 Primitive transductions .................... 54
2.5.3 Event-driven transductions ........................................ 57
2.6 Dynamics Structures .................................................... 58

3 Probabilistic Constraint Nets ........................................... 62
  3.1 Probabilistic Constraint Nets ......................................... 62
    3.1.1 Syntax of Probabilistic Constraint Nets ....................... 63
    3.1.2 Semantics of Probabilistic Constraint Nets ................... 69
    3.1.3 Fixpoint in distribution of partial orders .................... 73
    3.1.4 Semantics of Probabilistic Constraint Nets ................... 74
    3.1.5 Semantics of Modules ............................................ 75
    3.1.6 Family of Probabilistic Constraint Nets ....................... 79
    3.1.7 Stochastic temporal integration ............................... 81

4 Modeling in PCN .......................................................... 86
  4.1 Events ................................................................. 86
    4.1.1 Event generators ................................................ 87
    4.1.2 Event synchronizers ............................................. 90
  4.2 Types of Uncertainties ............................................. 90
  4.3 Computation in Probabilistic Constraint Nets ..................... 91
    4.3.1 Sequential and Analog computation ............................ 91
    4.3.2 Stochastic Taylor expansion .................................... 92

5 Models Subsumed by PCN ............................................... 97
  5.1 Discrete time and discrete domain systems ....................... 97
    5.1.1 DTMC to PCN conversion ....................................... 97
    5.1.2 DD-PCN equivalence to DTMC .................................. 98
  5.2 Discrete time and continuous domain systems ..................... 104
    5.2.1 DTMP to PCN conversion ....................................... 104
    5.2.2 DC-PCN equivalence to DTMP .................................. 104
  5.3 Continuous time and discrete domain systems ..................... 105
    5.3.1 CMTC to PCN conversion ....................................... 107
    5.3.2 PCN to CTMC ................................................... 107
  5.4 Continuous time, continuous domain ................................ 109
  5.5 Other models of interest .......................................... 110

6 Introduction to Behavioural Verification ............................ 111
  6.1 Average-Timed V-Automaton ........................................ 112
  6.2 Our Probabilistic Temporal Logic: PATTL .......................... 112
  6.3 Behavioural Verification ........................................... 113
7 Behavioural Verification with Average Timed $\forall$-Automaton
7.1 $\forall$-Automata
7.1.1 Average-timed automata
7.2 Model-Checking Approach
7.2.1 Behavioural Constraint Verification Rules for Discrete-time
7.2.2 Automatic Behaviour Verification
7.3 Generalizing the Approach

8 Behavioural Verification with a Probabilistic Temporal Logic
8.1 Introduction to CTL and PCTL
8.2 Probabilistic Arbitrary-time Temporal Logic (PATL)
8.2.1 Common Logical Connectives and Temporal Operators
8.3 Probabilistic Arbitrary-time Timed Temporal Logic (PATTL)
8.4 Time Structure and Verification Issues
8.4.1 The Structure of Time
8.5 Model Checking of PATTL over finite PCN and non-continuous time structure
8.5.1 Verifying modal formulae in PATTL
8.5.2 Model checking of the $X$ operator
8.5.3 Model checking of the $U^\tau$ operator when $\tau = 0$
8.5.4 Model checking of the $U^\tau$ operator when $0 < \tau < \infty$
8.5.5 Model checking of the $U^\tau$ operator when $\tau = \infty$
8.5.6 Model checking of the $S^\tau$ operator
8.6 Model checking of PATTL formulae with continuous time structures
8.7 First Order PCN timed temporal logic (FPATTL)

9 Control Synthesis
9.1 Introduction
9.2 Stochastic and Robust Control
9.2.1 Robust Control of Package Delivery Robot
9.3 Planning under uncertainty
9.4 Introduction to Markov Decision Processes
9.5 MPDs correspondence to PCN
9.5.1 MDP to PCN Conversion
9.5.2 PCN to MDP conversion
9.6 Introduction to Partially Observable MDPs
9.7 POMDPs correspondence to PCN

10 Problems, Solutions and Contributions
10.1 Summary of Contributions
10.2 Future Work
10.2.1 Further Desired Theoretical Development
10.2.2 Further Practical Applications
List of Tables

4.1 Generic Types of probabilistic/stochastic models ........................................ 86
8.1 Comparing the diverse cases ................................................................. 152
List of Figures

1.1 The structure of a constraint-based agent system ........................................ 4
1.2 a) Dynamics of a mobile car with uncertain actuators; b) PCN model of Equation 1.1 6
1.3 Autonomous robot chasing a series of targets. Various effects of uncertainty in the odometry. ......................................................... 7
1.4 The problems and proposed solutions .......................................................... 10
1.5 Transduction Type Hierarchy ........................................................................ 12
1.6 The constraint net representing $\dot{s} = f(s)$ ................................................. 16

2.1 An event trace: each dot depicts a time point ................................................ 52
2.2 Event logic for “or” ...................................................................................... 56
2.3 Simple Transduction with uncertainty .......................................................... 57

3.1 Gaussian probability distribution as a Generator and random location. .......... 65
3.2 The probabilistic constraint net representing a Markov model. ....................... 65
3.3 PCN model of an Itô process. ......................................................................... 66
3.4 Combined Operations: cascade, parallel and feedback. ................................. 68
3.5 PCN model of an input/output probabilistic automaton (state denotes either $s'$ or $s$). 68
3.6 Simple PCN for a probabilistic sum. .............................................................. 70
3.8 Density of $dX = -X(X - 1)(X - 2)dt + dB_t$. ............................................. 72
3.9 Sample path of the system in Figure 2.3 from page 57. ................................. 75
3.10 Empirical Distribution of Example 2.3. ....................................................... 76
3.11 Evolution of the distributions of $f(x)$. ........................................................ 77

4.1 Two basic modules for event logics: NotEqual and F. ................................... 88
4.2 A sequential module ..................................................................................... 91

5.1 Representation of a DTMC as a PCN .......................................................... 98
5.2 Simple DD-PCN with three locations ............................................................ 100
5.3 Bayesian Network representation of the temporal dependence ....................... 102
5.4 Non-Unit delays for the XYZ DD-PCN ....................................................... 103
5.5 Representation of a DTMP as a PCN ............................................................ 104
5.6 PCN equivalent of a CTMC ......................................................................... 107
C.2 Value for the van PCN ............................................. 264
C.3 Value function for the cat mouse PCN ................................. 272
Acknowledgements

First and foremost, I would like to thank the members of my supervisory committee: Alan Mackworth, Joel Friedman, David Poole, Alan Hu, Marty Puterman, Mark Greenstreet, Paul Gustafson, Dinesh Pai and Harry Joe (chair). Without their contribution and direction, this thesis would not have been what it is now.

I am incredibly grateful that I had the opportunity to work with Alan Mackworth as my thesis supervisor and research collaborator. I could not have asked for a better supervisor. Alan’s unparalleled insight and perspective inspired me to tackle stimulating new research problems. His continued support (intellectual, emotional and financial) and trust in my abilities also greatly contributed to the success of this thesis.

Joel’s contribution to this thesis is incalculable. Without his direction and knowledge in the field of mathematics and topology, this work would not have been possible.

I would also like to thank David for many challenging discussions which kept me on my toes and forced me to reconsider many concepts and to see them under a different light.

Many thanks go to Craig Boutilier, my first research supervisor in LCI. Craig believed in me when I had no experience in the field and he gave me the opportunity to enter the challenging world of computer science and artificial intelligence. He proved to be a great supervisor and friend.

Valerie McRae, the LCI secretary acted like a surrogate mother throughout the duration of my studies. She was always there for me, providing help and supportive comments on a regular basis. I am very grateful to her.

I thank my family and close friends (you know who you are), for their wishes, belief, support and understanding during what proved to be a longer than expected process.

Finally, I thank the University of British Columbia for the Graduate Student Fellowship, NSERC for the Post Graduate Fellowships, FCAR for the Doctoral Fellowship and PRECARN for the scholarships they granted me over the course of my academic career. Financial security throughout my graduate studies provided me with peace of mind that allowed me to devoted my time to my research and I am very thankful for that.

ROBERT JR. ST-AUBIN

The University of British Columbia
June 2005
Je dédie cette thèse à ma famille et à mes amis qui m'ont offert tout le support dont j'aurais pu espérer et même plus.
Chapter 1

Introduction and Motivation

Computer-controlled robotic systems are becoming ubiquitous. Until recently, computer controls were mainly used in large industrial applications such as nuclear and chemical plants, assembly lines and forest industries. However, now such systems are everywhere, whether it is in our cars, our refrigerators or in our children's toys. As a larger portion of the population is exposed to these systems, there is a growing need for the design of intelligent, robust, reliable and safe robotic systems.

In this body of work, we are interested in the modeling and verification of robotic systems where the inherent uncertainty is explicitly modeled. A robotic system is a dynamical system which can be extremely complex. No tractable model can represent its dynamics and environment perfectly. Due to the limitations in modeling and sensing of such systems, they exhibit uncertainty and very often behave probabilistically. It is therefore important to be able to model and analyze real-time systems while taking into consideration the underlying uncertainty. To address this question, we have developed Probabilistic Constraint Nets (PCN): a framework for the modeling and analysis of systems exhibiting uncertainty. However, before getting into the thick of the subject, it is important that we clarify what we actually mean by the term uncertainty and to define how we intend to model it.

It is now well established that probability theory is a body of knowledge (although not the only one) that enables one to reason formally about uncertain events. Many different approaches on probability theory have arisen over time: the approaches that are most discussed in the literature are the objective (classical, frequentist) and subjective (Bayesian) approaches.
The key difference between the Bayesian and frequentist approach is in how probability is viewed and assessed. The frequentist approach views probability as a proportion: the probability of a coin landing heads-up is 0.5 because in a long series of coin tosses it lands heads-up half the time. On the other hand, the Bayesian approach views probability as a subjective measure which reflects personal belief. Somebody adopting a Bayesian approach might therefore decide that the probability of a coin landing heads-up is 0.5 because the evidence combined with their belief leads them to believe that the chance of obtaining a head or tail is equal.

Bayes' ideas emerged at the same time as frequentist methods were being advanced. However, Bayesian statistics did not develop in parallel with traditional frequentist techniques because the application of these methods is complex and was until recently intractable due to the sheer number of computations often required. The application of Bayesian techniques is therefore relatively new and is growing as a consequence of advances in modern computers and the development of appropriate statistical software. Moreover, Bayesian techniques are gaining momentum in the scientific community as researchers agree more and more on the observation that subjective probability is a natural concept developed by the human mind to quantify the plausibility of events in conditions of uncertainty. One could argue that Bayes' theorem is in fact a natural way of reasoning in updating probability.

There has been, and still is, disagreement between the frequentist and Bayesian community. This work does not intend on providing evidence on the "right" approach. Instead, we simply adopt the subjectivist (Bayesian) position regarding the interpretation of probability, which we see as more natural. The reader is referred to [BS94] for a thorough introduction to Bayesian theory.

Throughout this dissertation, we will also assume deterministic causality within the world. Hence the uncertainty in the systems of interest can be viewed as arising from the observer's ignorance or his inability to derive accurate models. The probability distributions that will be part of our models will then reflect the observer's belief of the system behaviour. For example, one might try to model a certain system function \( f(\cdot) \) but due to incomplete information, she might only be able to observe and reason about the function: \( g(\cdot) = f(\cdot) + \) "noise". This user might have a certain belief about the nature of the noise, and thus specify a certain probability distribution on it.

Furthermore, we will express causal relationships in the form of deterministic, functional equations, and introduce probabilities through the assumption that certain variables in these equations
are unobserved (or random). This reflects Laplace's conception of natural phenomena, according to which nature's laws are deterministic and randomness surfaces due merely to our ignorance of the underlying boundary conditions. It contrasts with the modern (quantum mechanical) conception of physics, according to which all of nature's laws are inherently probabilistic and determinism is but a convenient approximation. However, it seems to be widely accepted within the scientific community that for macro systems, physical laws are deterministic. Quantum mechanics applies to micro-systems, systems which we shall not focus on in this dissertation. We are mainly interested in robotic systems, and we will assume that the laws of physics or mechanics that they follow are deterministic.

Let us now conclude this discussion with an interesting passage on the defense of the subjective position, as given in de Finetti's *Theory of Probability*, [dF74]:

> The only relevant thing is uncertainty - the extent of our own knowledge and ignorance. The actual of whether or not the events considered are in some sense determined, or known by other people, and so on, is of no consequence.

> The numerous, different, opposed attempts to put forward particular points of view which, in the opinion of their supporters, would endow Probability Theory with a "nobler" status, or a "more scientific" character, or "firmer" philosophical or logical foundations, have only served to generate confusion and obscurity, and to provoke well-known polemics and disagreements - even between supporters of essentially the same framework.

*(de Finetti, 1970/1974, Preface, xi-xii)*

### 1.1 Considered Problems

From a systemic point of view, we view a robotic system as the coupling of a robotic agent (or simply a robot or an agent) to its environment. Obviously, the most common agents are robots. However, our framework applies equally to software agents, embedded devices or any animate agents in this world. Formally, an agent consists of the coupling of two distinct *constraint-based* modules: a *body* which usually encompasses the various sensors and actuators, and a *controller*, which is the (mostly) software portion that controls the behaviour of the agent. We view constraint satisfaction
as a dynamical process that approaches the solution set of the given constraints asymptotically; where a constraint is a (possibly) implicit relation on a set of variables. Generalizing, we view a constraint-based dynamical system as a dynamical system that approaches the solution set of the given constraints persistently. We will discuss this topic more in depth in the upcoming chapters.

With its sensors, the agent’s body senses the environment, and reports to the controller on the perceived state of the environment. In turn, the controller, now equipped with an updated report on the state of the environment, sends appropriate control signals to the actuators of the body to perform the required actions which change the state of the world.

The corresponding coupled relationship between a robotic agent and its environment is shown in details in Figure 1.1. From this figure, one can see how the coupled agent and environment act on, and react to, each other in a closed-loop system evolving over time. It is important to notice how the system is affected at different levels by the various types of uncertainty, for example, originating from external disturbances, sensor noise and uncertainty in the dynamics. Hence, to design proper robotic systems, one needs to solve the constraint satisfaction problem resulting from joining the knowledge about the various types of uncertainty with the constraints-based modules composing the agent.

The notions of constraint programming and of the Constraint Satisfaction Problem (CSP) are important paradigms that have been studied extensively. Typically, CSP’s provide problem solvers,
which are, more often then not, off-line solvers. Despite the advances in the field of CSP, one area that has yet to be explored thoroughly is the problem of designing dynamical systems in a constraint-based fashion. Dynamical systems should not be handled in an off-line approach but rather should be seen as online constraint satisfying systems. One proposed solution, Constraint Net (CN), was developed for building deterministic hybrid intelligent systems as situated agents [ZM95a]. In CN, a deterministic dynamical system is modeled, in a unitary way, by the coupling of an agent with its environment.

In order to handle embedded systems in an efficient way, we must move beyond the typical offline constraint satisfaction model and adopt a model where the solution to a constraint problem is a temporal trace of values, obtained via a non-anticipative transformational process, a transduction, of the input trace over time. Furthermore, the hybrid nature of most embedded systems necessitates that computations be performed on various time structures such as discrete, continuous or event-based. In addition, the computations should be performed while taking account of the uncertainty present in the system under study. In this thesis, we show how the PCN framework is suited for modeling and reasoning about such systems.

Before going any further, let us demonstrate key concepts by introducing an example based on a testbed for radio-controlled soccer playing car-like robots built at the Laboratory for Computational Intelligence (LCI) at the University of British Columbia. For each robot in the testbed, the controller is equipped with both digital and analog devices, hence making it a good example of an hybrid system [SM94]. This example is the stochastic version of the central example presented in the initial work on Constraint Net [Zha94], the framework which we build on.

In the original testbed, the robots could move forward and backward with a throttle setting, and were able to make turns by steering their two front wheels. The robots were incapable of moving sideways and the turning radius was restricted by mechanical stops in the steering gear. Zhang and Mackworth modeled these robots as a deterministic system. Here we extend this model to include noise in the actuators, hence introducing uncertainty in the dynamics of the robots.

We display, in Figure 1.2a), the configuration of one of the soccer playing robot-cars. In this model, \( v \) and \( \alpha \) respectively represent the velocity of the car and the current steering angle of the front wheels. At this level of modeling, one can see \( v \) and \( \alpha \) as control inputs of the system. Extending the deterministic dynamics presented in [Lat91, Zha94], the stochastic dynamics of the
The car can be modeled by the following stochastic differential equations:

$$\begin{align*}
\dot{x} &= v \cos(\theta) + W^x_t, \\
\dot{y} &= v \sin(\theta) + W^y_t, \\
\dot{\theta} &= \frac{v}{R} + W^\theta_t
\end{align*}$$

(1.1)

where \((x, y)\) is the coordinate of the position of the tail of the car, \(\theta\) is the heading direction of the car, \(R = L / \tan(\alpha)\) is turning radius given the length of the car \(L\) and \(W^i_t, i \in \{x, y, \theta\}\) is a one-dimensional "white noise"\(^1\) affecting the location of the car along with its heading angle.

So far, we have not yet directly addressed the importance of explicitly modeling a system's uncertainty. However, many factors, such as industrial production variations, incomplete information or even numerical precision errors, lead to the conclusion that uncertainty is inherently present in any real physical system. Failing to model the effects of these various sources of uncertainty can have a drastic influence on the modeled system's behaviour. Although the uncertainty model might not mimic exactly the nature of the uncertainty within the system under study, completely ignoring it in the modeling task can be even more disastrous. For instance, when designing an airplane controller, it is crucial to take the uncertainty of the wind speed into account, since a given controller might be optimal for a certain wind condition but might behave dangerously for other conditions.

\(^1\)We will formally discuss the notion of white noise when we introduce the concept of Brownian motion in § 2.1.4.
As another example, consider a simulation of the behaviour of an autonomous robotic system traveling to a series of goal locations, as shown in Figure 1.3 (goals are shown as \( \bigcirc \) in the figure). In the situation where the uncertainty in the odometry of the robot is ignored, one can see that whether or not a "reset" function of the belief state is used to relocate the robot when it has reached an intermediate goal, the robot's belief of its location will diverge from its true position as time goes by, as it is increasingly affected by the odometric uncertainty. From this example, we can see that for the robot to "know" where it is and eventually localize itself, a model of uncertainty is needed.

As shown in Figure 1.1, we consider three common ways in which the uncertainty can enter a system:

1. **Uncertainty in the Dynamics**: Uncertainty can enter a system through the model of the physical system. High-order and nonlinear dynamics can be unknown or deliberately ignored in order to keep the model simple and tractable. This lack of knowledge of the true underlying dynamics introduces some uncertainty in the model, thus rendering the behaviour somewhat
unpredictable.

Let us revisit the previous mobile robot example of Figure 1.3. Consider for instance that some nonlinear effects in the actuators of the robot need to be modeled in order to precisely capture their dynamics. To simplify the model, one might want to omit these effects. Yet, not accounting for the uncertainty introduced in the dynamics by the simpler model might lead to imperfect movement of the robot. As a result, the robot might have a false belief of where it is, even though it reaches the intermediate goals.

Another example of such uncertainty is the high-order effects of evaporation of water in a steam boiler system which are usually ignored to simplify the model.

2. **External Disturbances:** Disturbances (not attributed to the inaccuracy of the model of the agent itself), can also enter in the dynamics of the robot. Such uncertainty is caused by external disturbances that are outside of the control of the system. E.g., barometric pressure or wind speed are elements outside of the body's dynamics which can influence the behaviour of an airplane system.

3. **Sensor Noise:** This type of uncertainty enters the system through imprecise sensor measurements. For example, sensor noise could come from the sonar readings of a mobile robot.

Now that we have briefly introduced the notion of uncertainty, let us discuss the dynamical nature of the systems we are interested in.

Considering the dynamic aspect of the problem, we observe that the relationship of a robot with its environment is one that changes (somewhat unpredictably due to the underlying uncertainty) over time. Therefore, to reason about the overall behaviour of a robotic system, we need a model which characterizes the behaviours of its various components (body, controller and environment) and derives the overall behaviour of the complete robotic system.

As described above, the actions of a robotic agent are controlled by a software component called the controller. Robotic systems are in fact controlled dynamical systems. Such systems are designed to meet certain requirements such that their behaviour must satisfy certain behavioural constraints. Typical requirements on the behaviour of robotic systems include safety: a system should never be in a certain undesirable situation; reachability: a system should eventually reach a given predefined
goal and persistence: a system should approach a given goal infinitely often. In order to characterize the desired properties of a system, it is important to be able to represent such behavioural constraints. Therefore, a formal language for specifying behavioural constraints is needed. Moreover, given a representation of a behavioural constraint, a formal method for behaviour verification is essential for ensuring the correctness of the system with respect to these specifications. However, both the specification and the verification task must be performed while taking into account the uncertainty in the systems. Behavioural constraint specifications must be able to express uncertain (or probabilistic) requirements, while the verification methods must be able to handle the fact that while some behaviours might diverge from the specifications, the system will obey them with respect to some given measure (e.g., on average or according to some probability threshold).

Another challenge in the modeling and analysis of robotic systems is the design of the controller, most often referred to as the control synthesis problem. Given a model for the dynamics of the robotic agent and its environment, the task is to automatically generate a controller ensuring that the system's behaviour will follow a certain specification. This problem is especially complex in the presence of uncertainty. Although the field of stochastic optimal control has made great leaps in the recent past, the problem of control synthesis on uncertain robotic systems is still mostly unexplored.

In summary, we propose to address four problems that are at the core of designing and analyzing uncertain robotic systems:

1. **Modeling:** how to model a robotic system exhibiting uncertainty?

2. **Behavioural Constraint Specifications:** how to specify properties that the uncertain behaviour of a robotic system should display?

3. **Behavioural Verification:** how to guarantee that a robot will do the right thing, where the right thing is defined according to the designer’s favorite measure (e.g., safety or reachability) for unpredictable behaviour?

4. **Control Synthesis:** how to automatically generate a controller that will act “optimally” given the uncertainty present in the robotic system?

In Figure 1.4 we present a visual representation of the summary of the problems we consider in this thesis along with the proposed solutions. These proposed solutions are discussed in the
Will the robot do the right thing?
On average? Probabilistically?

Behaviour Verification
Verification Algorithms

Uncertain Robotic Systems
PCN Model

Requirements Specification
Average timed \( \mathcal{A} \)-Automata
\& PATTL

Control Synthesis
Robust, Stochastic Control

What is the possible behaviour of the robot?

What is the right thing for the robot to do considering its uncertain environment?

How to make the robot do the right thing, even under uncertainty

Figure 1.4: The problems and proposed solutions

1.2 Proposed Solutions to the Considered Problems

When initially considering the task of modeling uncertain hybrid systems, our approach was motivated by numerous considerations. The first one corresponded to the fact that hybrid systems consist of interacting discrete and continuous components, both in terms of time and domain. Therefore, rather than simply developing a model with fixed time and domain structures, an encompassing model for hybrid systems should be developed on both abstract time structures and abstract data types. Secondly, most hybrid systems are complex and need to be specified with multiple interacting components. Hence, an efficient model for hybrid systems should allow hierarchical and modular modeling. Third, by combining discrete and continuous components, hybrid systems are generalizations of those basic systems. Therefore, any framework modeling a hybrid system should be at least as powerful as the existing computational model used to model the basic systems. Finally, uncertainty is inherent in any realistic hybrid system. Whether it is through sensor noise, external disturbances or imprecision in the model of the dynamics, uncertainty has an important effect on the behaviour of hybrid systems. Therefore, a model for a hybrid system should support probabilistic
and non-deterministic modeling. In short, a model for hybrid systems should be unitary, modular, powerful and allow for explicit modeling of uncertainty within the system.

As time is an intrinsic component of a dynamical hybrid system, we start with a general definition of time. We view time as a linearly ordered set. In order to reason about the evolution of time, we associate a metric distance with any two time points and a measure with some intervals of time points. By using such a general time structure, we create an abstraction for event-based as well as discrete and continuous time. In order to study discrete and continuous domains in a unitary fashion, we cast domain structures within abstract algebra topology and measure theory. Given such a time structure and a domain structure, we define two basic types of element in dynamical systems: stochastic traces that are stochastic functions from time to domains, and transductions that are mappings from stochastic traces to stochastic traces with the causal restriction (which is called adapted in the measure theory literature), i.e., the output value at any time is (stochastically) determined only by its input values up to (and including) that time. We will define these notions in detail in Chapter 2.

Using these notions, we develop the Probabilistic Constraint Net (PCN) model, an extension of the Constraint Net (CN) model [ZM95a]. PCN is built on an abstract dynamics structure composed of a multi-sorted set of stochastic trace spaces and a set of basic transductions. Basic transductions are the building blocks of our framework. By combining basic transductions we are able to build complex models of systems. The set of basic transductions is made of the following components which will be formally defined in Chapter 2: transliterations (memory-less combinational processes), unit and transport delays and generators. Generators allow for the modeling of uncertainty by introducing random variables in the model. We differentiate between deterministic and probabilistic transductions depending on whether or not they encompass a generator. Compound transductions of each type are built by combining simple transductions of the same type with transliterations and delays. Figure 1.5 shows the hierarchy of transductions within the PCN framework. Note that non-deterministic transductions are not represented on this diagram as these transductions are obtained from any transduction with a hidden input. We will clearly explain the concept of hidden inputs when we describe the PCN syntax in Chapter 3. As an example of what a transduction can be, consider the function \( f(x) = 2x \). This function is a deterministic transduction (in fact it is simply a transliteration) while \( g(x) = f(x) + N \) (where \( N \) is a Gaussian random vari-
Figure 1.5: Transduction Type Hierarchy

able), is a probabilistic transduction made of the transliteration $f$ along with the generator $N$ for a Gaussian random variable.

Our framework also includes event-driven transductions. Event-driven transductions act as bridges between continuous and discrete time components, or as synchronizers among asynchronous components. This will allow the user to model hybrid systems possessing multiple clocks.

Syntactically, a probabilistic constraint net is represented as a bipartite graph with two types of nodes: locations and transductions, and with a set of connections between locations and transductions. Locations can be seen as variables of the system while transductions are the transformational processes (functions) that are applied to these variables.

Semantically, a probabilistic constraint net represents a set of stochastic equations, with locations as variables (possibly random variables if they are output of probabilistic transductions), and transductions as (possibly random) functions. The semantics of a probabilistic constraint net model, with the values over time of each location denoting a stochastic trace, is the least solution of the set of equations. Since uncertainty is present in the model, a solution to the set of equations is a random process that can be depicted by its probability distribution.

As systems grow in complexity, it becomes essential to divide them into simpler components interacting together to create the system as a whole. In the PCN framework, we define a module as a probabilistic constraint net with a set of locations serving as its interface. Using modular and aggregation operators on modules, it is then possible to obtain a hierarchical model of the system.
under study. The semantics of hierarchical systems can then be obtained from the semantics of their subsystems (modules) and their connections.

As a simple example of a PCN model, one can denote Equation 1.1 as a probabilistic constraint net in which \( \sin, \cos, \tan \) and \(*\) are transliterations (basic transductions). The graphical representation of this PCN is shown in Figure 1.2(b). We will clearly define the meaning of each component of the graphical representation of a PCN when we formally introduce the syntax in § 3.1.1. For this system, one can build a PCN module using the locations (variables) \( v, \alpha, x, y, \theta \) as its interface.

In general, we can model a control system as a module that can be further decomposed into a hierarchy of interactive modules. The higher level control signals are built on event-driven transductions while the lower levels are analog control components. Note that in our framework, unlike in most modeling paradigms, the environment of the robot can also be modeled as its own module. From Figure 1.1, one can observe that a robotic system as a whole can be obtained by integrating a module for the body, the controller and the environment, which can be represented, in general, by the following equations:

\[
X = BODY(U, Y), \quad U = CONTROLLER(X, Y), \quad Y = ENVIRONMENT(X).
\]

Contributions

The general contribution of this thesis is to augments Constraint Nets with probabilistic event generators to produce Probabilistic Constraint Nets (PCN), an intuitive and very general framework for modeling and verifying probabilistic hybrid systems. More specifically, we establish a unified foundation for hybrid dynamical systems exhibiting uncertainty and further propose that an integrated approach to the design and analysis of robotic systems and behaviours should be taken and that uncertainty within the systems should be expressed explicitly and considered throughout the whole analysis.

PCN introduces an abstraction and a unitary framework to model hybrid dynamical systems exhibiting uncertainty. PCN is modular and hierarchical, that is, the dynamics of the environment as well as the dynamics of the system can be modeled individually, and then, integrated using aggregation operators provided by the framework. Moreover, PCN supports multiple levels of abstraction, based on abstract algebra, topology and measure theory, to model and analyze a system and its underlying uncertainty at different levels of detail. Due to its rigorous measure theoretic and alge-
braic foundations, PCN can be used to define programming semantics of real-time languages for uncertain controlled dynamical systems.

Second, to address the specification of behavioural constraints, we develop both a probabilistic arbitrary-timed branching time logic (PATTL) and average timed $\forall$-automata as specification languages. PATTL is a probabilistic branching time logic developed on abstract time and domain structures which extends already defined logics such as the Computation Tree Logic (CTL) [Eme90] and the Probabilistic Computational Tree Logic (PCTL) of Hansson and Jonsson [HJ94]. Average timed $\forall$-automata extend timed $\forall$-automata to accept timed traces where the average behaviours of the system is considered. These finite automata are powerful enough to specify properties of sequential and timed behaviours of hybrid systems, such as safety, reachability, persistence and real-time response corresponding to the on average constraint. Third, we develop a formal verification method for average timed $\forall$-automata specification, by combining a generalized model checking technique for automata with a generalized stability analysis method for uncertain dynamical systems. This verification method can be semi-automated for discrete time systems and further automated for finite domain systems. Furthermore, we show that existing verification methods can be extended to apply to the task of verifying PATTL requirements on PCN models.

Finally, we discuss an approach to control synthesis for uncertain dynamical system emerging from stochastic and robust control as well as from the dynamic programming field of research.

1.3 Related Work

In this section, we present what the author believes to be some of the most relevant work done in the area of modeling and verification of probabilistic systems. We first provide a survey of the framework on which PCN is based: Constraint Nets. Following this short introduction, we present the notion of probabilistic systems and some of the most common models used to represent such systems. We conclude with a survey of the research performed on the verification of behavioural constraints of probabilistic systems.

1.3.1 Review of Constraint Nets

The CN modeling framework is built on a topological view of time and domain structures along with notions of traces, events and transductions. CN is a modeling framework for deterministic
system. It does not allow for the explicit modeling of uncertainty. The set of basic transductions in CN is limited to transliterations and delays. For the remainder of this dissertation, when we refer to a deterministic system within the PCN framework, we will mean a system composed of only transliterations and delays, i.e., a system which does not include any generators.

The following definitions, which we review below for the sake of clarity, represent the foundation for the notion of time and domain in the Constraint Net framework. We retain these concepts and extend them to develop the Probabilistic Constraint Net model. We refer the reader to Chapter 3: *Topological Structure of Dynamics*, from [Zha94], for a thorough introduction to these concepts and more concerning the CN modeling framework.

Within (P)CN, we view time and domains as fully abstract concepts, defined as follows:

- **Time** is presented as a linearly ordered set \((T, \leq)\) with a least element \(t_0\), and a metric \(d\) on the set \(T\). With this abstract representation of time, one can model discrete, continuous or event-based systems.

- Simple and composite domains denote, respectively, simple data types (e.g. reals, integers, Boolean and characters) and structured data types (e.g. arrays, vectors or objects). A *simple* domain is represented as a pair \((A \cup \{-, \notin\}, d_A)\) where \(A\) is a set, \(-, \notin\) means undefined in \(A\), and \(d_A\) is a metric on \(A\). *Composite* domains are obtained by the product of simple domains. With this abstract notion, domains can be numerical or symbolic, discrete or continuous.

We also view the behaviour of a dynamical system as a set of traces, where the trace of each variable is defined as follow:

- **Traces** can intuitively be perceived as changes of values over time. Formally, a mapping \(v : T \rightarrow A\) from time \(T\) to range \(A\) is called a *trace*. An *event* trace is a trace with a Boolean range. An event in an event trace is a transition from 0 to 1 or from 1 to 0.

It is important to note that for a deterministic system, the initial value of a trace completely defines the whole trace. Indeed, given the (deterministic) dynamics of a system, with the initial value of each variable, one can completely predict the value of each variable over the time history of the system. In the presence of uncertainty, however, this is not true. Given the model of the
uncertain dynamics of a system and its initial values, one can only predict the possible trajectories, i.e., the set of all traces that the system can take. Therefore, one has to be extremely careful when designing systems exhibiting uncertainty since although some trajectories might be acceptable for a system, chances are that some undesired trajectory can be achieved with the current conditions. This is related to the notion of behaviour verification, which we will discuss in Chapter 6-8.

In the CN framework, functions are referred to as transductions. Formally, these are defined as:

- **Transductions** are causal mappings from inputs to outputs over time, either operating according to a certain reference time or activated by external events. Note that in PCN, the notion of causal mapping will be replaced by adapted mapping, its measure theoretic equivalent. However, both concepts have a similar intuitive meaning. The class of simple transductions contains transliterations and delays. A transliteration is a pointwise extension of a function. It can be seen as a transformational process without internal state (memory). A delay can be seen as a memory cell.

A constraint net consists of a finite set of locations, a finite set of transductions and a finite set of connections. Formally, a constraint net is a triple $\langle Lc, Td, Cn \rangle$, where $Lc$ is a finite set of locations, $Td$ is a finite set of labels of transductions, each with an output port and a set of input ports, $Cn$ is a set of connections between locations. A location can be regarded as a wire, a memory cell or a variable.

One of the advantages of CN is its graphical representation which allows one to represent a constraint net by a bipartite graph where locations are depicted by circles, transductions by boxes and connections by arcs. For example, the graph in Figure 1.6 represents the constraint net with a continuous time structure for the differential equation $\dot{s} = f(s)$ with initial value $s(t_0) = s_0$.

Semantically, a constraint net represents a set of equations, where locations are variables and
transductions are functions. The semantics of the constraint net, with each location denoting a trace, is the least solution of the set of equations.

**Specification and Verification**

Along with the formal modeling framework of Constraint Nets, Zhang [Zha94] developed two specification languages to represent the behavior requirements of a system: *Timed Linear Temporal Logic* (TLTL) and *Timed V-automata*.

TLTL is a generalization of *Linear Temporal Logic* [MP91] where “linear” indicates linear orders and “timed” stands for metric distances between time points. Even though TLTL allows for formal specification requirements, there is no general procedure for verifying the behavior of a given system. However, TLTL has been proven to be very useful when used within the problem of control synthesis as we will show in next section.

One of the most popular alternatives to temporal logics for expressing the behavior of a dynamical system is automata. In this framework, the behavior of a system can be seen as a language, and the specification can be represented as an automaton. Given such a representation, the verification step amounts to showing the inclusion of the behavior language within the language accepted by the automaton.

Timed V-automata extend V-automata [MP87] by accepting timed traces. They provide a graphical representation, which is more intuitive and sometimes simpler than (temporal) logics. Furthermore, a formal verification method (via a set of sound and complete verification rules), based on model checking and stability analysis, was developed. It was shown that, given a constraint net model of a discrete time system, a set of state formulae can be deduced and then checked using an automatic or interactive theorem prover. It was also demonstrated that, if the system has a finite state space, then the verification rules can be used to deduce a fully automatic verification algorithm with polynomial time complexity in the size of the specification and the size of the system.

**Constraint-Based Dynamical Systems**

It is argued that most dynamical systems are inherently constraint-based, where constraints may range from physical limitations or environmental restrictions to safety requirements. In the CN framework, constraint satisfaction is seen as a dynamical process that asymptotically approaches
the solution set of the given (possibly time-varying) constraints. A constraint solver is viewed as a constraint net whose behavior is a dynamical process that is asymptotically stable at the solution set of the constraints, i.e., whenever the system exits the solution set due to some disturbance, it will eventually return to a state which is part of the set satisfying the constraints. The behavior of such a constraint-based system can be specified using TLTL or timed \( \forall \)-automata. For example, using the specification language of TLTL, a persistence specification would be represented by \( \Box \Diamond C^\epsilon \) which means that for \( C^\epsilon \), the \( \epsilon \)-neighborhood of the solution set of the set of constraints \( C \) will always (\( \Box \)) eventually (\( \Diamond \)) be reached.

Many constraint methods can be implemented in the CN framework. Two types of problems are mentioned in [Zha94], global consistency and optimization. Typically, global consistency denotes the problem of finding a solution satisfying all the given constraints. Thus, global consistency corresponds to solving hard constraints. On the other hand, unconstrained optimization refers to the problem of minimizing an energy function of \( n \) arguments \( \xi : \mathbb{R}^n \rightarrow \mathbb{R} \) and corresponds to solving soft constraints. Finally, another type of constraint satisfaction problem is introduced: constrained optimization. This type of problem consists of solving (soft) constraints subject to the satisfaction of a set of hard constraints.

Typically, there are two different types of constraint methods: discrete relaxation and differential optimization. The former can be represented as a state transition system while the latter can be represented by a state integration system. In [Zha94], a few constraint methods are introduced; namely the (discrete) projection method for global consistency; the (discrete) Newton's method along with the (continuous) gradient method for unconstrained optimization and finally, the (continuous) penalty method and the (continuous) Lagrange multiplier method for constrained optimization.

**Control Synthesis**

The problem of control synthesis is, given some requirements specification for the behavior of a dynamical system along with models of the environment and the body, to generate a controller inducing a behavior which satisfies the specification. With CN models of the environment and the body, Zhang and Mackworth [ZM95b] use TLTL for the requirement specifications, which are restricted to constraint-based specifications such as reachability, safety and persistence. They then
synthesize the controller using constraint methods such as those presented in the previous section. Basically, they view controllers as embedded real-time constraint solvers.

1.3.2 Modeling Probabilistic Systems

We are now ready to introduce our new paradigm, PCN, which extends CN to explicitly model and reason with uncertainty inherent in dynamical systems. However, let us first start by defining the more general class of non-deterministic systems. As opposed to deterministic systems, for which the behaviors exhibit no randomness and thus can be predicted perfectly, non-deterministic systems present inherent incomplete description which renders state transitions unpredictable. For a given system, when the nature of the underlying randomness of the transitions is unknown, and is not estimated by a probability distribution over states, we say that the system is non-deterministic.

In general, the occurrence of non-determinism is due to intervention from the environment over which the system has no control or to the asynchronicity of concurrent processes. On the other hand, when the randomness of the state transitions of the systems is modeled, we talk about the notion of probabilistic or stochastic systems. Probabilistic systems are divided into two classes. We call a probabilistic system purely probabilistic if, for every state of the system, there is exactly one specified probability distribution over the next possible states. A system that exhibits both probabilistic and non-deterministic behavior is referred to as a generalized probabilistic system.

Let us now introduce the most commonly used frameworks for modeling systems with uncertainty.

Probabilistic Systems

One of the most commonly used formalisms for modeling probabilistic systems is the finite-state discrete-time Markov chain.

A Markov chain has a specified initial state $s_0$, and for each state $s_i$ included in the state space $S$, there is an assignment of truth values to atomic propositions. The set of true atomic propositions

\[ \text{(Note that deterministic systems can also be seen as purely probabilistic models over the trivial one-point probability space. This means that for each state, the probability distribution is restricted to 1 for the realized state and 0 for all other states.)} \]

\[ \text{(The initial state of the system can either be given and unique or else a prior probability distribution over all "starting" states is given.)} \]
uniquely identifies each state.

Conceptually, a Markov chain is the representation of a probabilistic system. At each time step, the system changes state following a probability distribution given by the transition probability function $P : S \times S \rightarrow [0, 1]$. Paths, which are also called execution sequences, arise from resolving the probabilistic choices. Formally, a path in a Markov chain $M$ is an infinite sequence $\pi = s_0s_1 \ldots$ where $s_0$ is the initial state of $M$, $s_i$ are states and $P(s_i, s_{i+1}) > 0 \ \forall i$. Finite paths can also be extended to infinite ones by simply repeating the last state infinitely. The probability of a given finite path $\pi$, starting at $\pi(0)$ in $M$ is given by $P_p(\pi) = P(\pi(0), \pi(1))P(\pi(1), \pi(2)) \ldots$, where $\pi(i)$ is the $i$-th state of the path $\pi$.

**Generalized Probabilistic Systems**

The modeling of a probabilistic system requires the full specification of the transition probabilities. However, in many practical settings, estimating accurately the transition probabilities can be a complex task. One might want to avoid such a task and leave some of the transition probabilities unspecified, thus modeling those as non-deterministic transitions. Therefore, for many systems, a formalism allowing us to model behavior that is both probabilistic and non-deterministic is desirable.

Many models that take into account probability and non-determinism have been recently presented in the computer science literature. An extension of the Markov chain model presented earlier is the Markov Decision Process (MDP). MDPs were introduced by Bellman [Bel57] and Howard [How60] and they have been the subject of much research in Decision Theoretic Planning (AI) [BDH99] and Operation Research. The Independent Choice Logic of [Poo97] has been introduced as a model for multiple agents under uncertainty. Inspired by probabilistic Horn abduction [Poo93], game theory [Mye91, Ord86], Markov Decision Processes (MDP) [Put94] and Bayesian Networks [Pea85, Pea88], it provides a natural and concise representation of agents under uncertainty. A logical representation is used to tackle the probabilistic issue. [CBT00] introduced DTGolog, a framework based on MDPs with the Golog programming language [LRL+97]. They present a framework that combines decision theoretic planning with agent programming.

Several related models for generalized probabilistic systems include probabilistic finite-state programs [PZ86], concurrent Markov chains [Var85], Probabilistic Non-Deterministic Systems.
(PNS) [BdA95], *Timed Probabilistic Non-Deterministic Systems* (TPNS) [dA97b] and *Coloured Petri Nets* [Jen81, Jen97]. TPNSs are similar to PNSs, with the distinction that TPNSs allow modeling of generalized probabilistic systems in which state transitions have different duration. Coloured Petri Nets (CP-nets or CPNs) is a modeling language developed for systems in which communication, synchronization and resource sharing play an important role. CP-nets combine the strengths of ordinary Petri nets with the strengths of a high-level programming language. Petri nets provide the primitives for process interaction, while the programming language provides the primitives for the definition of data types and the manipulations of data values. The syntax of Petri Nets (and its generalization Coloured Petri Nets) is very similar to that of PCN, i.e., a bipartite graph. However, the semantics of PCN is for maximum parallelism, while the semantics of Petri Nets is for concurrency. Moreover, Petri Nets are not as general as PCNs since timed Petri Nets are restricted to a single global clock while our framework allows for the modeling of hybrid systems with multiple clocks.

### 1.3.3 Requirement Specifications and Verification Techniques

The modeling of a probabilistic system focuses mainly on the structure and components of the system. However, no matter how fine grained the resulting model is, the overall behavior of the system cannot be fully specified. It is often very important to be able to impose restrictions on the behavior of the system. These restrictions represent global properties that should continuously hold in the system under study. For example, a requirements specification for a coffee delivery robot could be that coffee will be delivered within a bounded period of time after reception of the request. Requirements specifications restrict the behavior of a system by requiring (or forbidding) the system to be in certain states. These specifications become essential when building safe and reliable systems. Therefore, formal methods for expressing requirements specification and verifying that they hold are called for.

There are two main approaches to the problems of modeling and verifying systems: the first one consists of a single language $L$, used for both the modeling and the specifications of the requirements of the system. The verification task then amounts to showing that the set of behaviors of the model $L_M$ is a subset of the behavior allowed by the specification $L_S$, i.e., $L_M \subseteq L_S \subseteq L$. The
second approach uses two different languages: a modeling language $M$ and a specification language $S$. For a given system $A$, the verification procedure then amounts to showing that $M_A$ entails $S_A$, i.e., $M_A \models S_A$. Since the power of a specification language and the simplicity of the verification procedure are inversely related, we have to reach a compromise between the expressibility of the specification language and the applicability of the verification method. Using the same language for modeling and specification might render the verification procedure infeasible for complex dynamical systems. As was the case for the constraint net framework, we will, in this dissertation, use two different languages for modeling and specification.

In this section, we present several types of formalism that have been used to specify probabilistic properties of systems along with algorithms used to verify those properties. We restrict our survey to methods which use different languages for modeling and specification.

**Qualitative Verification and Temporal Logics**

Given a probabilistic system $P$, showing that a property represented by a temporal formula $\psi$ is fulfilled by all computations of the system, is referred to as *qualitative verification*. This amounts to showing that $\psi$ is satisfied with probability 1. A lot of work has been done on the subject of qualitative verification. In this section, we attempt to summarize the most relevant research from the literature.

Three temporal logics evaluated on Markov chains were introduced by [LS82]. In this work, they present a linear time system which follows a linear history while permitting reference to un­taken alternatives. [Pnu83] presented an alternative approach based on standard linear time logics. However, instead of specifying a new temporal logic, he introduced the notion of *extreme fairness*. This concept of fairness imposes fairness on probabilistic choices. [HS84] introduced a system which is based on branching time temporal logics interpreted on Markov chains.

Probably the first one to raise the question of qualitative verification, [Var85] solved the problem on systems modeled with *concurrent Markov Chains* by automata theoretic methods which he extended to take into account the probabilistic nature of the problem. The complexity of the algorithm presented was showed by [CY88] to be doubly exponential in the size of the temporal property. They established the optimality of Vardi’s algorithm by proving that the lower bound for this problem was doubly exponential.
[PZ93] extended previous work [PZ86] and introduced a qualitative verification method for probabilistic systems. The specifications are written using a restricted linear time temporal logic (RTL) which allows for all the operators of TL except for the until future operator. Unlike the algorithm of [CY88], the ones presented in [PZ86, PZ93] are of single-exponential complexity in the size of the specification. The reason for this diminution in the complexity is due to the use of RTL as a specification language. [PZ93] also present the notion of α-fairness, an improvement on the concept of extreme fairness of [Pnu83].

Real-time systems with continuous random delays and discrete probability and time have been modeled via the Real-time probabilistic systems of [ACD91]. This work introduced an algorithm for verifying qualitative specifications written in the real-time temporal logic (TCTL) of [ACD90]. Later, [ACD92] extended this work for continuous-time systems with specifications using the notion of timed automata introduced in [AD90].

Quantitative Verification and Temporal Logics

In contrast with qualitative verification which attempts to prove that a certain property holds for every computation with probability one, quantitative verification refers to determining the probability with which a property is satisfied within a given system.

In the last few years, there has been a significant amount of research done in the area of probabilistic model checking. Two of the most popular temporal logic approaches are linear-time logics and branching-time logics. The former considers time to be a linear sequence while the latter adopts a tree structure time, allowing some instances to have more than one successor. The choice between linear and branching models should be dictated by the type of properties one wishes to study. For this work, we intend on modeling probabilistic systems, and since probability measures on system behaviors are similar in structure to the path quantifiers used in branching-time temporal logics, we will focus on these logics exclusively.

In this section, we introduce a few temporal logics based on the branching-time temporal logics CTL and CTL* [CE81, BAPM83, EH85]. The underlying model of CTL is a tree of all possible computations.

Probabilistic real time Computational Tree Logic (PCTL) [HJ94] was the earliest of many ex-

\footnote{For a more in depth discussion of this issue, the reader is referred to [Lam80].}
tensions made to CTL and CTL*. It considered systems modeled as discrete Markov chains and was
developed to specify the probability of satisfying temporal formulae within a given number of chain
transitions. The language allows formulae such as: $\diamond \leq 20^{0.2} \psi$. This formula expresses the property
that, with probability of at least 0.2, the formula $\psi$ will become true within 20 state transitions. The
algorithm presented in this work has polynomial time complexity in the size of both the formula and
the Markov chain. PCTL was later extended by [Han94] to take into account non-determinism and
provide a more refined model of time and yielded an algorithm with time complexity exponential in
the size of the system.

Later, [ASB+95] introduced pCTL*, a probabilistic variant of CTL* where the operator P, used
to express bounds on the probability of systems behaviors, is added. They also introduce the logic
pCTL which is essentially identical to the logic PCTL of [HJ94]. This work considered two differ­
ent types of system models: discrete Markov processes and generalized Markov processes. Further­
more, although they show, based on results arising from real closed field theory, that model checking
with generalized Markov processes is elementary decidable, no practical verification algorithm is
presented.

A variant of pCTL* was later presented by [BdA95] which extended the logic to systems with
non-determinism. They show that model checking for PCTL and PCTL* on PNSs can be done in
time polynomial in the size of the PNS. However, in terms of the size of the formula, linear and
doubly exponential times are respectively required for the PCTL and PCTL* model checking.

Based on PCTL, the Probabilistic Branching Time logic (PBTL) was introduced by [BK97,
BK98a, BK98b] and was intended for systems where fairness constraints are imposed. Syntactically,
PBTL is almost identical to PCTL with the addition of universal and existential quantifiers over what
they call adversaries. Baier and Kwiatkowska gave a model checking algorithm for PNSs with
PBTL specifications. It is based on the algorithm from [BdA95] and has the same time complexity
(polyomial in the size of the system and linear in the size of the formula).

Recently, [dAKN+00] developed a symbolic model checker for PBTL using Multi-Terminal
Binary Decision Diagrams (MTBDD) [CFZ96] which are an extension of Bryant’s Boolean De­
cision Diagram (BDD) [Bry86]. They adopt the Kronecker representation of [Pla85] which yields
a very compact MTBDD encoding of the system. Their tool allows the model checking of purely

---

5In the literature, adversaries are also referred to strategies [BdA95] or schedulers [Var85, BK97].
6These diagrams are also known as Algebraic Decision Diagrams (ADD) [BFG+93]
probabilistic systems of up to $10^{25}$ states, and generalized probabilistic systems of up to $10^{30}$ states. It was shown that the model checking of qualitative properties could be done very rapidly. Furthermore, since PCTL is a subset of PBTL, their tool can also be used to verify properties expressed in that logic.

In all the logics discussed so far, the representation of a single time unit is interpreted as an instantaneous transition in the model of the system. However, when modeling real-time systems, this may not be desirable, since it is possible for different events to take different amounts of time. As mentioned above, TPNS were developed exactly to address that issue. Along with the TPNS model, [dA97a] introduced the logic pTL*. The motivation behind this work is to be able to study performance and reliability properties of probabilistic systems. pTL* extends CTL* with two new operators: P as described earlier and D which is used to express bounds on the average time between events. The time complexity of the verification algorithm presented in this work has been shown to be polynomial in the size of the system and doubly exponential in the size of the specification formula.

**Probabilistic Bisimulation**

Bisimulation is a very important concept in the field of concurrent systems [DEP02, Eda95, LS91]. Bisimulation is one of the most popular methods for process equivalence as it enjoys a fixed-point characterization. With bisimulation, one obtains an equivalence relation which allows for the comparison of states of a system with each other. Desharnais et al. showed that bisimulation was sound and complete for PCTL* and CSL. That is, two states will satisfy the same set of PCTL* or CSL formula if and only if they are bisimilar [DGJP02, DP03]. This provides an interesting approach to the verification of properties satisfied by a model as one can use a simpler, yet bisimilar, model to prove the satisfaction of the property.
Chapter 2

Measure-Theoretical and Topological Structure of Dynamics

In this chapter, we present a measure-theoretical and topological approach to the structure of dynamical systems exhibiting uncertainty. We start with an introduction to probability, measure and topological theory as these concepts will play a central role in defining the structure of dynamics. The proofs of these results, along with any other formal result presented throughout this dissertation, are given in Appendix A. Within the field of topological theory which is broad and vast, we focus on two particular types of topology: partial order topology and metric topology. Based on these two types of topology and on measure-theoretic concepts, we formalize the key constituents of our model of probabilistic dynamical systems: time, domain and behaviour structures. We will refer to stochastic traces when discussing the (uncertain) evolution of systems over time. In a probabilistic setting, a stochastic trace denotes a set of traces which defines a system's behaviour. One of the most important concepts introduced in this chapter is one of (probabilistic) transduction, that we will define as a causal mapping from stochastic traces to stochastic traces. To conclude this chapter, we define abstract dynamics structures.

2.1 General Topology, Partial Order, Metric Space and Measure Theory

The Probabilistic Constraint Net framework has been developed to model probabilistic systems and processes which deterministic frameworks cannot adequately describe. With the PCN framework,
we want to account explicitly for the sources of uncertainty in the systems of interest. Historically, probability theory has been developed to assign probabilities to events of interest associated with the outcome of some experiment. Probability (measure) theory appears to be ideally suited to serve as a foundation for the PCN framework.

Before introducing the PCN framework formally, we wish to provide an overview of some of the mathematical preliminaries in general topology theory and some of the elementary results of probability and measure theory that will be used throughout this dissertation. General topology allows us to reason about convergence, connectivity and continuity while probability and measure theory are tools that enable us to reason about integration in arbitrary measure spaces. For a more comprehensive introduction to the mathematical foundations introduced here, the reader is referred to [Gem67, Hen88, Vic89, MA86, War72, Roy88] while we suggest [Bil86, Bre68, Wil91, Rud66] for a more thorough training in measure and probability theory. Since all these notions are well known results, we quote them freely in this document. Moreover, some of the mathematical preliminaries introduced here are also components of the CN framework. Hence we reproduce some of the definitions introduced in Chapter 3 of [Zha94].

The fundamental concept of probability theory is a general space of outcomes, called a sample space, $\Omega$, which contains all possible outcomes of the experiment conducted. We will denote a single elementary outcome as $\omega$, with $\omega \in \Omega$. Now let us define $A$ as a specific event of interest, a set of outcomes of the experiment: $A \subset \Omega$. We will assume that such experiments are of the form of a probability space $(\Omega, \mathcal{F}, P)$ where $\Omega$ is the sample space, $\mathcal{F}$ is a $\sigma$-algebra on $\Omega$ and $P$ is a probability measure on $(\Omega, \mathcal{F})$. We will define these notions shortly but let us first introduce the notion of topological space, a concept central to the CN formalism and thus to the PCN framework as well.

**Definition 2.1 (Topology and Topological space)** Let $X$ be a set and $\emptyset$ be the empty set. A collection $\tau$ of subsets of $X$ is said to be a topology on $X$ iff the following conditions are satisfied:

- $X \in \tau$ and $\emptyset \in \tau$.
- If $X_1 \in \tau$, $X_2 \in \tau$, then $X_1 \cap X_2 \in \tau$.

$\text{\footnote{Recall that in view of our Bayesian approach to probability theory, we view uncertainty as arising from the user’s lack of knowledge or as a result of an overly abstracted model.}}$
• If \( X_i \in \tau \) for all \( i \in I \) (\( I \) finite), then \( \bigcup_i X_i \in \tau \).

\((X, \tau)\) is called a topological space.

When it is clear from the context, we will use \( X \) to denote topological space \((X, \tau)\).

The members of a topology \( \tau \) are said to be \( \tau \)-open subsets of \( X \), or simply open if no ambiguity arises. A subset \( S \) of \( X \) is closed iff \( X - S \) is open. From this definition, it is easy to show that for any topology on \( X \), \( X \) and \( \emptyset \) are both open and closed.

There are two extreme topologies on \( X \). The coarsest (or smallest) topology on \( X \) is called trivial if only \( X \) and \( \emptyset \) are open and the finest (or largest) topology on \( X \) is called discrete if \( \tau = \mathcal{P}(X) \), where \( \mathcal{P}(X) \) denotes the power set of \( X \). We say that a topology \( \tau_1 \) is a finer topology than \( \tau_2 \) iff \( \tau_1 \supseteq \tau_2 \). As mentioned earlier, topologies allow us to reason about convergence. Therefore, it should not be surprising that topologies can also be defined in terms of limit points.

Let \( x \in X \) and \( N(x) \) be a \( \tau \)-open subset of \( X \) containing \( x \). \( N(x) \) is called a neighborhood of \( x \) w.r.t. \( \tau \). A point \( x \) of \( X \) is a limit point of a subset \( S \) of \( X \) iff every neighborhood of \( x \) also contains a point of \( S \) distinct from \( x \), i.e., \( \forall N(x), N(x) \cap S - \{x\} \neq \emptyset \).

**Proposition 2.1** (1) A subset is closed iff it includes all its limit points. (2) A topology is trivial iff every point \( x \) is a limit point of any subset with elements distinct from \( x \). A topology is discrete iff no point is a limit point of any subset.

We provided results with regards to the convergence notion of topologies. Now we define connectivity and continuity on topological spaces. The notion of continuity will not only be useful for topological concepts, but will also be central to our approach with measure theory.

A topological space is said to be separated if it is the union of two disjoint, non-empty open sets; it is otherwise connected.

**Proposition 2.2** A topological space is connected iff the only sets that are both open and closed are the empty set and the total set.

Now let us introduce the notion of continuous function. Let \((X, \tau)\) and \((X', \tau')\) be topological spaces. A function \( f : \Omega \times X \rightarrow X' \) is continuous iff for any \( \tau' \)-open subset \( S' \) of \( X' \), \( f^{-1}(S') = \{(\omega, x) | f(\omega, x) \in S'\} \) is \( \tau \)-open. Moreover, a function \( f_\omega : X \rightarrow X' \) is pathwise continuous iff for any \( \tau' \)-open subset \( S' \) of \( X' \), \( f_\omega^{-1}(S') = \{x | f_\omega(x) \in S'\} \) is \( \tau \)-open.
Proposition 2.3 (1) Continuous functions are closed under functional composition. (2) A function $f_\omega : X \to X'$ is pathwise continuous, if $x \in X$ is a limit point of $S \subset X$ implies that $f_\omega(x)$ is a point or a limit point of $f_\omega(S) = \{f_\omega(x) | x \in S\}$.

There exists smaller collections of subsets that can represent the open sets. These collections are referred to as a basis and a sub-basis of a topology.

Definition 2.2 (Basis and Sub-basis) A subset $B$ of a topology $\tau$ is said to be a basis for $\tau$ iff each member of $\tau$ is the union of members of $B$. A subset $S$ of $\tau$ is said to be a sub-basis for $\tau$ iff the set $B = \{B | B$ is the intersection of finitely many members of $S\}$ is a basis for $\tau$.

It is also possible to derive new topologies that are based on known ones. Two important types of such derived topologies are called subspace and product topologies.

Proposition 2.4 Let $(X, \tau)$ be a topological space, $X' \subseteq X$ and $\tau' = \{W | W = X' \cap U, U \in \tau\}$. The collection $\tau'$ is a topology on $X'$. We call $\tau'$ the subspace topology on $X'$, and $(X', \tau')$ a subspace of $(X, \tau)$.

Let $\{(X_i, \tau_i)\}_{i \in I}$ be a family of topological spaces and let $\times_i X_i$ be the product set of $\{X_i\}_{i \in I}$. Let $S = \{\times_i V_i | V_i = X_i$ for all but one $i \in I$, and $V_i \in \tau_i$ for all $i \in I\}$. We call $\tau$ the product topology on $\times_i X_i$ iff $S$ is a sub-basis for $\tau$. We call $(\times_i X_i, \tau)$ the product space of $\{(X_i, \tau_i)\}_{i \in I}$. If $X_i = X$ with the same topology for all $i \in I$, $\times_i X_i$ is denoted by $X^I$.

Proposition 2.5 Let $\{X_i\}_{i \in I}$ be a family of topological spaces and $J$ be an arbitrary index set. Then $(\times_i X_i)^J = \times_J X_i^J$.

Definition 2.3 (Hausdorff Topology ($T_2$ space)) A topological space $(X, \tau)$ is said to be Hausdorff ($T_2$) if given distinct $x, y \in X$, there exist disjoint open sets $U, V \in \tau$ (that is, $U \cap V = \emptyset$) such that $x \in U$ and $y \in V$. The trivial topology is non-Hausdorff and the discrete topology is Hausdorff.

Next, we will introduce the notions of partial order topology and metric topology. These two important types of topologies are central to the foundation of our framework. We will highlight the fact that partial order topologies in general are non-Hausdorff and metric topologies are Hausdorff.
2.1.1 Partial order

The results and notions introduced in this section are analogous to those presented in Section 3.1.2 of [Zha94].

The application of a partial order relation to a given set leads to a partially ordered set, which we call, for simplicity, a partial order. Formally, the notion of partial order is defined as follows:

Definition 2.4 (Partial order) Let $A$ be a set. A binary relation $\leq_A \subseteq A \times A$ is called a partial order relation iff $\leq_A$ is reflexive, anti-symmetric and transitive. $(A, \leq_A)$ is called a partial order; it is called a linear order or total order iff, in addition, $\forall a_1, a_2 \in A$, either $a_1 \leq_A a_2$ or $a_2 \leq_A a_1$.

For any partial order relation $\leq_A$, we define $<_A$ as the strict relation of $\leq_A$, i.e., $a_1 <_A a_2$ iff $a_1 \leq_A a_2$ and $a_1 \neq a_2$. We will simply use $A$ to denote partial order $(A, \leq_A)$ if no ambiguity arises.

Now let us introduce the notions of partial orders related to subsets and set products.

Definition 2.5 (Sub-partial order) Let $(A, \leq_A)$ be a partial order and $A' \subseteq A$. A partial order relation $\leq_{A'} \subseteq A' \times A'$ is called the sub-partial order relation on $A'$ iff $a_1 <_{A'} a_2$ whenever $a_1 <_A a_2$. $(A', \leq_{A'})$ is called a sub-partial order of $(A, \leq_A)$.

Definition 2.6 (Product partial order) Let $\{A_i\}_{i \in I}$ be a set of partial orders and $A = \times_{i \in I} A_i$. A partial order relation $\leq_A \subseteq A \times A$ is called the product partial order relation on $A$ iff $a \leq_A a'$ whenever $a_i \leq_{A_i} a'_i$ for all $i \in I$. $(A, \leq_A)$ is called the product partial order of $\{(A_i, \leq_{A_i})\}_{i \in I}$.

Let us now introduce the notion of least and greatest element of a partial order. Note that not all partial orders have such elements.

Definition 2.7 (Least (Greatest) element) Let $A$ be a partial order. An element $\bot_A \in A$ (resp. $\top_A \in A$) is a least (greatest) element in $A$ iff it satisfies the following axiom: $\bot_A \leq_A a$ (resp. $a \leq_A \top_A$), $\forall a \in A$.

It follows from the antisymmetry of $\leq_A$ that least (greatest) elements, if they exist, are unique.

Definition 2.8 (Flat partial order) A flat partial order, written $\overline{A}$, is a set $A$ augmented with a new element $\bot_A$, viz., $\overline{A} = A \cup \{\bot_A\}$ such that $a \leq_{\overline{A}} a'$ implies $a = a'$ or $a = \bot_A$. 
Hence, any set $A$ can be extended to a flat partial order by augmenting a least element $\bot_A \in A$. The element $\bot_A$ is the least element of $\bar{A}$ and $\bot_A$ means undefined in $A$. With this augmentation, we obtain the property that any partial function to $A$ can be extended into a total function to $\bar{A}$, i.e., $f(a) = \bot_A$ if $f$ is not defined at $a$. In this dissertation, we will only consider total functions unless explicitly stated.

A subset of a partial order may have a least upper bound and/or a greatest lower bound.

**Definition 2.9 (Least upper (Greatest lower) bound)** Let $A$ be a partial order, $D \subseteq A$ and $a \in A$. Then $a$ is an upper (lower) bound of $D$ iff $d < A a$ ($d > A a$) for every $d \in D$. Moreover, $a$ is a least upper bound (lub) (greatest lower bound (glb)) of $D$ iff

1. $a$ is an upper (lower) bound of $D$ and
2. if $d'$ is an upper (lower) bound of $D$ then $a \leq_A d'$ ($a \geq_A d'$).

Similarly to the least (greatest) element, it follows from the antisymmetry of $\leq_A$ that the existence of least upper bound (greatest lower bound), guarantees its uniqueness. We use $\bigvee_A D$ ($\bigwedge_A D$) to denote the least upper (greatest lower) bound of $D$ in $A$, when it exists, and we will drop the subscript $A$ if it is clear from context. To adhere to usual mathematical conventions, to denote $\bigvee$ and $\bigwedge$, we will use “sup”, “inf” when $A$ is the set of real numbers and “max”, “min” when $D$ is finite.

One important kind of subset of a partial order is directed subset.

**Definition 2.10 (Directed subset)** Let $A$ be a partial order and $D \subseteq A$. $D$ is directed iff $D \neq \emptyset$ and for all $d_1, d_2 \in D$, the set $\{d_1, d_2\}$ has an upper bound in $D$.

A complete partial order is a type of partial order that will prove very useful throughout this dissertation. We thus present the formal definition along with two propositions related to complete partial orders.

**Definition 2.11 (Complete partial order (cpo))** A partial order $A$ is complete iff:

1. it contains a least element, denoted $\bot_A$, and
2. every directed subset of $A$ has a least upper bound in $A$. 

31
Proposition 2.6 A flat partial order is a cpo.

Proposition 2.7 The product of cpos is a cpo. Let \( \{A_i\}_{i \in I} \) be a set of cpos and \( A = \times_i A_i \). The least element of \( A \) is \( \bot_A \) with \( \bot_A = \bot_{A_i}, \forall i \in I \). Let \( D \) be a directed subset of \( A \). The least upper bound of \( D \) is \( \bigvee_A D \) with \( \bigvee_A D_i = \bigvee_{A_i} D_i, \forall i \in I \), where \( D_i \) is the projection of \( D \) onto its \( i \)th component, i.e., \( D_i = \Pi_i D \).

A topology can be defined from a partial order.

Definition 2.12 (Partial order topology) Let \( A \) be a partial order. A subset \( S \) of \( A \) is open iff (1) \( S \) is upward closed, i.e., \( a \in S \) implies that \( \forall a' \geq_A a, a' \in S \), and (2) \( S \) is inaccessible from any directed subset \( D \) of \( A \), i.e., if \( \bigvee_A D \in S \), then \( \exists a \in D \), such that \( a \in S \). This collection of open sets on \( A \) forms the partial order topology of \( A \).

A partial order \( (A, \leq_A) \) is non-trivial iff there exist two elements \( a, a' \) in \( A \) such that \( a <_A a' \).

Proposition 2.8 The partial order topology of a non-trivial partial order is non-Hausdorff.

Note that partial order topologies have different properties than their more familiar Hausdorff counterparts. For example, every open set in the partial order topology over \( \mathbb{R} \cup \{-\infty, +\infty\} \) with the usual \( < \) order is of the form \( \{x \in \mathbb{R} | x > c\} \) for some \( c \in \mathbb{R} \). Thus, \( (3, +\infty) \) is an open set in this topology, but \( (3, 4) \) is not. Recall that a function is continuous iff pre-images of open subsets are open subsets. Thus, the function \( \lambda x.\sqrt{x} \) is not continuous in the partial order topology over the reals.

Let us now introduce the notion of lattice which will be essential when proving the main results on the semantics of our framework in Chapter 3.

Definition 2.13 Lattice A lattice \( \mathcal{U} \) is defined as a tuple \( (A, \leq) \), formed by a non-empty set \( S \) and a binary relation \( \leq \). The relation \( \leq \) induces a partial order in \( A \). Moreover, it is assumed that for any two elements \( a, b \in A \), there is a least upper bound and a greatest lower bound.

The lattice \( \mathcal{U} \) is called complete if every subset \( B \) of \( A \) has a least upper bound and a greatest lower bound. For notions and facts concerning lattices, the reader should consult [Bir67]

\[ ^2\lambda x.\text{expr}(x) \] is a lambda expression of a function \( f \), equivalent to \( \forall x, f(x) = \text{expr}(x) \).
The following two propositions declare the properties of continuous and pathwise continuous functions in partial order topologies.

**Proposition 2.9** Any continuous (or pathwise continuous) function is monotonic, i.e., if \( f : \Omega \times A \rightarrow A' \) is continuous (pathwise continuous), then \((\omega_1, a_1) \leq_{\Omega \times A} (\omega_2, a_2) \) \((a_1 \leq A a_2)\) implies \( f(\omega_1, a_1) \leq_{A'} f(\omega_2, a_2) \) (\( f(\omega(a_1) \leq_{A'} f(\omega(a_2)) \)).

**Proposition 2.10** Let \( A \) and \( A' \) be two cpos. Then \( f : \Omega \times A \rightarrow A' \) is continuous iff for every directed subset \( D \subseteq (\Omega \times A) \),

1. \( f(D) = \{ f(d) | d \in D \} \) is directed and
2. \( f(\bigvee_{\Omega \times A} D) = \bigvee_{A'} f(D). \)

The same result applies to pathwise continuous functions.

### 2.1.2 Metric space

Metric topology is the most direct generalization of the topology used for real numbers in analysis. A metric space \( S \) is a topological space; we call this kind of topology a metric topology. Let us now formally introduce the notions of metric, metric space and metric topology.

**Definition 2.14 (Metric and Metric Space)** Let \( X \) be a set and \( \mathbb{R}^+ \) be the set of non-negative real numbers. A real values function \( d : X \times X \rightarrow \mathbb{R}^+ \) is called a metric or sometimes a distance function on \( X \) iff

- \( d(x, y) = d(y, x). \)
- \( d(x, y) \leq d(x, z) + d(z, y). \)
- \( d(x, y) = 0 \) iff \( x = y. \)

\( (X, d) \) is called a metric space.

Intuitively, a metric space is a set \( X \) with a global distance function (the metric \( d \)) that, for every two points \( x, y \in X \), gives the distance between them as a nonnegative real number \( d(x, y) \).
Definition 2.15 (Metric topology) The metric topology of a metric space is a topology with the set of spherical neighborhoods as a sub-basis, where the spherical \( e \)-neighborhood of \( x \) is \( \{ x' | d(x', x) < e \} \), denoted \( N^e(x) \), for \( (X, d) \) a metric space, \( x \in X \) and \( e \) a positive real number.

Proposition 2.11 Metric topologies are Hausdorff.

In a metric space, \( U \) is open if for every \( x \) in \( U \) we may find an \( \epsilon > 0 \) such that \( D_\epsilon(x) \) is also contained in \( U \).

Another important concept used in analysis is measure. However, before introducing the notion of measure formally, we will present the concepts of algebra and \( \sigma \)-algebra, on which the definition of measure is dependent.

Definition 2.16 (Algebra) Let \( \Omega \) be an abstract set. A collection \( \mathcal{F} \) of subsets of \( \Omega \) is called an algebra on \( \Omega \) if

(i) \( \Omega \in \mathcal{F} \),

(ii) \( F \in \mathcal{F} \Rightarrow F^c := \Omega \setminus F \in \mathcal{F} \), where \( \setminus \) denotes set difference.

(iii) \( F, G \in \mathcal{F} \Rightarrow F \cup G \in \mathcal{F} \)

Note that from (i), (ii) we obtain that \( \emptyset \in \mathcal{F} \) and from (ii), (iii) we obtain that:

\[ F, G \in \mathcal{F} \Rightarrow F \cap G \in \mathcal{F}. \]

Thus, an algebra on \( \Omega \) is simply a family of subsets of \( \Omega \) closed under finitely many set operations (\( \setminus, \cup, \cap \)).

Definition 2.17 (\( \sigma \)-algebra) A collection \( \mathcal{F} \) of subsets of \( \Omega \) is called a \( \sigma \)-algebra on \( \Omega \) if \( \mathcal{F} \) is an algebra on \( \Omega \) such that if \( F_n \in \mathcal{F} \), \( n \in \mathbb{N} \) then

\[ \bigcup_n F_n \in \mathcal{F}. \]

\( \sigma \)-algebras are used to define measures and integration.
Once again, it can be shown that $\cap_n F_n \in \mathcal{F}$. Thus, a $\sigma$-algebra on $\Omega$ is a family of subsets of $\Omega$ closed under any countable collection of set operations.

Let us now define the notion of filtration, which arises when looking at sequences of increasing $\sigma$-algebra.

**Definition 2.18** A family $\mathcal{M} = \{M_t\}_{t \geq 0}$ of $\sigma$-algebra $M_t \subseteq \mathcal{F}$ is a filtration on $(\Omega, \mathcal{F})$ if $\{M_t\}$ is increasing, i.e., $0 \leq s < t \Rightarrow M_s \subseteq M_t$.

It is natural to wonder what is the difference between topology and $\sigma$-algebra as both definitions appear very similar. In fact, the definitions of topology and $\sigma$-algebra are different. Topology does not require closure under complement operation. The most important difference is that open sets are defined based on topology, rather than $\sigma$-algebra while measure is defined based on $\sigma$-algebra, rather than topology. Note however, that set theory is applicable in both topology and $\sigma$-algebra.

Now let us introduce the notion of set functions along with some important properties.

**Definition 2.19 (Additive and Countably Additive Set Functions)** Let $\Omega$ be a set, let $\Sigma_0$ be an algebra on $\Omega$, and let $\gamma$ be a non-negative set function defined as:

$$\gamma : \mathcal{F} \rightarrow [0, \infty].$$

Then $\gamma$ is said to be additive if $\gamma(\emptyset) = 0$ and for $F,G \in \mathcal{F}$, we have

$$F \cap G = \emptyset \Rightarrow \gamma(F \cup G) = \gamma(F) + \gamma(G).$$

The mapping $\gamma_0$ is furthermore called countably additive if $\gamma$ is additive and whenever $(F_n : n \in \mathbb{N})$ is a sequence of disjoint sets in $\mathcal{F}$ with the union $F = \bigcup F_n$ in $\mathcal{F}$, then

$$\gamma(F) = \sum_n \gamma(F_n).$$

**Definition 2.20 (Measurable space)** A pair $(\Omega, \mathcal{F})$, where $\Omega$ is a set and $\mathcal{F}$ is a $\sigma$-algebra on $\Omega$, is called a measurable space.

**Definition 2.21 (Measure space)** Let $(\Omega, \mathcal{F})$ be a measurable space, with $\mathcal{F}$ being a $\sigma$-algebra on $\Omega$. The mapping

$$\gamma : \mathcal{F} \rightarrow [0, \infty]$$

\footnote{We are not requiring that $\mathcal{F}$ be a $\sigma$-algebra, thus this has to be assumed.}
is called a measure on \((\Omega, \mathcal{F})\) if \(\gamma\) is countably additive. The triple \((\Omega, \mathcal{F}, \gamma)\) is called a measure space.

**Definition 2.22 (Borel \(\sigma\)-algebra)** Let \((X, \tau)\) be a topological space. \(\mathcal{B}(X)\), the Borel \(\sigma\)-algebra on \(X\) is the smallest \(\sigma\)-algebra generated by the family of open subsets of \(X\) which contains \(\tau\). The members of \(\mathcal{B}(X)\) are called the Borel sets of \(X\).

**Example 2.1** Let us define \(\mathcal{B} := \mathcal{B}(\mathbb{R})\). The \(\sigma\)-algebra \(\mathcal{B}\) is perhaps the most important of all \(\sigma\)-algebras. Every subset of \(\mathbb{R}\) used in practical applications is an element of \(\mathcal{B}\). □

Now that we have introduced the notion of measure and measure space, we are ready to define the important notion of probability measure.

**Definition 2.23 (Probability measure)** The measure \(P\) is called a probability measure if

\[
P(\Omega) = 1,
\]

and the triple \((\Omega, \mathcal{F}, P)\) is referred to as a probability space.

Therefore, a probability measure is a real-valued function defined on the \(\sigma\)-algebra \(\mathcal{F}\) which assigns a value between 0 and 1 to each set \(A\) which is a member of \(\mathcal{F}\) \((A \in \mathcal{F})\).

### 2.1.3 Random Variables

Equipped with the above notions of probability, measure and topological theory, we are now able to formally construct a random variable (which will obviously be of great importance for the PCN framework), but first let us introduce the key notion of measurable functions.

**Definition 2.24 (Measurable function)** A function \(f : \Omega_1 \to \Omega_2\) of two measurable spaces \((\Omega_1, \mathcal{F}_1)\) and \((\Omega_2, \mathcal{F}_2)\) is called measurable if \(f^{-1}(\mathcal{F}_2) \subseteq \mathcal{F}_1\). If \(\Omega_1\) and \(\Omega_2\) are topological spaces, we call \(f : \Omega_1 \to \Omega_2\) Borel-measurable if it is measurable with respect to the Borel \(\sigma\)-algebra of \(\Omega_1\) and \(\Omega_2\).

An important result, which will prove useful when proving properties of PCNs, arises from Borel sets and continuous functions:
Proposition 2.12  Let \((X, \mathcal{B})\) be a measurable space. Every continuous mapping of \(X\) is Borel measurable.

Definition 2.25 (Random variable) Let \((\Omega, \mathcal{F})\) be a measurable space. A function \(X : (\Omega, \mathcal{F}) \to \mathbb{R}\) is a random variable if for every subset \(A \subseteq \Omega\) of the form

\[ A = \{ \omega : X(\omega) \leq \xi \}, \xi \in \mathbb{R} \]

is an element of the \(\sigma\)-algebra \(\mathcal{F}\).

By definition of a random variable \(X\), all sets of the form

\[ A = \{ \omega : x(\omega) \leq \xi \} \]

have well-defined probabilities. The probabilities are well-defined since \(A \subset \Omega\) and \(A \in \mathcal{F}\), and as defined above, \(P\) assigns probabilities to all such sets \(A\). Therefore, the existence of such a probability function is guaranteed.

Definition 2.26 (Probability Distribution Function) A real-valued function associated with a random variable \(X\) and defined by

\[ F_X(\xi) = P(\{ \omega : X(\omega) \leq \xi \}) \]

is called the (cumulative) probability distribution function. We note that probability distribution functions are by definition monotonic non-decreasing.

The probability distribution function is a basic entity associated with any random variable that allows us to generate probabilities of sets of interest. By definition of random variables, we are assured of its existence. However, we are not assured of the existence of its derivative everywhere, but if it does exist, it is often easier to use and more revealing in terms of graphical interpretation.

If a scalar-valued function \(f^X(\cdot)\) exists such that

\[ F_x(\xi) = \int_{-\infty}^{\xi} f^x(\rho)d\rho \]

holds for all values of \(\xi\), then this function is the probability density function of \(X\). We are not guaranteed of the existence of this function, but if \(F_x(\cdot)\) is absolutely continuous, then it exists. Furthermore, one can show that the probability that \(X(\omega) = x\) lies in any set \(A\) is then

\[ P(\{ \omega : X(\omega) \in A \}) = \int_A f^x(\xi)d\xi. \]
Now let us review what can be obtained through a random variable mapping $X$. We saw that $X$ maps $\Omega$ into $\mathbb{R}$ such that each irreducible set in $\Omega$ maps into a value in $\mathbb{R}$. Thus, the sets of interest in $\mathbb{R}$ will be elements in the Borel $\sigma$-algebra $\mathcal{F}_B$ associated with $\mathbb{R}$. For all sets $A \subset \mathbb{R}$ and $A \in \mathcal{F}_B$, we can define probabilities through $P_x(A) = \int_A f_x(\xi) d\xi$, where $P_x(\cdot)$ is the probability function (Borel measure) associated with $\mathbb{R}$. Therefore, we now have a new probability space, $(\mathbb{R}, \mathcal{F}_B, P_x)$, generated by the mapping $X$ from the original probability space:

$$(\Omega, \mathcal{F}, P) \xrightarrow{X} (\mathbb{R}, \mathcal{F}_B, P_x)$$

Thus, one can quite often neglect the original probability space and describe a problem conveniently in terms of the new probability space $(\mathbb{R}, \mathcal{F}_B, P_x)$.

**Functions of Random Variables**

In the previous section we introduced the concept of random variable as a measurable function from $\Omega$ to $\mathbb{R}$. When modeling probabilistic dynamical systems, many inputs are in fact random variables and it is important to understand the effects of such inputs on the resulting outputs of the system. Therefore, we believe that the notion of function of random variables deserves to be introduced in some detail.

Let assume that $X$ is a vector random variable that maps the sample space $\Omega$ into the $n$-dimensional Euclidean space $\mathbb{R}^n$. Now consider a continuous mapping $\theta(\cdot)$ from $\mathbb{R}^n$ into $\mathbb{R}^m$, thus generating a vector $y \in \mathbb{R}^m$ from a vector $x \in \mathbb{R}^n$. This mapping can be out of a more general class of functions than the continuous functions, referred to as the Baire functions (Borel measurable functions), composed of continuous functions and limits of continuous functions.

Now let us define the $m$-dimensional function $\phi$ as the composite mapping $\phi := \theta[X(\cdot)]$. Then $\phi$ is itself a random variable. It can be shown that every Baire function of a random variable is also a random variable.

Recall from the previous section that $X$ generates a new probability space $(\mathbb{R}^n, \mathcal{F}_B, P_x)$ from the original probability space $(\Omega, \mathcal{F}, P)$. If $\theta$ is a measurable function on $\mathbb{R}^n$, then for every set of interest $A$ in the range space $\mathbb{R}^m$, the inverse image in $\mathbb{R}^n$, $x \in \mathbb{R}^n : \theta(x) \in A$, is an event for which probability has been defined through $P_x$. If we were to view $(\mathbb{R}^n, \mathcal{F}_B, P_x)$ as the underlying probability space, then this just defines $\theta(\cdot)$ itself as a random variable mapping from the sample.
space $\mathbb{R}^n$ into the space $\mathbb{R}^m$. Thus we would get

$$(\Omega, \mathcal{F}, P) \overset{X(\cdot)}{\longrightarrow} (\mathbb{R}^n, \mathcal{F}_B, P_x) \overset{\theta(\cdot)}{\longrightarrow} (\mathbb{R}^m, \mathcal{F}_B, P_\phi)$$

Then we can directly map the original probability space into the new probability space:

$$(\Omega, \mathcal{F}, P) \overset{\phi(\cdot)=\theta[X(\cdot)]}{\longrightarrow} (\mathbb{R}^m, \mathcal{F}_B, P_\phi)$$

Therefore, it is obvious to see that the variable $\phi$ has a distribution induced by the distribution of $X$.

$$F_\phi(\xi) = P(\omega : \phi(\omega) \leq \xi) = P_x(x : \theta(x) \leq \xi)$$

Finishing up this section, we define the concept of limits. For uncertain dynamical systems, we very often have a (finite or infinite) sequence $X_1, X_2, \ldots, X_n$ of random variables and are interested in their asymptotic behaviour, that is in the existence of a random variable $X$ which is the limit of the $X_n$ in some sense. There are several different ways in which such a convergence can be defined. Broadly speaking these fall in two classes, one in which the realizations (which will be defined as stochastic traces shortly) of $X_n$ are required to be close in some way to those of $X$ and another one in which only their probability distribution need to be close. In this body of work, we will consider the latter class of convergence, and more specifically convergence in distribution.

Let us first formally define the notion of convergence in distribution.

**Definition 2.27 (Convergence in Distribution)**

An infinite sequence of random variables $X_1, X_2, \ldots, X_n, \ldots$ is said to converge in distribution to a random variable $X$ if

$$\lim_{n \to \infty} F_{X_n}(x) = F_X(x), \text{ at all continuity points of } F_X,$$

where $F_X$ is the probability distribution function of the random variable $X$. This type of convergence is also known as convergence in law.

Given any sample space $\Omega$, linear order $L$ and topological space $X$, $\nu : \Omega \times L \rightarrow X$ is called a stochastic linear set of values. A limit of $\nu$ is defined as a generalization of the convergence in distribution of a sequence.
Definition 2.28 (Limit in distribution) Let $X$ be a topological space, $\Omega$ a sample space, and $v : \Omega \times L \rightarrow X$ be a stochastic linear set of values. A random variable $v^* : \Omega \rightarrow X$ is called a limit in distribution of $v$, written $F_v \rightarrow F_{v^*}$, if $\lim_{t \to \infty} P(v(t) \leq x) = P(v^* \leq x)$ for all continuity point $x \in X$ of $F_X$.

If $L$ has a greatest element $l_0$, then $F_v \rightarrow F_{v(l_0)}$. Therefore, the concept of limits in distribution is also a generalization of the “final” value in distribution. We will use $\lim_{t \to \infty} F_{v(t)}$ to denote the limit in distribution of $v$ if it is unique. For Hausdorff topologies, the following proposition shows that limits in distribution are in fact unique.

**Proposition 2.13** If $X$ is of a Hausdorff topology and $v : \Omega \times L \rightarrow X$ is a stochastic linear set of values, then $F_v \rightarrow F_{v_1}$ and $F_v \rightarrow F_{v_2}$ imply $F_{v_1} = F_{v_2}$.

Note that fully deterministic linear set of values, that for which $|\Omega| = 1$, are simply special cases of the general stochastic linear set values introduced above. In this case, the limit in distribution simplifies to the usual notion of limit of a sequence (the weight of the limiting distribution is concentrated at the value of that limit). Moreover, we can show that for Hausdorff topologies, these limits are unique and possess the point-wiseness property [Zha94].

**2.1.4 White Noise, Brownian Motion and Stochastic Integrals**

For continuous time systems, it has become the norm for dynamical systems to be modeled by a set of differential equations of the general form $dX/dt = b(t, X_t)$. However, since we are interested in explicitly modeling the uncertainty inherent in these systems, we need to find a suitable mathematical interpretation of the *noise* that will be included in models with continuous time structures.

Consider the following differential equation augmented with an uncertainty term denoted by $W_t$:

$$\frac{dX}{dt} = b(t, X_t) + g(t, X_t) \cdot W_t \quad (2.1)$$

It seems reasonable to assume that any stochastic process $W_t$ that represents the uncertainty term in Equation 2.1 will have the following properties:

1. $W_t$ is independent of $W_s$ whenever $t \neq s$. 

40
2. \( E(W_t) = 0 \) for all \( t \).

3. The joint distribution of \( \{W_{t_1+t}, \ldots, W_{t_k+t}\} \) is independent of the value \( t \). This is equivalent to stating that the distribution \( \{W_t\} \) is stationary.

The assumptions (1), (2) and (3) suggest that \( W_t \) should have *stationary independent increments with mean 0*. However, it can be shown that there does not exist any *reasonable* stochastic process which satisfy (1) and (3) in continuous time. Such a process cannot have continuous paths [Oks98]. In fact, Kallianpur showed that if we require \( E(W_t^2) = 1 \), then \( W_t \) cannot even be measurable with respect the \( \sigma \)-algebra \( B \times \mathcal{F} \) [Kal80].

Although it is possible to represent \( W_t \) as a *white noise* process which is constructed as a probability measure on the space \( S' \) of tempered distribution on \([0, \infty)\) [Hid80, Adl81, Roz82], it is more common to avoid this type of construction and instead rewrite Equation 2.1 in a form that allows for the replacement of \( W_t \) with a proper stochastic process. Based on this, we can show that the only process with continuous paths is Brownian motion \( B_t \) [Kni81]. The Brownian motion process is also referred to as the *Wiener* process.

Using the usual integration notation, we can rewrite Equation 2.1 to obtain an equation of the form

\[
X_t = X_0 + \int_0^t b(s, X_s)ds + \int_0^t g(s, X_s)dB_s
\]  

(2.2)

where \( B_t(\omega) \) is a 1-dimensional Brownian motion starting at the origin and where it is assumed that the stochastic process \( X_t = X_t(\omega) \) of Equation 2.1 is the solution to Equation 2.2 where an appropriate meaning for the last integral is provided.

Brownian motion is independent of each of its time realizations which can let us think of Gaussian white noise as the formal derivative of Brownian Motion. It is well known that \( E(B_t^2) = t \) and \( E(B_0^2) = 0 \) so that \( E(B_{t+\delta} - B_t)^2 = t + \delta - 2t + t = \delta \). We can see that \( B_{t+\delta} - B_t \) is roughly of the order of \( \sqrt{\delta} \) and thus we do not expect the sample paths of a Brownian motion to be almost surely *differentiable* anywhere [Bre68]. Actually, this reflects the fact that the path variations of \( B_t \) are too big to enable us to define the rightmost integral of Equation 2.2 in the Riemann-Stieltjes sense. In particular, the total variation of the path of a Brownian motion is infinite.

It can be shown that unlike the Riemann-Stieltjes integral for deterministic functions, when calculating the value of the rightmost integral of Equation 2.2, the choice of the interval end points
affect the final result. The key difference between the results of the Itô integral and the result in a Riemann-Stieltjes integral from real analysis is that the integrand is not evaluated on any point \( t' \in [t_i, t_{i+1}] \) but precisely at the left endpoint \( t_i \) of the interval. The value of a stochastic integral changes with the choice of the point \( t' \) where the integrand is evaluated. Choosing the mid point \( (t_i + t_{i+1})/2 \) leads to the Stratonovich integral, and in general the result is different from the Itô integral.

**Example 2.2 Comparing Itô and Riemann-Stieltjes**

Let us now compare the results of the Riemann-Stieltjes and Itô interpretations in calculating \( \int_0^T B_t dB_t \). When considering the integral as a the Riemann-Stieltjes integral the result is simply \( 1/2B_T^2 \). However, the results under the Itô interpretation, is \( 1/2B_T^2 - 1/2T \) while the result for the Stratonovich interpretation is the same as the Riemann-Stieltjes, i.e., \( 1/2B_T^2 \). □

Based on these reflections, a natural question that one might ask would be: Which interpretation of "\( \int_0^t g(s, X_s)dB_s \)" makes Equation 2.2 the correct mathematical model for Equation 2.1?

Although there are some cases for which the Stratonovich interpretation is more reasonable, the specific feature of the Itô model of not looking into the future, as apparent from the selection of the end points of its calculation, is a powerful reason to select the Itô paradigm when in the presence of real-time dynamical systems. For a similar, more in depth discussion about why Itô is more appropriate for biological systems, the reader is referred to [Tur77]. Moreover, as Itô integrals are Martingales [Oks98], this leads to an important computational advantage over the Stratonovich interpretation as Martingales are the stochastic analog of conservations laws: expectation is "conserved", i.e., for a filtration \( \mathcal{F}_t \), \( E(X_{t+1} | \mathcal{F}_t) = X_t \), \( \forall t \). Hence, in this dissertation, we will adopt the Itô interpretation for its convenience and computational advantage. Furthermore, it can be showed that the solution of Equation 2.2 obtained via the Stratonovich interpretation is identical to the solution obtained from the modified Itô equation

\[
X_t = X_0 + \int_0^t b(s, X_s)ds + \frac{1}{2} \int g'(s, X_s)g(s, X_s)ds + \int_0^t g(s, X_s)dB_s
\]  

(2.3)

where \( g' \) is the derivative of \( g(t, x) \) with respect to \( x \) [Str66]. Hence, due to this explicit connection between the two types of integrals, the choice of paradigm is not as critical as it may appear since one can always jump back and forth between the Itô and Stratonovich interpretation with a simple modification.
Let us now enumerate some important conditions of stochastic process. These conditions will prove essential when discussing the existence and uniqueness of stochastic differential equations (SDEs) in § 4.3.2. We first define the notion of $\sigma$-algebra generated by a random variable and the concept of adaptiveness.

**Definition 2.29** Let $\mathcal{F}_t = \mathcal{F}_t^t$ denote the $\sigma$-algebra generated by the random variables $B_s(\cdot)$, $s \leq t$, where $B_t(\omega)$ is the $n$-dimensional Brownian motion. Hence, $\mathcal{F}_t$ is the smallest $\sigma$-algebra that contains all the sets of the form $\{\omega; B_{t_1}(\omega) \in F_1, \ldots, B_{t_k}(\omega) \in F_k\}$, with $t_j \leq t$ and $F_j \subset \mathbb{R}^n$ being Borel sets, $j \leq k = 1, 2, \ldots$.

Intuitively, $\mathcal{F}_t$ can be seen as denoting the history of $B_s$ up to time $t$. Hence, a random function $r(\omega)$ is $\mathcal{F}_t$-measurable if and only if $r$ is equivalent to the pointwise limit of sums of functions of the form $g_1(B_{t_1})g_2(B_{t_2}) \cdots g_k(B_{t_k})$, where the functions $g$ are bounded continuous functions and $t_j \leq t$ for $j \leq k$, $k = 1, 2, \ldots$. For example, the function $r_1(\omega) = \frac{1}{4}B_t(\omega)$ is $\mathcal{F}_t$-measurable while $r_2(\omega) = B_{3t}(\omega)$ is clearly not.

**Definition 2.30** Assume $\{\mathcal{F}_t\}_{t \geq 0}$ to be an increasing family of $\sigma$-algebra of subsets of $\Omega$. We call $\mathcal{F}_t$-adapted, a process $f(t, \omega) : [0, \infty) \times \Omega \to \mathbb{R}^n$ if, for each $t \geq 0$, the function $\omega \mapsto f(t, \omega)$ is $\mathcal{F}_t$-measurable.

Hence, it is easy to show that $r_1(\omega) = B_{t/4}(\omega)$ is $\mathcal{F}_t$-adapted, unlike $r_2(\omega) = B_{3t}(\omega)$.

### 2.2 Time Structures

As we are modeling dynamical systems, a model of time and its evolution is necessary. In fact, a clear notion of the concept of time is central to understanding dynamics. As it was done within the CN framework, we formalize time using an abstract structure that captures its most important properties. In general, a time structure can be considered as a totally ordered set with an initial start time, an associated metric for "the distance between any two time points" and a measure for "the duration of an interval of time." Formally, we define the concept of time structure as follows.

**Definition 2.31** (Time structure) A time structure is a triple $\langle T, d, \mu \rangle$ where

- $T$ is a linearly ordered set $\langle T, \leq \rangle$ with 0 as the least element;
• \((T, d)\) forms a metric space with \(d\) as a metric satisfying: for all \(t_0 \leq t_1 \leq t_2\),

\[
d(t_0, t_2) = d(t_0, t_1) + d(t_1, t_2),
\]

\(\{t \mid m(t) \leq \tau\}\) has a greatest element and \(\{t \mid m(t) \geq \tau\}\) has a least element for all \(0 \leq \tau < \sup \{m(t) \mid t \in T\}\) where \(m(t) = d(0, t)\):

• \((T, \sigma, \mu)\) forms a measure space with \(\sigma\) as the Borel set of topological space \((T, d)\) and \(\mu\) as a Borel measure satisfying \(\mu([t_1, t_2]) \leq d(t_1, t_2)\) for all \(t_1 \leq t_2\) where \([t_1, t_2) = \{t \mid t_1 \leq t < t_2\}\) and \(\mu([t_1, t_2)) = \mu([0, t_2)) - \mu([0, t_1))\).

To abridge the notation, we will simply use \(T\) to refer to the time structure \((T, d, \mu)\) when no ambiguity arises. In general, we have \(\mu([t_1, t_2)) = d(t_1, t_2)\). However, when \(T\) is an abstraction of another time structure, it is possible that \(\exists t_1, t_2, \mu([t_1, t_2)) < d(t_1, t_2)\). Discussions on time abstraction can be found in Chapter 6, Behavior Analysis of [Zha94].

We will refer to a time structure \(T\) as being infinite iff \(T\) has no greatest element and \(\mu(T) = \infty\). Moreover, the time structure \(T\) is continuous iff its metric space is connected while it is discrete iff its metric topology is discrete.

**Example 2.3** Consider the set of natural numbers \(\mathbb{N}\) and the set of nonnegative real numbers \(\mathbb{R}^+\) along with the metric \(d(t_1, t_2) = |t_1 - t_2|\) and the measure \(\mu([0, t)) = t\). \(\mathbb{N}\) and \(\mathbb{R}^+\) respectively define discrete and continuous time structures. The set \(\{1 - \frac{1}{2^n} \mid n \in \mathbb{N}\}\) with the metric and measure defined above also defines a discrete time structure. However, the sets \(\{1 - \frac{1}{2^n} \mid n \in \mathbb{N}\} \cup \{1\}, \{0\} \cup \{\frac{1}{2^n} \mid n \in \mathbb{N}\}\) and \([0, 4] \cup [5, 7]\) with the metric \(d\) and the measure \(\mu\) form time structures neither discrete nor continuous. \(\Box\)

Note that the set of rational numbers \(\mathbb{Q}\) with the metric \(d\) and the measure \(\mu\) does not form a time structure. This can be proved using the fact that the set \(\mathbb{Q}\) of rationals lacks the least upper bound property stating that if a set \(S\) has the property that every nonempty subset of \(S\) which has an upper bound also has a least upper bound. This property is summarized in the following proposition.

**Proposition 2.14** Equivalent properties of time structures

1. For any time structure \((T, d, \mu),\) if \(T \subset T\) has an upper bound in \(T,\) \(T\) has a least upper bound in \(T.\)
2. The following properties for a time structure are equivalent:

(a) \( \langle T, d, \mu \rangle \) is discrete.

(b) Let \( \{t_1, t_2\} = \{t | t_1 < t < t_2\} \). For all \( t \), if \( t \) is not the least element of \( T \), then \( \exists t' < t \), denoted \( \text{pre}(t) \), such that \( (t', t) = 0 \), and for all \( t \), if \( t \) is not the greatest element of \( T \), then \( \exists t' > t \), denoted \( \text{suc}(t) \), such that \( (t, t') = 0 \).

(c) \( \langle T, d, \mu \rangle \) is well-founded, i.e., for all \( t \in T \), \( [0, t) \) is finite.

3. The following properties for a time structure are equivalent:

(a) \( \langle T, d, \mu \rangle \) is continuous.

(b) \( \langle T, d, \mu \rangle \) is dense, i.e., for all \( t_1 < t_2 \), there exists \( t_0 \) such that \( t_1 < t_0 < t_2 \).

Now let us discuss the relationship between two different time structures. A time structure \( \langle T, d, \mu \rangle \) can be related to another time structure \( \langle T_r, d_r, \mu_r \rangle \), where \( \langle T_r, \leq_r \rangle \) is a total order with \( 0_r \) as the least element, by a reference time mapping \( h : T \rightarrow T_r \) satisfying the following properties:

- the order among time points is preserved: \( i < t' \) implies \( h(i) <_r h(t') \),

- the least element is preserved: \( h(0) = 0_r \),

- the distance between two time points is preserved: \( d(t_1, t_2) = d_r(h(t_1), h(t_2)) \), and

- the measure on any finite time interval is preserved: \( \mu([0, t)) = \mu_r([0_r, h(t)]) \).

In such a case, we will call \( T_r \) the reference time of \( T \), and \( T \) the sample time of \( T_r \). For example, if \( h : N \rightarrow \mathbb{R}^+ \) is defined as \( h(n) = n \), \( \mathbb{R}^+ \) is a reference time of \( N \). For any time structure \( T \), a reference time of \( T \) is as "dense" as \( T \). Furthermore, it can easily be shown that the reference relation is in fact transitive [Zha94].

Now that we have a formal definition of the concept of time, we wish to formalize the notion of domains, stochastic traces and events in a similar fashion. The definitions associated with the notion of domain are analogous to those introduced in §3.3 of [Zha94]. Since these notions are of importance to a clear understanding of the upcoming sections, we reproduce some results from that section and adapt some to the stochastic case.
2.3 Domain Structures

As with time, we formalize domains as abstract structures so that discrete and continuous domains are defined uniformly. A domain can be either simple or composite. Simple domains denote simple data types, such as reals, integers, Booleans and characters; composite domains denote structured data types, such as arrays, vectors, strings, objects, structures and records.

**Definition 2.32 (Simple domain)** A simple domain is a pair \( \langle A \cup \{\perp_A\}, d_A \rangle \) where \( A \) is a set, \( \perp_A \) means undefined in \( A \), and \( d_A \) is a metric on \( A \).

Let \( \overline{A} = A \cup \{\perp_A\} \). For simplicity, we will use \( \overline{A} \) to refer to simple domain \( \langle \overline{A}, d_A \rangle \) when no ambiguity arises. For example, let \( \mathbb{R} \) be the set of real numbers, \( \overline{\mathbb{R}} \) is a simple domain with a connected metric space; let \( B = \{0, 1\} \), \( \overline{B} \) is a simple domain with a discrete topology on \( B \).

Any simple domain \( \overline{A} \) is associated with a partial order relation \( \leq_{\overline{A}} \). \( \langle \overline{A}, \leq_{\overline{A}} \rangle \) is a flat partial order with \( \perp_A \) as the least element. In addition, \( \overline{A} \) is associated with a derived metric topology \( \tau = \tau_A \cup \{\overline{A}\} \) where \( \tau_A \) is the metric topology on \( A \) derived from the metric \( d_A \).

**Proposition 2.15** \( \{\perp_A\} \) is not \( \tau \)-open. The only neighborhood of \( \perp_A \) is \( \overline{A} \).

A simple domain \( \langle A, d_A \rangle \) can also be represented as a triple \( \langle \overline{A}, \leq_{\overline{A}}, \tau \rangle \) where \( \leq_{\overline{A}} \) is the partial order relation and \( \tau \) is the derived metric topology.

A domain is defined recursively based on simple domains.

**Definition 2.33 (Domain)** \( \langle A, \leq_A, \tau \rangle \), with \( \leq_A \) as the partial order relation and \( \tau \) as the derived metric topology, is a domain iff:

- it is a simple domain; or

- it is a composite domain, i.e., it is the product of a family of domains \( \{\langle A_i, \leq_{A_i}, \tau_i \rangle\}_{i \in I} \) such that \( \langle A, \leq_A \rangle \) is the product partial order of the family of partial orders \( \{\langle A_i, \leq_{A_i} \rangle\}_{i \in I} \) and \( \langle A, \tau \rangle \) is the product space of the family of topological spaces \( \{\langle A_i, \tau_i \rangle\}_{i \in I} \).

Note that there is no restriction on the index set \( I \), which can be arbitrary (finite or infinite, countable or uncountable). For simplicity, we will use \( A \) to refer to domain \( \langle A, \leq_A, \tau \rangle \) when no ambiguity arises. For example, let \( n \) be a natural number, then \( \mathbb{R}^n \) is a composite domain with \( n \) components;
let $N$ be the set of natural numbers, then $N \rightarrow \overline{B}$ (or equivalently, $\overline{B}^N$) is a composite domain with infinitely many components.

Given a simple domain $\overline{A}$, a value $a \in \overline{A}$ is well-defined iff $a \neq \bot_A$. Given a composite domain $\times_I A_i$, a value $a \in \times_I A_i$ is well-defined iff $a_i$ is well-defined for all $i \in I$. A value in a domain is undefined iff it is the least element of the domain.

Intuitively, for any domain, its partial order topology characterizes the information (or defined-ness) hierarchies of data and its derived metric topology characterizes the limit properties of data.

**Proposition 2.16** For any domain, its partial order topology is finer than its derived metric topology, and both are non-Hausdorff.

A signature is a syntactical structure of a multi-sorted set of data with associated functions.

**Definition 2.34 (Signature)** Let $\Sigma = (S, F)$ be a signature where $S$ is a set of sorts and $F$ is a set of function symbols. $F$ is equipped with a mapping type: $F \rightarrow S^* \times S$ where $S^*$ denotes the set of all finite tuples of $S$. For any $f \in F$, type$(f)$ is the type of $f$. We use $f : s^* \rightarrow s$ to denote $f \in F$ with type$(f) = (s^*, s)$.

**Example 2.4 Two Basic Signatures** The signature of an algebra on the Naturals can be denoted by $\Sigma_N = \langle N, \{0, +, -, \times\} \rangle$. This signature has only one sort, $N$, with 4 different function symbols.

The Boolean algebra can be described as: $\Sigma_b = \langle \{b\}, \{0, \neg, \land, \lor\} \rangle$ with $0 \rightarrow b$, $\neg : b \rightarrow b$, $\land : b, b \rightarrow b$, and $\lor : b, b \rightarrow b$. $\Sigma_b$ has one sort with a constant $0$ (nullary function), a unary function $\neg$, and two binary functions $\land$ and $\lor$. □

A domain structure of some signature is defined as follows.

**Definition 2.35 ($\Sigma$-domain structure)** Let $\Sigma = (S, F)$ be a signature. A $\Sigma$-domain structure $A$ is a pair $\langle \{A_s\}_{s \in S}, \{f^A\}_{f \in F} \rangle$ where for each $s \in S$, $A_s$ is a domain of sort $s$, and for each $f : s^* \rightarrow s \in F$ with $s^* : I \rightarrow S$ and $s \in S$, $f^A : \times_I A_s$ $\rightarrow A_s$ is a function denoted by $f$, which is continuous in the partial order topology.

To be continuous on a domain in its partial order topology is not a real restriction on a function. Strict functions are continuous functions in partial order topologies. A function is strict w.r.t. an argument iff its output is undefined whenever the value of that argument is undefined. A function is strict iff it is strict w.r.t. all of its arguments.
Given any partial or total function $f : \Omega \times I A_i \rightarrow A$, a continuous function $\overline{f} : \Omega \times I \overline{A_i} \rightarrow \overline{A}$ can be defined as:

$$\overline{f}(\omega, a) = \begin{cases} 
  f(\omega, a) & \text{if } (\omega, a) \in \Omega \times I A_i \text{ and } f(\omega, a) \text{ is defined} \\
  \bot_A & \text{otherwise.}
\end{cases}$$

We call $\overline{f}$ a strict extension of function $f$. We will also use $f$ to denote its strict extension if no ambiguity arises. For example, let $\Sigma_r = \langle \{r\}, \{0, +, \cdot\} \rangle$ with $0 :\rightarrow r, + : r, r \rightarrow r$ and $\cdot : r, r \rightarrow r$. Then $\langle \{\mathbb{R}\}, \{0, +, \cdot\} \rangle$ is a $\Sigma_r$-domain structure, where $+$ and $\cdot$ are strict extensions of addition and multiplication on $\mathbb{R}$, respectively.

However, not every extension of a function that is continuous should also be strict. For example, $\langle \{\mathbb{B}\}, \{0, \neg, \land, \lor\} \rangle$ is a $\Sigma_b$-domain structure where $\neg$, $\land$ and $\lor$ are negation, conjunction and disjunction, respectively. Function $\lor : \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$ is continuous but not strict, since $\lor$ is an "or" logic satisfying $1 \lor x = 1$ for all $x \in \mathbb{B}$, thus, $1 \lor \bot_\mathbb{B} \neq \bot_\mathbb{B}$.

The following propositions characterize the general properties of continuous functions on simple domains.

**Proposition 2.17** (1) Function $f : \Omega \times \overline{A} \rightarrow \overline{A'}$ is continuous in the partial order topology iff $f$ is strict or constant. (2) If $f : \Omega \times \overline{A} \rightarrow \overline{A'}$ is continuous in the derived metric topology, then $f$ is continuous in the partial order topology. (3) Function $f : \Omega \times \overline{A} \rightarrow \overline{A'}$ is continuous in the derived metric topology iff $f$ is continuous in the partial order topology and the restriction of $f$ on $\Omega \times A$ and $A'$ is continuous in the metric topology, namely, for any open subset $S$ of $A'$, $f^{-1}(S) \cap (\Omega \times A)$ is open.

The properties of continuous functions in partial order topologies can be generalized to composite domains. A function $f : \Omega \times I A_i \rightarrow A$ is continuous w.r.t. an argument $j$, iff function $\lambda a_j f(\omega, a, a_j)$ is continuous for all $(\omega, a) \in \Omega \times I \{j\} A_i$.

**Proposition 2.18** Let $I$ be a finite index set. (1) Function $f : \Omega \times I A_i \rightarrow A$ is continuous in the partial order topology iff $f$ is continuous w.r.t. all $i \in I$. (2) If $f : \Omega \times I \overline{A_i} \rightarrow \overline{A}$ is continuous in the derived metric topology, then $f$ is continuous in the partial order topology. (3) Function $f : \Omega \times I \overline{A_i} \rightarrow \overline{A}$ is continuous in the derived metric topology iff $f$ is continuous in the partial order topology and the restriction of $f$ on $\Omega \times I A_i$ and $A$ is continuous in the product metric topology, namely, for any open subset $S$ of $A$, $f^{-1}(S) \cap (\Omega \times I A_i)$ is open.
A function is well-defined iff its output is well-defined whenever its input is well-defined. Both well-definedness and strictness are closed under functional composition, and a function can be both well-defined and strict.

For example, a widely used conditional function, \( \text{cond} : A \times A \times A' \times A' \rightarrow A' \), is defined as follows:

\[
\text{cond}(x, y, u, v) = \begin{cases} 
\bot_{A'} & \text{if } x = \bot_A \text{ or } y = \bot_A \\
u & \text{else if } x = y \\
v & \text{otherwise.}
\end{cases}
\] (2.4)

Function \( \text{cond} \) is continuous in the partial order topology; it is continuous in the derived metric topology if \( A \) is of a discrete topology. Furthermore, it is well-defined and strict w.r.t. arguments \( x \) and \( y \).

### 2.4 Stochastic Traces and Events

In this section, we define the concept of stochastic trace, an extension of the concept of trace that was introduced in the CN framework. Stochastic traces will be central in representing the dynamical behaviour of the systems modeled within the PCN framework. A stochastic trace intuitively denotes the (random) changes of values over time. Formally, a stochastic trace is a mapping \( v : \Omega \times \mathcal{T} \rightarrow A \) from sample space \( \Omega \) and time domain \( \mathcal{T} \) to value domain \( A \). For a given \( \omega \in \Omega \), the function \( v_\omega : \mathcal{T} \rightarrow A \) is simply called a trace. In the literature, a trace is often referred to as a sample function, a realization, a trajectory or a path of the underlying stochastic process. We will use \( v \) to denote both the stochastic trace \( v \) or one of its realization trace \( v_\omega \) when it is clear from the context and no ambiguity arises.

A stochastic trace \( v \) is well-defined iff \( v(\omega, t) \) is well-defined for all \( (\omega, t) \in \Omega \times \mathcal{T} \). A stochastic trace \( v \) is undefined iff \( v(\omega, t) \) is undefined for any \( (\omega, t) \in \Omega \times \mathcal{T} \). For example, denote a Brownian motion process by \( B_t(\omega) \) and \( \mathcal{T} = \mathbb{R}^+ \) and \( A = \mathbb{R} \). Then have that \( v = \lambda \omega, t.B_t(\omega) \) is a well-defined stochastic trace. For a fixed \( \omega \) in \( \Omega \), \( v_\omega = \lambda t.B_t(\omega) \) represents a path of the Brownian motion process. On the other hand, \( v_1 = \lambda t.\cos(t) \) and \( v_2 = \lambda t.e^{-t} \) are well-defined deterministic traces, i.e., stochastic traces for which \( |\Omega| = 1 \).

Due to the fact that physical systems encompass uncertainty, one is often more interested in the distribution of the system rather than in one specific trace. Hence, rather than merely looking at one
given execution trace, one may pay attention to the distribution of traces of the system.

One important feature of a trace is that it provides complete information about the current execu-
tion of the system of interest at every time point. In the presence of uncertainty, the limiting value
of a specific execution trace \( v_\omega \) is of little interest since the measure of that trace is typically zero.
The distribution of a stochastic trace, on the other hand, provides complete information about the
probability of the state of the system at every finite time point.

Although trace distribution values at infinite time points are not represented explicitly, they can
be derived when limits (in distributions) are introduced. The limiting distribution of a stochastic
trace can provide useful information when assessing the behaviour of the system in the long run. For
example, consider the stochastic trace associated to the system denoted by \( f : \Omega \times \mathbb{R}^+ \rightarrow \mathbb{R}^+ \), where
\[ f(\omega, t) = 1 + B_t(\omega)e^{-t}, \]
with \( B_t(\omega) \) a Brownian motion process. For each value of \( t \), one can easily
show that \( f \) follows a Gaussian distribution with mean 1 and variance \( te^{-2t} \) \((F_f = \mathcal{N}(1, te^{-2t}))\). The limiting distribution is hence \( \lim_{t \to \infty} \mathcal{N}(1, te^{-2t}) = \mathcal{N}(1, 0) \), which indicates that in the long
run, the system will converge to value 1 and will not fluctuate away from it, despite being influenced
by a Brownian motion with increasing variance.

Let \( A \) be a domain, \( L \) a linear order and \( v : \Omega \times L \rightarrow A \) be a stochastic linear set of values. The
distribution of a random variable \( v^* : \Omega \rightarrow A \) is a limit in distribution of \( v \), written \( F_v \rightarrow F_{v^*} \), iff
\[ P(v^* \leq a) = \lim_{t \to \infty} P(v(t) \leq a) \]
for any \( a \in A \).

Limits in distribution of \( v \) may not be unique. However, the set of limits in distribution of \( v \) has
the following properties.

**Proposition 2.19** Let \( v : \Omega \times L \rightarrow A \) be a stochastic linear set of values. Then

1. \( F_v \rightarrow F_{\bot A} \), and
2. \( F_v \rightarrow F_{v_1^*} \) and \( F_v \rightarrow F_{v_2^*} \) imply that either \( F_{v_1^*} = F_{v_2^*} \) or one of \( F_{v_1^*} \) and \( F_{v_2^*} \) is \( F_{\bot A} \).

Now equipped with the concept of limit in distribution, we can complete a stochastic trace with
its distribution at limit time points. Given a time structure \( T \), let \( T^\infty \) be the set of downward closed
intervals, i.e., for any \( T \in T^\infty \), (1) \( T \neq \emptyset \) and (2) \( t \in T \) implies that for all \( t' \leq t, t' \in T \).
A stochastic trace \( v : \Omega \times T \rightarrow A \) can be extended to its completion \( v^\infty : \Omega \times T^\infty \rightarrow A \) as
\[ v^\infty(T) = \lim_{t \uparrow T} v|_T \]
where \( v|_T \) denotes the restriction of \( v \) onto \( T \). If \( T \) has a greatest element \( t_0 \),
then \( F_{v^\infty(T)} = F_{v(t_0)} \). A trace completion provides distributions at infinite as well as at finite time
points. Note that \( T \in T^\infty \), for any stochastic trace \( v : \Omega \times T \rightarrow A \), \( v^\infty(T) = \lim v \) can be
considered as the “final” value. For simplicity, we will use \( v \) to refer to both \( v \) and its completion \( v^\infty \) when no ambiguity arises.

Let us introduce notation that will prove helpful for the reminder of this dissertation. Let \( T_{<t} = \{ t' | t' < t \} \). Then \( T_{<t} \subseteq T^\infty \) whenever \( t > 0 \). We use \( \text{pre}(t) \) to denote both \( T_{<t} \) and the greatest element of \( T_{<t} \), if it exists. Let \( T_{\leq t-\tau} = \{ t' | t' < t, d(t, t') \geq \tau \} \) for \( \tau > 0 \). Then \( T_{\leq t-\tau} \subseteq T^\infty \) whenever \( m(t) \geq \tau \). Moreover, it can be shown that for any time structure \( T \), \( T_{\leq t-\tau} \) has a greatest element whenever \( m(t) \geq \tau \) [Zha94]. To denote the greatest element of \( T_{\leq t-\tau} \) when \( m(t) \geq \tau \), we will use \( t - \tau \).

We now define the notion of stochastic trace space, formed from the set of all possible stochastic traces.

**Definition 2.36 (Stochastic Trace Space)** Given a time structure \( T \) and a domain \( (A, \leq_A, \tau) \), the stochastic trace space is a triple \( (A^{\Omega \times T}, \leq_{A^{\Omega \times T}}, \Gamma) \) where \( A^{\Omega \times T} \) is the product set (the set of all functions from \( \Omega \times T \) to \( A \)), \( \leq_{A^{\Omega \times T}} \) is the product partial order relation constructed from the partial order relation \( \leq_A \), and \( \Gamma \) is the product topology constructed from the derived metric topology \( \tau \).

For a fixed \( \omega \in \Omega \), the triple \( (A^{\omega \times T}, \leq_{A^{\omega \times T}}, \Gamma) \) simplifies to what we will call a trace space. Once again, to abridge the notation, we will refer to the trace space \( (A^{\Omega \times T}, \leq_{A^{\Omega \times T}}, \Gamma) \) as simply \( A^T \), when no ambiguity arises. A stochastic trace space is essentially a composite domain. Therefore limits in the distribution of a linear set of stochastic traces can be defined accordingly. The reader is referred to §3.4 of [Zha94] for a formal description of the results in the deterministic cases. The extension to the stochastic case is straightforward.

We now present the definitions of non-intermittent and right-continuous stochastic traces, two special types of stochastic traces.

**Definition 2.37 (Non-intermittent stochastic trace)**
A trace \( v_\omega : T \rightarrow A \) is path-wise non-intermittent iff for a fixed \( \omega \) and for any \( T \in T^\infty \), \( v(T) = \bot_A \) implies that \( \forall T' \supseteq T, v(T') = \bot_A \). A trace \( v_\omega : T \rightarrow \times_i A_i \) is path-wise non-intermittent iff \( v_i \) is path-wise non-intermittent for all \( i \in I \). A stochastic trace \( v : \Omega \times T \rightarrow \bar{A} \) is non-intermittent if \( v \) is path-wise non-intermittent for every \( \omega \) in \( \Omega \).
Definition 2.38 (Right-continuous stochastic trace)

A right-continuous stochastic trace is a special type of trace defined as follows. A trace $v_\omega : T \rightarrow A$ is right-continuous at $t_0$ iff $\forall t > t_0, t \rightarrow t_0$ implies $v_\omega(t) \rightarrow v_\omega(t_0)$; $v_\omega$ is right-continuous iff it is right-continuous at all $t \in T$. A stochastic trace is right-continuous if its execution traces $v_\omega$ are path-wise right-continuous for all $\omega$ in $\Omega$. A discrete-time stochastic trace is always right-continuous according to this definition.

A stochastic event trace is a non-intermittent and right-continuous stochastic trace whose domain is $[\mathbb{B}]$. For each $\omega \in \Omega$, an event trace $e_\omega : T \rightarrow [\mathbb{B}]$ with $e_\omega \neq \lambda t$. $\bot_\mathbb{B}$ generates a structure $(T_{e_\omega}, d_{e_\omega}, \mu_{e_\omega})$ from $(T, d, \mu)$ where

- $T_{e_\omega} \subseteq T$ is defined as $T_{e_\omega} = \{0\} \cup \{t > 0 | e_\omega(t) \neq \bot_\mathbb{B}, e_\omega(t) \neq e_\omega(pre(t))\}$,
- $d_{e_\omega} = d_{T_{e_\omega} \times T_{e_\omega}}$,
- $\forall t \in T_{e_\omega}, \mu_{e_\omega}(\delta_{0,t}) = \mu(\delta_{0,t})$, and $\mu_{e_\omega}(T_{e_\omega}) = \mu(T)$ for $T = \{t | e_\omega(t) \neq \bot_\mathbb{B}\}$.

Proposition 2.20 For any time structure $T$ and any event trace $e_\omega$, $(T_{e_\omega}, d_{e_\omega}, \mu_{e_\omega})$ is a discrete sample time structure of $T$.

For any event-based time, each transition point of the event trace defines a time point (Figure 2.1).

Similarly to stochastic traces and stochastic trace spaces, the set of all possible stochastic event traces on a reference time structure, associated with a partial order relation and a derived metric topology, forms a stochastic event space.
Definition 2.39 (Stochastic event space) A stochastic event space is a triple \( (\mathcal{E}^{\Omega \times T}, \leq_{\mathcal{E}^{\Omega \times T}}, \Gamma') \) where \( T \) is a time structure, \( \mathcal{E}^{\Omega \times T} \subseteq \mathbb{B}^{\Omega \times T} \) is the set of all stochastic event traces on \( \Omega \times T \), \( \leq_{\mathcal{E}^{\Omega \times T}} \) is the sub partial order relation of \( \leq_{\mathbb{B}^{\Omega \times T}} \), and \( \Gamma' \) is the subspace topology of \( \Gamma \) that is the derived metric topology of \( \mathbb{B}^{\Omega \times T} \).

Similarly to stochastic trace spaces, for fixed \( \omega \) in \( \Omega \), we will call the triple \( (\mathcal{E}^{\omega \times T}, \leq_{\mathcal{E}^{\omega \times T}}, \Gamma') \), an event space \( \mathcal{E}^T \).

2.5 Transductions

Within the PCN paradigm, transductions will dictate the evolution, and hence the behaviour, of the modeled systems. Intuitively, transductions are mathematical models of general transformational processes. In PCN, transductions are similar to functions of random variables. For a given input, the output is unpredictable other than it is known to obey the probability distribution function resulting from the application of the transduction to the random variable. The concept of transduction is at the core of the ability of PCN to model the inherent uncertainty in the systems under study.

We will characterize two classes of transduction: primitive transductions and event-driven transductions. These classes will be constructed from the functional composition of three types of basic transductions: transliterations, delays and generators. Generators will be used to represent random variables which will introduce uncertainty in the model. Finally, we will introduce the notion of event-driven transductions. Event-driven transductions will prove essential to construct models of temporally hybrid systems, i.e., systems encompassing different time structures. Note that deterministic transductions will be seen as a simple case of a transduction: one with no random variable as input.

2.5.1 Transductions: general concepts

Formally, a transduction is a mapping from input stochastic traces to output stochastic traces that satisfies the causal relationship between its inputs and outputs, i.e., the output value at any time depends only on inputs up to and including that time. The causal relationship stipulates that the evolution of the system cannot be dictated by the future state of the system, but only by past and present values. Formally, causality can be defined as follows
Definition 2.40 (Causality via $\mathcal{F}_t$-adaptedness) Assume $\{\mathcal{F}_t\}_{t \geq 0}$ to be an increasing family of $\sigma$-algebra of subsets of $A^{\Omega \times T}$. A mapping $F(v)(\omega, t) : A^{\Omega \times T} \rightarrow A^{\Omega \times T'}$ is causal if $F(v)(\omega, t)$ is $\mathcal{F}_t$-adapted. A causal mapping on stochastic trace spaces is called a transduction.

For instance, a probabilistic state automaton (also known as a Markov chain) with an initial state defines a transduction (composed of two basic transductions: a generator and a unit delay) on a discrete time structure; a temporal stochastic integration with a given initial value is a typical transduction on a continuous time structure. Just as nullary functions represent constants, nullary transductions represent stochastic traces. Transductions are closed under functional composition.

2.5.2 Primitive transductions

Primitive transductions are defined on a generic time structure $T$. As mentioned above, *primitive transductions* are functional compositions of three types of *basic transduction*: generators, transiterations and delays.

First, let us introduce the notion of generators. A generator will be the most basic component of our framework to introduce uncertainty in the model. Generators will in fact represent random variables that will be incorporated in transiterations and ultimately primitive transductions.

Definition 2.41 (Generator) Let $A$ be a domain, $\Omega$ be a sample space and $T$ a time structure. Moreover, let $F_{X|A}$ denote the (potentially conditional) cumulative distribution function for the random variable $X$. A generator $G^A(v_0) : \Omega \times T \times A \rightarrow A$ is a basic transduction defined as

$$G^A(v_0, F_X)(v) = \begin{cases} v_0 & \text{if } t = 0 \\ \text{rand}(F_{X|A}v(\omega, t))(t, \omega) & \text{otherwise} \end{cases}$$

where rand$(F_{X|A}v(\omega, \cdot))$ is a random number generator associated with $F_{X|A}$.

We allow the distribution function $F_{X|A}$ to be conditioned on $t$ and values of the systems to produce a general model of uncertainty. This enables the user to model systems where the uncertainty component is non-stationary and conditioned on the state of the system. Also note that in this dissertation, we are not interested in the simulation of random variables per se, but rather in the analysis of the resulting models. Hence, we will assume that we are given, for each generator included in the model, appropriate random number generators. For more details on this widely studied field, we
refer the reader to the following books and articles chosen from the plethora of work done on random number generators [BFS87, Dev86, Gen98, HL00, Knu98, LK00, L'E94, L'E98].

**Definition 2.42 (Transliteration)** A transliteration is a pointwise extension of a function. Formally, let \( f : \Omega \times A \rightarrow A' \) be a function and \( T \) be a time structure. The pointwise extension of \( f \) onto \( T \) is a mapping \( f_T : A^{\Omega \times T} \rightarrow A'^{\Omega \times T} \) satisfying \( f_T(\omega, t) = \lambda \omega, t.f(\omega, t) \).

By this definition, \((f \circ g)_T = f_T \circ g_T\). We will also use \( f \) to denote transliteration \( f_T \) if no ambiguity arises.

Intuitively, a transliteration is a transformational process without memory or internal state, such as a combinational circuit. For example, consider the function \( f : \Omega \times \mathbb{R} \rightarrow \mathbb{R} \) defined as \( f(x) = 0.5x + Y(\omega) \), where \( Y(\cdot) \) is a stationary random variable uniformly distributed over the set \{1, 2\}. The pointwise extension \( f_T \) of \( f \) is a simple transliteration producing an output resulting in the sum of the input value \( x \) times 0.5 with a random value included in \{1, 2\}. We will come back to this seemingly simple transliteration as it will serve as basis for a running example throughout this section. We will also use this example to illustrate the semantics of the PCN framework.

Note that in the absence of any random variable within the transliteration, the transformational process is simply a deterministic function of the input.

An important class of transliteration is that of asynchronous event control [Sut89b]. For example, let \( \oplus : \overline{B} \times \overline{B} \rightarrow \overline{B} \) be a function defined as \( x \oplus y = (\neg x) \land y \lor x \land (\neg y) \), i.e., the "exclusive or". The transliteration \( \oplus \), as introduced in [Zha94], functions as the basic "or" logic in asynchronous event control (Figure 2.2). For a more detailed description of deterministic event logics, the reader is referred to §5 of [Zha94].

Now let us present the last type of basic transductions: delays. There are two types of delay: unit delays and transport delays.

For a given trace, a unit delay \( \delta^A_T(\omega, v_0) \) acts as a unit memory for data in domain \( A \), given a discrete time structure. We will use \( \delta(v_0) \) to denote unit delay \( \delta^A_T(\omega, v_0) \) if no ambiguity arises.

**Definition 2.43 (Unit delay)** Let \( A \) be a domain, \( v_0 \) a well-defined value in \( A \), and \( T \) a discrete time structure. A unit delay \( \delta^A_T(\omega, v_0) : A^{\Omega \times T} \rightarrow A^{\Omega \times T} \) is a transduction defined as

\[
\delta^A_T(\omega, v_0)(v) = \lambda t. \begin{cases} v_0 & \text{if } t = 0 \\ v(\omega, \text{pre}(t)) & \text{otherwise} \end{cases}
\]
Figure 2.2: Event logic for "or"

where \( v_0 \) is called the initial output value of the unit delay.

However, in the presence of non-discrete time structures, unit delays may not be meaningful. Hence we need a transduction that is suitable for more general time structures.

**Definition 2.44 (Transport delay)** Let \( A \) be a domain, \( v_0 \) a well-defined value in \( A \), \( T \) a time structure and \( \tau > 0 \). A transport delay \( \Delta^A_T(\tau)(\omega, v_0) : A^{\mathbb{N} \times T} \to A^{\mathbb{N} \times T} \) is a transduction defined as

\[
\Delta^A_T(\tau)(\omega, v_0)(v) = \begin{cases} 
  v_0 & \text{if } m(t) < \tau \\
  v(\omega, t - \tau) & \text{otherwise}
\end{cases}
\]

where \( v_0 \) is called the initial output value of the transport delay and \( \tau \) is called the time delay.

We will use \( \Delta(\tau)(v_0) \) to denote transport delay \( \Delta^A_T(\tau)(\omega, v_0) \) if no ambiguity arises. Transport delays are essential for modeling sequential behaviors in dynamical systems.

Let us now introduce an simple example of a primitive transduction. Let \( F \) denote the transduction represented in Figure 2.3 where \( f(\omega, x) = 0.5x + Y(\omega) \), and \( Y_\omega : \Omega \to \{1, 2\} \) is the generator following \( F_Y \), a distribution with uniform probability over the set \( \{1, 2\} \). Note that the unit delay \( \delta(0) \) is introduced to eliminate an algebraic loop, Hence, the output of this transduction would be a random sequence of values where the value at time \( t + 1 \) would be half of the value at time \( t \).
added to 1 or 2 with 50% probability. A possible execution trace resulting from this transduction on $T = N$ is \{0, 1, 2.5, 3.25, 2.625, ...\} with measure 0.0625.

2.5.3 Event-driven transductions

As defined above, a primitive transduction maps stochastic traces to stochastic traces with the same time structure. However, in general, a hybrid system consists of components of different time structures. Therefore, it is important to introduce a mechanism for handling such situations. In this section, we introduce event-driven transductions, which will allow us to model components with various time structures.

In order to properly introduce the notion of event-driven transductions, we need to define the concept of sample and extension traces. Let $T_r$ be a reference time of $T$ with a reference time mapping $h$. The sample stochastic trace of $v : \Omega \times T_r \to A$ onto $T$ is a stochastic trace $\tilde{v} : \Omega \times T \to A$ satisfying

$\tilde{v} = \lambda \omega, t. v(\omega, h(t))$.

The extension stochastic trace of $v : \Omega \times T \to A$ onto $T_r$ is a stochastic trace $\overline{v} : \Omega \times T_r \to A$ satisfying

$\overline{v} = \lambda \omega, t_r. \left\{ \begin{array}{ll} v(\omega, h^{-1}(t_r)) & \text{if } \exists t \in T, \mu_r([0_r, t_r]) \leq \mu([0, t]) \text{ or } \mu_r([0_r, t_r)) < \mu(T) \\ \bot_A & \text{otherwise} \end{array} \right.$
where $h^{-1}(t_r) = \{ t | h(t) \leq r \} \in T^\infty$.

Both sampling and extension can be seen as transformational processes on traces, hence they are transductions. **Sampling** is a type of transduction whose output is a sample trace of its input. **Extending** is a type of transduction whose output is an extension trace of its input.

**Proposition 2.21** Sampling and extending are continuous transductions

An event-driven transduction is a primitive transduction augmented with an extra input which is an event trace; it operates at each event point and the output value holds between two events. This additional event trace input of an event-driven transduction is called the **clock** of the transduction. Intuitively, an event-driven transduction works as follows. First, the input trace with the reference time $T$ is sampled onto the sample time $T_e$ generated by the event trace $e$. Then, the primitive transduction is performed on $T_e$. Finally, the output trace is extended from $T_e$ back to $T$.

**Definition 2.45 (Event-driven transduction)** Let $T$ be a time structure and $F_T : A^\Omega \times T \rightarrow A^\Omega \times T$ a primitive transduction. Let $E^\Omega \times T$ be the set of all stochastic event traces on time structure $T$. The event-driven transduction of $F$ is a mapping $F_T^e : E^\Omega \times T \times A^\Omega \times T \rightarrow A^\Omega \times T$ satisfying:

$$F_T^e(e, v) = \begin{cases} \lambda t. \perp_{A'} & \text{if } e = \lambda t. \perp_B \\
F_T(e(v)) & \text{otherwise.} \end{cases}$$

We will use $F^e$ to denote event-driven transduction $F_T^e$ if no ambiguity arises.

### 2.6 Dynamics Structures

With preliminaries established, we define an abstract structure of dynamics.

**Definition 2.46 (Σ-dynamics structure)** Let $Σ = \langle S, F \rangle$ be a signature. Given a Σ-domain structure $A$ and a time structure $T$, a Σ-dynamics structure $D(T, A)$ is pair $\langle \mathcal{V}, \mathcal{F} \rangle$ such that

- $\mathcal{V} = \{ A^\Omega \times T \}_{s \in S} \cup E^\Omega \times T$ where $A^\Omega \times T$ is a stochastic trace space of sort $s$ and $E^\Omega \times T$ is the stochastic event space;
- $\mathcal{F} = \mathcal{F}_T \cup F_T^e$ where $\mathcal{F}_T$ is the set of basic transductions, including the set of translators $\{ f_A^s \}_{f \in F}$, the set of unit delays $\{ \delta_T^A(v_s) \}_{s \in S, v_s \in A_s}$, the set of transport delays
\{A^t_s(V_S)\}_{s \in S, t > 0, v_s \in A_s}, and the set of generators \(\{A^t_s\}_{s \in S}\); \(\mathcal{F}_T^o\) is the set of event-driven transductions derived from the set of basic transductions, i.e., \(\{F^o|F \in \mathcal{F}_T\}\).

We will now complete this chapter by enumerating the various properties of dynamics structures.

The following propositions establish the fact that the partial order of a trace space and the partial order of an event space are cpos.

**Proposition 2.22** The partial order of a domain is a cpo.

**Proposition 2.23** The partial order of a stochastic trace space is a cpo.

**Proposition 2.24** The partial order of a stochastic event space is a cpo.

The following propositions characterize the continuity of basic transductions in partial order topologies.

**Proposition 2.25** A transliteration \(f_T : A^{\Omega \times T} \rightarrow A'^{\Omega \times T}\) on any sample space \(\Omega\) and time structure \(T\) is continuous if \(f : A \rightarrow A'\) is continuous.

**Proposition 2.26** A unit delay on any discrete time structure is continuous.

**Proposition 2.27** A transport delay is continuous.

The following proposition characterizes the continuity of event-driven transductions.

**Proposition 2.28** An event-driven transduction \(F^o\) is continuous if its primitive transduction \(F\) on any discrete time structure is continuous.

The following theorem concludes these properties.

**Theorem 2.1** (\(\Sigma\)-dynamics structure) Let \(A\) be a \(\Sigma\)-domain structure and \(T\) a time structure. The \(\Sigma\)-dynamics structure \(\mathcal{D}(T, A) = (\mathcal{V}, \mathcal{F})\) satisfies (1) \(\mathcal{V}\) is a multi-sorted set of cpos and (2) transliterations, transport delays and event-driven transductions in \(\mathcal{F}\) are continuous in the partial order topology. If, in addition, \(T\) is discrete, all transductions in \(\mathcal{F}\) are continuous in the partial order topology.
Transductions are functions. The well-definedness and strictness of a transduction is the well-definedness and strictness of the function, respectively. The following propositions characterize well-defined and/or strict transductions in dynamics structures.

**Proposition 2.29** A transliteration \( f_T \) is well-defined iff function \( f \) is well-defined; \( f_T \) is strict w.r.t. an argument iff \( f \) is strict w.r.t. the argument.

**Proposition 2.30** Any delay is not strict. A unit delay on any discrete time structure is well-defined. A transport delay is well-defined.

**Proposition 2.31** An event-driven transduction \( F^\circ \) is well-defined iff \( F \) on any discrete time structure is well-defined; \( F^\circ \) is strict w.r.t. its event input, and \( F^\circ \) is strict w.r.t. one of the other input arguments iff \( F \) is strict w.r.t. the argument.

Event traces are non-intermittent and right-continuous. We call a transduction non-intermittent iff its output is non-intermittent whenever its input is non-intermittent. We call a transduction right-continuous iff its output is right-continuous whenever its input is right-continuous. The following propositions characterize non-intermittent and/or right-continuous transductions in dynamics structures.

**Proposition 2.32** A transliteration \( f_T \) is right-continuous if \( f \) is continuous in the derived metric topology; \( f_T \) with \( f : \times_i \overline{A_i} \rightarrow \overline{A} \) is non-intermittent if \( f \) is strict, well-defined and continuous in the derived metric topology.

**Proposition 2.33** A delay is non-intermittent. A transport delay is right-continuous.

**Proposition 2.34** An event-driven transduction is right-continuous. An event-driven transduction \( F^\circ \) is non-intermittent if \( F \) is non-intermittent.

For example, the “event or” transduction \( \oplus \) (Figure 2.2) is well-defined and strict; it is also right-continuous and non-intermittent. “Event or” is a typical event synchronizer. In Chapter 5, Modeling in Constraint Nets, we will define other event synchronizers that are all non-intermittent and right-continuous.

We have presented a topological structure of dynamics by formalizing time, domains and traces in topological spaces and by characterizing primitive and event-driven transductions. With such
a topological structure, continuous/discrete time and domains can be represented uniformly, and hybrid dynamical systems can be studied in a unitary model.
Chapter 3

Probabilistic Constraint Nets

By definition, hybrid dynamical systems can have multiple sorts associated to diverse data types. Moreover, the various components of such systems can rely on different clocks, whether synchronized or event-based.

The PCN framework developed in this thesis is designed to model stochastic systems as situated agents, which in general cannot be adequately described by existing deterministic frameworks. Furthermore, it allows for a complete hybrid modeling approach, where one can model time and domains as either discrete, continuous or both as well as incorporate a model for the uncertainty in the system. The flexibility of our framework is a great asset as it allows a designer to model a complex system under the umbrella of a single modeling language.

As our framework extends the CN framework, we refer to the brief survey of the CN framework presented in § 1.3.1. For a thorough introduction, however, we direct the reader to the original work of [ZM95a].

3.1 Probabilistic Constraint Nets

The CN modeling tool was built on a topological view of time and domain structures along with notions of traces, events and transductions. As it became obvious from Chapter 2, we retained these concepts and extended them with a measure-theoretic approach and a notion of transduction encompassing random locations obtained via generators. These definitions represent the foundation for the notion of time and domain in the Probabilistic Constraint Net framework.
3.1.1 Syntax of Probabilistic Constraint Nets

Similarly to a constraint net, a probabilistic constraint net consists of a finite set of locations, a finite set of transductions and a finite set of connections. However, in order to be able to handle the uncertainty in the systems that we model, we add an essential component: the generator. A generator acts like a random number generator, following a given probability distribution and inducing a random location as its output. Thus, in practice, generators can be represented as discrete (e.g. Poisson, uniform) or continuous (Gaussian, exponential) probability distributions although we will use a general (and formal) measure theoretic definition.

**Definition 3.1 (Probabilistic Constraint Nets)** A probabilistic constraint net is a tuple $PCN = \langle Lc, Td, Cn \rangle$, where $Lc$ is a finite set of locations, each associated with a sort; $Td$ is a finite set of labels of transductions (either deterministic or probabilistic), each with an output port and a set of input ports, and each port is associated with a sort; $Cn$ is a set of connections between locations and ports of the same sort, with the restrictions that (1) no location is isolated, (2) there is at most one output port connected to each location, (3) each port of a transduction connects to a unique location.

Intuitively, each location is of fixed sort; a location's value typically changes over time. A location can be regarded as a wire, a channel, a variable, or a memory cell. A output location of a generator will be viewed as a random variable.

Each transduction is a causal mapping from inputs to output over time, operating according to a certain reference time or activated by external events. Note that probabilistic transductions are built of at least one basic transduction called a generator. Every generator is associated with a given probability distribution, either discrete or continuous, thus the sort of the output of a probabilistic transduction is the sort of its probability distribution.

Connections relate locations with ports of transductions. A clock is a special kind of location connected to the event ports of event-driven transductions.

A location $l$ is called an output location of a PCN iff $l$ connects to the output port of a transduction in $Td$; otherwise, since isolated locations are not allowed it is an input location. We will use the notation $I(PCN)$ and $O(PCN)$ to denote the set of input and output locations of a probabilistic constraint net $PCN$. A probabilistic constraint net is open if there exists at least one input location,
otherwise it is said to be closed.

Another feature of our framework is its graphical representation. A PCN can be represented by a bipartite graph where locations are depicted by circles, transductions by boxes, generators by double boxes and connections by arcs. To differentiate them from deterministic locations, we depict random locations with double circles. It should be noted that, as mentioned above, a generator induces a \textit{random} variable as its output location since the output port of a generator will follow a probability distribution. The transductions with this (random) location as input will then behave like a deterministic transduction with probabilistic inputs, thus generating probabilistic outputs which will follow a mixture probability distribution, the nature of which will be dependent on the transduction and on the distribution of the input locations. Therefore, even though we will talk about output locations of generators as being random locations, in the presence of feedback and at least one generator, all locations will be random locations. However, we chose to differentiate between locations and random locations as it provides a visual and intuitive way of assessing where uncertainty initially enters the system.

Most commonly used families of probability distributions are parameterized, i.e., one can fully specify a probability distribution by giving values to the parameters of the family. By allowing input locations to be connected to generators (which are basic transductions), we also provide a very general framework that allows the designer to include any probability distribution in a model, whether it is one with static parameters or a more complex conditional distribution depending on time or on the current state of the system. The ability of generators to be dependent on certain locations of the model also greatly simplifies the design task when modeling a complex system for which the various uncertain inputs are not fully known. Indeed, specifying the parameters of a probability distribution is often hard and counter-intuitive. Therefore, a designer could set the parameters of the distribution to some \textit{default} location value, and then, as the system evolves, learn the values of the parameters of the distribution, thus updating their values as a better estimate is being learned. For example, to model sensor noise with a PCN generator following a Gaussian probability distribution on the discrete time structure $T = \mathbb{N}$, one would simply need to connect the inputs of the generator to the locations holding the static values of the mean $\mu$ and the variance $\sigma^2$ to generate samples from the Gaussian distribution at every time point in $T$ (see Figure 3.1).
To exemplify the graphical syntax of PCN further, we present in Figure 3.2 an open probabilistic constraint net on a discrete time structure where $f$ is a transliteration, $\delta$ is a unit delay with initial value $s_0$ and $Pr(i_p)$ is a generator following the given discrete probability distribution $p$. The random location $i_p$ is, at each time step, a realization (sample) from the distribution $p$. This PCN is thus a representation of a probabilistic state automaton: $s(0) = s_0, s(n+1) = f(i(n), i_p(n), s(n))$. This state automaton can be showed to be equivalent to a Markov model with state space $I \times S$.

As an example of continuous time system modeling, Figure 3.3 displays a closed probabilistic constraint net representing the correct mathematical interpretation of a generic autonomous SDE (also known as an Itô process):

$$X_t = X_{t_0} + \int_{t_0}^{t} f(X_s)ds + \int_{t_0}^{t} g(X_s)dB_s$$

(3.1)

Note that the second integral $\int$ in this equation is the Itô Stochastic integral which differs from the usual Riemann integral.
The Modeling of Subsystems

Complex physical systems may be composed of a set of subsystems which by interacting together in a hierarchical fashion produce the behavior of the global system studied. Based on the definitions of subnets and modules for CN, we introduce the notions of subnets and modules within the PCN framework.

**Definition 3.2 (Subnet)** A probabilistic constraint net $PCN_1 = (Lc_1, Td_1, Cn_1)$ is a subnet of $PCN_2 = (Lc_2, Td_2, Cn_2)$, written $PCN_1 \subseteq PCN_2$ iff $Lc_1 \subseteq Lc_2$, $Td_1 \subseteq Td_2$, $Cn_1 \subseteq Cn_2$ and $I(PCN_1) \subseteq I(PCN_2)$.

**Definition 3.3 (Module)** A module is a triple $(PCN, I, O)$, where $PCN$ is a probabilistic constraint net, $I \subseteq I(PCN)$ and $O \subseteq O(PCN)$ are subsets of the input and output locations of $PCN$. We say that $I \cup O$ defines the interface of the module. When it is clear by the context, we will use the notation $PCN(I, O)$ to denote the module $(PCN, I, O)$.

Similarly to the general definition of a PCN, a module $PCN(I, O)$ is closed if $I = \emptyset$; it is otherwise open. Graphically, a module will be represented by a box with rounded corners. Moreover, locations in $I(PCN) - I$ and $O(PCN) - O$ are respectively called hidden inputs and hidden outputs and are used to model non-determinism in a system.

Three basic operations that can be applied to obtain a new module from existing ones were introduced in [Zha94]; namely union, coalescence and hiding. We extend these operators for the PCN framework.

The union operation is used to obtain a new module created by two modules side by side. Formally, let $PCN_1 = (Lc_1, Td_1, Cn_1)$ and $PCN_2 = (Lc_2, Td_2, Cn_2)$ be two probabilistic con-
constraint nets, with \( Lc_1 \cap Lc_2 = \emptyset \) and \( Td_1 \cap Td_2 = \emptyset \), then the union of \( PCN_1(I_1, O_1) \) and \( PCN_2(I_2, O_2) \), written \( PCN_1(I_1, O_1) \parallel PCN_2(I_2, O_2) \), is a new module \( PCN(I, O) \) where \( PCN = \langle Lc, Td, Cn \rangle \) is a probabilistic constraint net with \( Lc = Lc_1 \cup Lc_2 \), \( Td = Td_1 \cup Td_2 \) and \( Cn = Cn_1 \cup Cn_2 \), \( I \cup O \) defines its interface with \( I = I_1 \cup I_2 \) and \( O = O_1 \cup O_2 \).

The coalescence operator combines two locations in the interface of a module into one, with the restriction that at least one of these two locations is an input location. Formally, let \( PCN = \langle Lc, Td, Cn \rangle \) be a probabilistic constraint net, \( l \in I \) and \( l' \in I \cup O \) be of the same sort, the coalescence of \( PCN(I, O) \) for \( l \) and \( l' \), denoted \( PCN(I, O)/(l, l') \), is a new module \( PCN'(I', O') \) with \( PCN' = \langle Lc[l'/l], Td, Cn[l'/l] \rangle \), \( I' = I - \{l\} \) and \( O' = O \), where \( X[u/x] \) denotes that \( x \) in \( X \) is replaced by \( u \).

Finally, the hiding operation, as its name suggests, deletes a location from the interface by turning it into a hidden location. Formally, let \( PCN = \langle Lc, Td, Cn \rangle \) be a probabilistic constraint net and \( l \in I \cup O \), the hiding of \( PCN(I, O) \) for \( l \), denoted \( PCN(I, O)\backslash l \), is a new module \( PCN'(I', O') \) with \( PCN' = PCN, I' = I - \{l\} \) and \( O' = O - \{l\} \).

In addition to these three basic operations, three combined operations have been developed in [Zha94]: cascade connections, parallel connections and feedback connections. Intuitively, the cascade connection joins two modules in series, the parallel connection does it in parallel and the feedback connection connects an output of a module to an input of its own. Since the extension of these operations to the PCN framework is straightforward, we simply illustrate them in Figure 3.4. The reader is referred to the original work on constraint nets for more details on these operations [Zha94].

The introduction of modules in our framework provides many beneficial effects. First of all, modules allow for the hierarchical composition of simpler structures for complex systems. This facilitates the design task as one can use existing modules as building blocks for modeling more complex systems. For example, consider the probabilistic state automaton of Figure 3.2 along with a deterministic transliteration labeled Output. By selecting either \( \{i, s\} \) or \( \{i, s'\} \) as the interface for this probabilistic constraint net, we obtain a probabilistic state automaton module \( PSA \). By cascading the module \( PSA \) to the transliteration Output, we can construct the input/output probabilistic automaton \( IOPA \) showed in Figure 3.5 In addition to providing the ability to model systems hier-

\(^1\)Note that \( Td \) is the set of transduction labels, which can be different for the same transduction.
archically, modules provide an efficient and flexible way of constructing various different systems with the same set of basic components. For example, let us revisit the input/output probabilistic
automaton $IOPA$ of Figure 3.5. Formally, an input/output probabilistic automaton is defined as a tuple $(I, S, O, f_s, f_o)$ where $I, S$ and $O$ represent the finite set of input values, states and output values respectively. $f_s$ and $f_o$ are known as the state transition and output functions. Note that the state transition and output functions need not be deterministic and can be uncertain, following a given probability distribution. The output function relates a value to the state of the system. For example, it can be seen as a state observation arising from a sensor reading. Deterministic output functions are not very interesting as they only provide a relabeling of the current state of the system. On the other hand, probabilistic output functions can be used to reason about the current state of the system when the actual state is unknown.

For our example, if we select $\{i, s'\}$ as the interface of the probabilistic state automaton $PSA$ then $IOPA$ is a Hidden Markov model [BP66, BE67, BS68, BPSW70] with $f_s = f$ and $f_o = g \circ f$. If, on the other hand, we select $\{i, s\}$ as the interface of $PSA$, then we get that $IOPA$ has a deterministic output mapping $f_o = g$ and hence is a Markov model. Note also that for the former choice of interface, $IOPA$ is equivalent to a probabilistic Mealy machine [Mea55] while the latter choice of interface leads to a probabilistic Moore machine [Moo56] with a deterministic output function.

Finally, another reason to introduce modules in our framework is the ability to capture internal structure of a system via hidden outputs and to introduce nondeterminism with hidden inputs. With the addition of nondeterminism to our probabilistic modeling framework, we are now able to model completely general systems with any type of uncertainty, whether it be probabilistic, stochastic or nondeterministic.

### 3.1.2 Semantics of Probabilistic Constraint Nets

We have briefly introduced the syntax of the probabilistic constraint nets model, which has the useful properties of being graphical and modular. However, the syntax does not provide a meaning for the model. Indeed, there are multiple models with similar syntax to probabilistic constraint nets (Petri Nets [Pet81] and their generalization Coloured Petri Nets [Jen81] for example) that have completely different interpretations. Therefore, it is necessary to have a formal semantics of probabilistic constraint nets in order to correctly interpret models of complex physical systems.

The fixpoint theory of partial order has been used as a semantical model for programming lan-
Figure 3.6: Simple PCN for a probabilistic sum.

Luages and models [Hen88]: in this case, a program (or a model) defines a function \( f \) and its semantics are defined to be the least solution of \( x = f(x) \), or the least fixpoint of \( f \). A similar approach was developed to provide a fixpoint semantics for the Constraint Net model [ZM95a]. However, even though our framework is similar to that of Constraint Nets, the semantics of PCN differ significantly from that of CN. This is due to the fact that we have now introduced uncertainty in the set of equations induced by the PCN model. Hence, a probabilistic constraint net is a set of equations with locations serving as variables. Some of the variables (locations) in the equations, those that are outputs of generators, are in fact random variables, obeying some probability distribution, which in turn affect the value of the transductions for which they are inputs. Transductions play the role of functions and the connections between locations and transductions generates a set of equations. Obviously, the semantics of a PCN should be a solution to this set of equations containing random variables. Figure 3.6 demonstrates the effect of random locations on the transductions. Transduction Add is a very simple transliteration representing the sum of two (probabilistic) inputs \( X \) and \( Y \). It is easy to notice that the output value for this transliteration also follows a probability distribution. In this case, there are 4 possible values which each have different likelihood of occurrence. One should note that although the distribution of a random variable is helpful in reasoning about its behaviour, one can reason about statistics such as the expected value, that is, one can redefine the notion of behavior in terms of average behavior for the system. In our simple example, we can see that the average output value of the system is 9.25.

Since the equations in a PCN model do not converge to a fixpoint but rather to a stationary distribution, the fixpoint theory of partial order cannot be utilized directly to provide a denotational
semantics for PCN. In fact, in the presence of uncertainty in the system, the least solution of an
equation with random variables is a Markov stochastic process.

To further illustrate the difference between the semantics of a deterministic system (CN) and one
encompassing uncertainty (PCN), let us compare two dynamical systems with nominal component

\[
\dot{X}_t = -X_t (X_t - 1)(X_t - 2).
\]

The first one is deterministic and has two distinct stable attractors (equilibria),\(^2\) at 2 and at 0, as
shown in Figure 3.7(a). The behaviour of this system is fully determined by its initial value and it
reaches one of the two stable fixpoints based on this initial value.

The second system, which cannot be modeled with a constraint net, is stochastically affected by
a simple Brownian motion process. A sample path for this system, for an initial value of \(X_0 = -2\),
is shown in Figure 3.7(b). For this specific realization, the system is initially attracted toward the
closest equilibrium which is at \(X = 0\). The system then fluctuates around this attractor, reacting
under the influence of the Brownian motion component and, around time \(t = 12\), a large enough
noise disturbance pushes the system over the value of 1, causing the system to be attracted toward the
other equilibrium, at \(X = 2\). Another spike of noise flips the system back to the lower equilibrium at
\(t = 35\) and so on. This example shows the effect of uncertainty on the system and its behaviour. In
this case, there is no fixpoint for this realization nor for the full system. For a set of sample paths with

\(^2\)There are in fact three different equilibria, at 0, 1 and 2 respectively. However, the equilibrium at 1 is
unstable. Any shift in value will cause the system to move away from this unstable equilibrium and move
towards one of the other two stable equilibria.
non-zero measure, the system will keep moving back and forth between the two stable equilibria as it is affected by the noise introduced by the Brownian motion component of the equation. However, the system will reach a stationary distribution. That is, in the long run, the probability distribution of the system will remain unchanged, independent of time. The corresponding density function for this distribution is shown in Figure 3.8. One can clearly observe that the system is symmetrically distributed with higher weight around the two stable equilibria located at $X = 0$ and $X = 2$. One should note that if the effect of the Brownian noise is diminished, the peaks at $X = 0$ and $X = 2$ rise or fall (depending on the starting value) as the noise is less likely to cause a jump large enough to cause the other equilibrium to become the main attractor. Letting the effect of the noise converge to zero would lead to the deterministic case as presented in Figure 3.7a), that is, the stationary distribution would be degenerate everywhere except at the equilibrium corresponding to the initial value of the system. Hence a deterministic system is in fact a simple case of the more general stochastic system.

We define the semantics for the Probabilistic Constraint Net model to be the least fixpoint of the distribution of the solution to the set of equations of the PCN model. These semantics are, as it was mentioned in the previous paragraph, applicable to any system, whether it be stochastic or deterministic.
3.1.3 Fixpoint in distribution of partial orders

A fixpoint in the distribution of a function $f$ can be considered as a solution of the equation $x = f(x)$, where $f(\cdot)$ is an stochastic function. The least fixpoint is the least element in the fixpoint set.

**Definition 3.4 (Fixpoint in distribution and Least fixpoint)** Let $f : \Omega \times A \rightarrow A$ be a function on a sample space $\Omega$ and a partial order $A$. A function $g : \Omega \times A \rightarrow A$ is a fixpoint in distribution of $f$ iff the distribution of $g$ is a stationary distribution for $f$. It is the least fixpoint in distribution of $f$ iff, in addition, $F_g \leq F_{g'}$ for every other function $g'$ which is a fixpoint in distribution of $f$.

Least fixpoints in distribution, if they exist, are unique. The least fixpoint in distribution of $f$ will be denoted by $\mu.F_f$.

Based on the above definition, we can state our first fixpoint in distribution theorem as follows.

**Theorem 3.1 (Fixpoint Theorem I)** Let $A$ be a cpo and assume that either $A$ is also a total order or that the set of distributions over $A$ is a cpo and the function over distributions is continuous. Then, every continuous function $f : \Omega \times A \rightarrow A$ or pathwise continuous function $f^\omega : A \rightarrow A$ (for a fixed $\omega \in \Omega$) has a least fixpoint in distribution.

We now present our second fixpoint in distribution theorem which is applicable to a function of two arguments.

**Theorem 3.2 (Fixpoint Theorem II)** Let $A$ and $A'$ be two epocs and assume that either $A$, $A'$ are also total orders or that the set of distributions over $A'$ is a cpo and the function over distributions is continuous. If $f : \Omega \times A \times A' \rightarrow A'$ is a continuous function, then there exists a unique continuous function $\mu.f : \Omega \times A \rightarrow A'$, such that for all $a \in A$, the distribution of $(\mu.f)(a)$ is the least fixpoint in distribution of $\lambda\omega,x.f_\omega(a,x)$.

The distribution of the continuous function $\mu.f : \Omega \times A \rightarrow A'$ is called the least fixpoint in distribution of function $f : \Omega \times A \times A' \rightarrow A'$ or the least solution of the equation $y = f(x, y)$.

Continuous and pathwise continuous functions can also be extended.

**Proposition 3.1** Let $I \subseteq J$ be an index set. If $f : \Omega \times (\times_I A_i) \rightarrow A$ is a continuous or pathwise continuous function, then the extension of $f$, $f' : \Omega \times (\times_J A_J) \rightarrow A$ satisfying $f'(\omega, a) = f(\omega, a_{|I})$, is a continuous or pathwise continuous function.
Formally, a set of equations can also be written as \( \bar{o} = \bar{f}(\bar{x}, \bar{t}, \bar{o}) \) where \( \bar{t} \) is a tuple of input variables and \( \bar{o} \) is a tuple of output variables. Based on our previous results, if \( \bar{f} \) is continuous, then its least fixpoint in distribution is a continuous function, denoted \( \mu.\bar{f} \).

3.1.4 Semantics of Probabilistic Constraint Nets

In this section, we define the fixpoint in distribution semantics of probabilistic constraint nets. Let \( \Sigma = (S, F) \) be a signature and \( c \in S \) be a special sort for clocks. A probabilistic constraint net with signature \( \Sigma \) is a tuple \( PCN_S = (Lc, Td, Cn) \) where

- each location \( l \in Lc \) is associated with a sort \( s \in S \), the sort of location \( l \) is written as \( s_l \);
- each transduction \( F \in Td \) is a basic transduction or an event-driven transduction, the sorts of the input and output ports of \( F \) are as follows:
  1. if \( F \) is a transliteration of a function \( f : s^* \rightarrow s \in F \), the sort of the output port is \( s \) and the sort of the input port \( i \) is \( s^*(i) \);
  2. if \( F \) is a unit delay \( \delta^s \) or a transport delay \( \Delta^s \), the sort of both input and output ports is \( s \);
  3. if \( F \) is an event-driven transduction, the sort of the event input port is \( c \), the sorts of the other ports are the same as its primitive transduction;

Let \( D(T, A) = (V, F) \) be a \( \Sigma \)-dynamics structure. \( PCN_S \) on \( (V, F) \) denotes a set of equations \( \{o = F_o(\bar{x})\}_{o \in O(PCN)} \), such that for any output location \( o \in O(PCN) \),

- \( F_o \) is a continuous or pathwise continuous transduction in \( F \) whose output port connects to \( o \),
- \( \bar{x} \) is the tuple of input locations of \( F_o \), i.e., the input port \( i \) of \( F_o \) connects to location \( \bar{x}(i) \).

The semantics of a probabilistic constraint net is defined as follows.

**Definition 3.5 (Semantics)** The semantics of a probabilistic constraint net \( PCN \) on a dynamics structure \( (V, F) \), denoted \([PCN]\), is the least stationary distribution of the set of equations \( \{o = F_o(\bar{x})\}_{o \in O(PCN)} \), given that \( F_o \) is a continuous or pathwise continuous transduction in \( F \) for all \( o \in O(PCN) \); it is a continuous or pathwise continuous transduction from the input trace space to the output trace space, i.e., \([PCN] : \times_{I(PCN) A_{s_i}^{1 \times T}} \rightarrow \times_{O(PCN) A_{s_o}^{1 \times T}} \).
Figure 3.9: Sample path of the system in Figure 2.3 from page 57.

Given any set of output locations $O$, the restriction of $[PCN]$ onto $O$, denoted $[PCN]_O : \times_I(PCN)A^T_\omega \rightarrow _OA^T_\omega$, is called the semantics of $PCN$ for $O$. For example, the probabilistic constraint net in Figure 2.3 on page 57 denotes equations $x' = f(x, \omega) = 0.5x + y(\omega)$ and $x = \delta(0)(x)$ with $F_Y = Uniform\{1, 2\}$ and $\Omega = \{\omega_1, \omega_2\}$. Given a discrete time structure $N$, a domain $I = \{1, 2\}$ for inputs and a domain $O = \mathbb{R}$ for output, the semantics for $x$ is $F : I^{\Omega \times N} \rightarrow \mathbb{R}^{\Omega \times N}$ such that $F(v)(0) = 0$ and $F(v)(n) = f(F(v)(n - 1), v(n - 1))$ where the limiting distribution for $F$ is stationary.

Let us show the derivation of the semantics of this model (see Figure 2.3). In Figure 3.9, we plot a realization trace of the system, while in Figure 3.10 we can see the empirical distribution of the system after 10000 time steps. The least fixpoint distribution follows a uniform distribution over the range $[2, 4]$. The evolution of the distributions is presented in Figure 3.11. One can see that the system’s distribution starts as uniform over the range $\{1, 1\}$ and the distribution gradually increases to reach a stationary distribution which follows a uniform distribution over $[2, 4]$.

### 3.1.5 Semantics of Modules

In the previous section, we have formally defined the semantics of a probabilistic constraint net as a stationary distribution of a transduction. Now let us introduce the semantics of a PCN module. Formally, we define the semantics of a module as a set of transductions.
Definition 3.6 (Semantics of PCN modules) Given that the semantics of a probabilistic constraint net $PCN$ is $[PCN] : \times_{I(PCN)}A_{\Omega}^{\times T} \rightarrow \times_{O(PCN)}A_{\Omega_{o}}^{\times T}$, the semantics of a module $PCN(I, O)$ is $[PCN(I, O)] = \{F_u : \times_{I}A_{\Omega_{i}}^{\times T} \rightarrow \times_{O}A_{\Omega_{o}}^{\times T}\}_{u \in U}$ where $F_u(i) = [PCN]_{O}(u, i)$ and $U \subset \times_{I(PCN)}A_{\Omega_{i}}^{\times T}$ is the set of well-defined hidden input traces.

Proposition 3.2 Here are some properties that can be inferred from module operators. The semantics of a composite module can be derived from the semantics of the components with which it was constructed:

- Cascade connection: If $PCN(I, O) = PCN_2(I_2, O_2) \circ PCN_1(I_1, O_1)$, then

  $[PCN(I, O)] = \{F_2 \circ F_1 | F_1 \in [PCN_1(I_1, O_1)], F_2 \in [PCN_2(I_2, O_2)]\}.$

- Parallel connection: If $PCN(I, O) = PCN_1(I_1, O_1) + PCN_2(I_2, O_2)$, then

  $[PCN(I, O)] = \{(F_1, F_2) | F_1 \in [PCN_1(I_1, O_1)], F_2 \in [PCN_2(I_2, O_2)]\}$
where \( F_1, F_2 \) and \( F_1, F_2 \) are functions.

- Feedback connection: If \( PCN'(I', O') = F(PCN(I, O)) \), then
  \[
  [PCN'(I', O')] = \{ \mu.F | F \in [PCN(I, O)] \}
  \]
  where \( \mu.F \) is the the least fixpoint of \( F \).

- Union: If \( PCN(I, O) = PCN_1(I_1, O_1) \parallel PCN_2(I_2, O_2) \), then
  \[
  [PCN(I, O)] = [PCN_1(I_1, O_1)] \times [PCN_2(I_2, O_2)].
  \]

We say that a probabilistic constraint net \( PCN \) is well-defined iff its semantics, transduction \([PCN]\), is well-defined. For example, consider again the probabilistic constraint net in Figure 3.2. Given a discrete time structure (e.g., \( T = \mathbb{N} \)), a well-defined function \( f \) and a proper probability distribution \( P_t \), the PCN is well-defined. Similarly, we will say that a module is well-defined iff all the transductions in its semantics are well-defined. Moreover, if a probabilistic constraint net is well-defined, then, by definition, all its modules are well-defined.

One important properties of modules is that their well-definedness is closed under the following module operations.
Proposition 3.3 If $CN_1(I_1, O_1)$ and $CN_2(I_2, O_2)$ are well-defined modules, then the following resulting modules:

- **Union Connection**: $PCN_1(I_1, O_1) \parallel PCN_2(I_2, O_2)$,
- **Cascade Connection**: $PCN_1(I_1, O_1) \circ PCN_2(I_2, O_2)$,
- **Parallel Connection**: $PCN_1(I_1, O_1) + PCN_2(I_2, O_2)$,

are well-defined modules.

Note, however, that well-definedness is not closed under the feedback operation.

The following proposition denotes the relationship between the well-definedness of a probabilistic constraint net model and the strictness of the transductions in the model.

Proposition 3.4 Let $A$ and $A'$ be two cpos and assume that either $A$ and $A'$ are also total orders or that the set of distributions over $A'$ is a cpo and the function over distributions is continuous. If $f : \Omega \times A \times A' \rightarrow A'$ is a strict continuous function w.r.t. its third argument ($a' \in A'$), then the least fixpoint of $f$, or the least solution of the equation $\omega = f(\omega, i, o)$, is undefined.

To illustrate this property, consider the PCN model of Figure 2.3. For the purpose of this example, let $+, \cdot : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ represent the strict extensions of the addition and multiplication operators, $+$ and $\cdot$, respectively. Obviously, one can derive the equivalent transcriptions associated with these two operators: $+, \cdot : \mathbb{R}^T \times \mathbb{R}^T \rightarrow \mathbb{R}^T$. The least solution of $x = 0.5x + y$ on the dynamics structure $D(T, \mathbb{R})$ is undefined, even though $g$, with $F_g = Uniform([2, 4])$, is a well-defined random variable which is the least fixpoint in distribution.

In general, and as illustrated in the example above, the presence of an algebraic loop precludes probabilistic constraint nets from being well-defined. We now define formally the notion of algebraic loop with the PCN framework and discuss its implications.

Definition 3.7 (Algebraic loop) A location $l$ is **strictly dependent** on a location $l'$ in the probabilistic constraint net $PCN$, denoted by $l \leftarrow l'$, iff: (1) there is a transduction $F$ in $PCN$ such that $l$ is the output location of $F$, $l'$ is an input location of $F$, and $F$ is strict w.r.t. the input port (indicating an input argument) that connects with $l'$; or (2) $\exists l'' : l \leftarrow l''$, $l'' \leftarrow l'$. We say that $PCN$ has an algebraic loop on a location $l$ iff $l$ is dependent on itself; i.e., $l \leftarrow l$. 

78
Proposition 3.5 (Adapted from Proposition 4.2.10 of [Zha94]) A module PCN(I, O) is not well-defined if there is an output location \( l \in O \) such that PCN has an algebraic loop on \( l \).

It is very common that an output location is also an input location of the same transduction. In fact, this is exactly the effect of the feedback connection. Hence, we need a mechanism to break algebraic loop and hence obtain a well-defined net. A common strategy to break an algebraic loop is simply to insert a delay since all real components have non-zero delays. For example, by inserting a unit delay \( \delta(0) \) to the equation \( x = 0.5x + y \), we have \( x = 0.5x + y, x = \delta(0)(z) \). This is a well-defined probabilistic constraint net which has the well-defined semantics equivalent to a uniform distribution over the range \([2, 4]\).

3.1.6 Family of Probabilistic Constraint Nets

As mentioned in the previous section, systems are often parameterized, yielding a family of systems whose behaviours may differ significantly from each other due to their parameter value. In this section, we introduce the notion of parameterized probabilistic constraint nets and present the notion of limiting semantics of such PCN models.

Let us first introduce the notion of a static parameter. Formally, a static parameter is a variable in a transduction whose value does not change over time; hence a static parameter could be an input location of a transduction only if this location's value does not change over the life of the system. Typical examples of static parameters of robotic systems include physical constants such as gravitational force, mass, friction coefficient as well as elements specific to the system of interest like initial state, time delay and parameters of probability distributions such as mean \( (\mu) \) and variance \( (\sigma^2) \) in the case of a Gaussian distribution. Note that for a static parameter the value is kept constant throughout the life of the agent. Each value thus represents a different agent, taken from the family of agents generated by the space of the parameter.

We now formally define a parameterized probabilistic constraint net. Let \( PCN \) be a probabilistic constraint net and \( \mathcal{P} \) be a set of static parameters in \( PCN \). We use \( PCN^\mathcal{P} \) and \( PCN^\mathcal{P}(I, O) \) to denote a parameterized net and a parameterized module, respectively. Each static parameter \( p \in \mathcal{P} \) is associated with a set of values \( D_p \). The cross-product \( \times_p D_p \) is called the parameter space of the system. The semantics of a parameterized net \( PCN^\mathcal{P} \) is defined as follows.
Definition 3.8 (Semantics of parameterized PCNs) The semantics of a parameterized probabilistic constraint net $PCN^p$, denoted $[PCN^p]$, is a mapping from the parameter space to the set of transductions of the system, i.e., $[PCN^p] : \times_p D_p \rightarrow (\times_{I(PCN)}A_{s_i}^{\Omega \times T} \rightarrow \times_{O(PCN)}A_{s_o}^{\Omega \times T})$ such that for any static parameter tuple $v \in \times_p D_p$, $[PCN^p](v) = [PCN[v/P]]$ where $PCN[v/P]$ denotes that each $p \in P$ in $PCN$ is replaced by its corresponding value $v(p)$.

The semantics of a parameterized module $PCN^p(I,O)$, denoted $[PCN^p(I,O)]$, is now also a function of the parameters: $[PCN^p(I,O)](v) = [PCN(I,O)[v/P]]$.

The main reason for introducing the notion of parameterized nets is two-fold. First, this enables a system designer to model and analyze a system under variations of its static parameters. Parameters can have a significant impact on the behaviour of a system. For example, a certain subset of the parameter space might cause the system to be unstable while the rest of the parameter space renders the system stable. For example, consider a parameterized version of our example of Figure 3.2. Let $k$ be a gain parameter with $D_k = \mathbb{R}$, and $z = kx + y, x = \delta(0)(z)$, with $y$ be time independent, uncorrelated output from a uniform distribution over $\{1,2\}$, be a probabilistic constraint net on dynamics structure $D(N, \mathbb{R})$.

As seen before, the semantics of this system is a sequence of random variables $0, 2 + y_1, k y_1 + y_2, k^2 y_1 + ky_2 + y_3, \ldots$ where $y_i$ denotes the realization from a uniform random variable over $\{1,2\}$ at time instant $i$. An analysis similar to the deterministic case can be performed to reason about the effect of parameter $k$ on the system’s behaviour. For general values of $k$ we have $\lim_{n \to \infty} z(n) = \sum_{i=0}^{n} k^i y_{n-i}$. It is easy to show that this series converges for $|k| < 1$ and diverges for any other value, i.e., $|k| \geq 1$. Moreover, we have that $E(z_n) = E(\sum_{i=0}^{n} k^i y_{n-i}) = E(y) \sum_{i=0}^{n} k^i$ since the $y_i$’s are independent and identically distributed. In the limit, for $|k| < 1$, we get $\lim_{n \to \infty} E(z_n) = E(y) \lim_{n \to \infty} \sum_{i=0}^{n} k^i = \frac{1}{1-k}$, with $\lim_{n \to \infty} F_{z(n)} = Uniform([2, 4])$ as discussed earlier.

Second, by introducing parameterization of PCN models, we allow for the notion of limiting semantics to be defined. Let $\mathbb{P}$ be a set of parameters, $\times_p D_p$ be the parameter space, and $\leq_{\times_p D_p}$ be a partial order relation. If $(\times_p D_p, \leq_{\times_p D_p})$ is a linear order, and $PCN^p$ is a closed parameterized net whose semantics is a mapping $[PCN^p] : \times_p D_p \rightarrow \times_{Lc}A_{s_i}^{\Omega \times T}$, the limiting semantics of $PCN^p$ w.r.t. the parameter set $\mathbb{P}$, written $[PCN^*]$, is defined as the limit of the linear set of traces $[PCN^p]$, i.e., $[PCN^*] = \lim [PCN^p]$.

An important parameter used with limiting semantics is the infinitesimal, denoted by $\epsilon$. Let $\epsilon$
be a parameter over the real range $D_e = (0, 1) \subset \mathbb{R}$ associated with the partial order relation $\leq_{D_e}$ defined as $e_1 \leq_{D_e} e_2$ iff $e_2 \leq_{\mathbb{R}} e_1$. $(D_e, \leq_{D_e})$ is a linear order. Then, the limiting semantics of the parameterized net $PCN^e$ is $\lim_{\epsilon \to 0} [PCN^e]$. Limiting semantics will prove essential to reason about the semantics of stochastic temporal integration, which we introduce in the following section.

### 3.1.7 Stochastic temporal integration

One of the most important transductions on continuous time structures is temporal integration. Several issues are raised when considering continuous uncertainty present in a system. Brownian motion is the only stochastic process with continuous paths that has desirable properties for modeling uncertainty. But how does one integrate over Brownian motion processes? Can it be done in a similar way to common Riemann integration following, for example, the forward Euler method? Furthermore, a desired property of temporal integration within the PCN framework, either deterministic or stochastic, is to be defined on any time or domain structure. To ensure the full abstraction of temporal integration, we define temporal integration on vector spaces (whether numerical or symbolic) and provide the semantics of probabilistic constraint nets with temporal integration using limiting semantics as introduced above. In this section, we will focus mainly on the stochastic integral, namely the Itô type. There are two main types of stochastic integral, namely Itô and Stratonovich integrals. Both interpretations have their specific uses in mathematical modeling, depending on the nature of the system being modeled. In subsequent sections however, we will refer to the Itô version of the integral as a simple transformation converts Itô to Stratonovich. For a thorough treatment of numerical simulation of SDE, the reader is referred to [Hig01]. Note also that a deterministic temporal integral arises as a special case of the stochastic version. For a complete discussion of deterministic temporal integration, the reader should consult §4.2.5 of [Zha94].

First, let us briefly introduce building blocks of temporal integration in PCN, i.e., the notion of vector space. For a thorough discussion on vector spaces, the reader is referred to [War72] or any introductory vector algebra and topology textbook. Formally, a vector space is a set $X$ associated with the functions sum and product: $+: X \times X \to X$ and $\cdot : \mathbb{R} \times X \to X$ and with $0_X \in X$ satisfying the following conditions:

$$x + y = y + x, (x + y) + z = x + (y + z),$$
\[ a(x + y) = ax + ay, (\alpha + \beta)x = \alpha x + \beta x, \]

\[ \alpha(\beta x) = (\alpha \beta)x, x + 0x = x, 0x = 0x, 1x = x. \]

As it is conventionally the case, we denote the sum of all elements in \( \{x_i\}_{i \in I} \) with \( \Sigma_I x_i \). Furthermore, a topological vector space is defined as a vector space with a topology such that the two operators + and \( \cdot \) are continuous functions. We will assume that + and \( \cdot \) are strict extensions.

Let \( U \) be a vector space with functions \( + : U \times U \to U \) and \( \cdot : \mathbb{R} \times U \to U \) continuous in metric topology. Stochastic temporal integration \( \int (s_0, B_\omega) : U^T \to U^T \) with an initial state \( s_0 \in U \) and a Brownian motion trace \( B_\omega : \mathbb{R}^+ \times T \) can be defined as follows.

First, we define stochastic temporal integration on a discrete time structure. For a discrete time structure \( T \), we have that for all \( t > 0 \), \( pre(t) \) denotes the previous time point. Moreover, assume that we have a \( B_\omega \) that is a discretized Brownian motion trace over \( T \). Stochastic temporal integration is defined as follows:

\[
\int (s_0, B_\omega)(u) = \lambda t. \begin{cases} 
\frac{s_0}{\Sigma_0 t' \leq t (B_\omega(t') - B_\omega(pre(t')) \cdot u(pre(t')) & \text{if } t = 0 \\
0 & \text{otherwise}
\end{cases}
\]

We can also represent \( \int (s_0, B_\omega) \) as the least solution of the equation

\[
s = \delta(s_0)(s) + dB_\omega \cdot \delta(0)(u)
\]

with

\[
dB_\omega = \lambda t. \begin{cases} 
0 & \text{if } t = 0 \\
B_\omega(t') - B_\omega(pre(t')) & \text{otherwise}
\end{cases}
\]

This equation for stochastic integration can be represented by a probabilistic constraint net over discrete time structures. Let us now extend temporal integration to an arbitrary time structure.

For any arbitrary time structure \( T \), stochastic temporal integration is defined using an infinitesimal event trace. More specifically, let \( T_e \) be a discrete sample time of the arbitrary time structure \( T \). \( T_e \) is generated by an event trace \( e \) with \( e = \Delta(\epsilon)(0)(-\epsilon) \) for an infinitesimal parameter \( \epsilon \). We define \( int_{s_0, B_\omega}(u, s) = \delta(s_0)(s) + dB_\omega \cdot \delta(0)(u) \) for a Brownian motion trace \( B_\omega \) on \( T \). Stochastic temporal integration \( \int (s_0, B_\omega) \) can be computed by a PCN module \( PCN(u, s) \) where \( PCN \) represents the following two equations:

\[
s = \int_{s_0}^e (e, u, s, B_\omega), \quad e = \Delta(\epsilon)(0)(-\epsilon)
\]
where \( e > 0 \) denotes an infinitesimal.

This definition can be considered as derived by the Euler-Maruyama method [Mar55], the stochastic equivalent to the forward Euler method. Its convergence has been proved in [GS04]. It represents the most-studied, best-understood and simplest-to-implement numerical method. Despite its popularity, the Euler-Maruyama method suffers from lack of numerical stability, low convergence order, incorrect stationary laws and some problems with the geometrical invariance properties. However, in this dissertation we are interested in semantics, rather than numerical simulation of stochastic differential equations. The reader is referred to [KP99] for an in depth introduction to the numerical solution and simulation of stochastic differential equations.

As an example, let us investigate the limiting semantics of the classical stochastic integral

\[
\int_0^t B_\omega(s)dB(s)
\]

for a given Brownian motion \( B_\omega(s) \). For this example, we have \( U = \mathbb{R} \), \( T = \mathbb{R}^+ \).

This equation can be modeled by a closed probabilistic constraint net represented by three equations:

\[
s = \text{int}^o_{s_0}(e, u, s, B_\omega), \quad e = \Delta(e)(0)(-e), \quad u = B_\omega.
\]

The solution for \( e \) is:

\[
e = \lambda t. \begin{cases} 0 & \text{if } \lfloor \frac{t}{\varepsilon} \rfloor \text{ is even} \\ 1 & \text{otherwise.} \end{cases}
\]

It can be shown that the exact solution of this SDE is [Oks98, Mao97]

\[
\int_0^t B_\omega(s)dB(s) = \frac{1}{2}(B_\omega^2(t) - t)
\]

Now let us demonstrate that the limiting semantics of this stochastic integral leads to this exact solution.

The solution for \( s \) is the least solution of \( s = \text{int}^o_{s_0}(e, B_\omega, s, B_\omega) \). Let \( s_0 = \lambda t. \perp \mathbb{R} \) be the least element. Then following the proof of the Fixpoint Theorem we get

\[
s^1 = \text{int}^o_{s_0}(e, B_\omega, s_0, B_\omega) = \lambda t. \begin{cases} s_0 & \text{if } t < \varepsilon \\ \perp \mathbb{R} & \text{otherwise,} \end{cases}
\]

\[
s^2 = \text{int}^o_{s_0}(e, B_\omega, s^1, B_\omega) = \lambda t. \begin{cases} s_0 & \text{if } t < \varepsilon \\ s_0 + B_\omega(0)(B_\omega(\varepsilon) - B_\omega(0)) & \text{if } \varepsilon \leq t < 2\varepsilon \\ \perp \mathbb{R} & \text{otherwise,} \end{cases}
\]
\[ s^3 = \text{int}_{s_0}^0 (e, B_\omega, s^1, B_\omega) = \lambda t. \left\{ \begin{array}{ll}
 s_0 & \text{if } t < \epsilon \\
 s_0 + B_\omega(0)(B_\omega(\epsilon) - B_\omega(0)) & \text{if } \epsilon \leq t < 2\epsilon \\
 s_0 + B_\omega(0)(B_\omega(\epsilon) - B_\omega(0)) + B_\omega(\epsilon)(B_\omega(2\epsilon) - B_\omega(\epsilon)) & \text{if } 2\epsilon \leq t < 3\epsilon \\
 \bot & \text{otherwise},
\end{array} \right. \]

\[ s^{k+1} = \text{int}_{s_0}^0 (e, B_\omega, s^k, B_\omega) = \lambda t. \left\{ \begin{array}{ll}
 s_0 & \text{if } t < \epsilon \\
 s_0 + B_\omega(0)(B_\omega(\epsilon) - B_\omega(0)) & \text{if } \epsilon \leq t < 2\epsilon \\
 \frac{1}{2} \sum_{i=0}^{k} [B_\omega(ke)^2 - B_\omega((k-1)e)^2 - (B_\omega(ke) - B_\omega((k-1)e))^2] & \text{if } ke \leq t < (k+1)e \\
 \bot & \text{otherwise}.
\end{array} \right. \]

Let \( s = \sqrt{\mathbb{R}^+ \{ s^k \} \} \). Then \( s = \lambda t. s^{(\frac{1}{2})+1}(t) \) is the least solution of the equation \( s = \text{int}_{s_0}^0 (e, B_\omega, s, B_\omega) \).

With a simple manipulation we get that
\[ s^{k+1} = \frac{1}{2} \left( B_\omega(t)^2 - B_\omega(0) - \sum_{j=0}^{k} (B_\omega((k+1)e) - B_\omega(ke))^2 \right). \tag{3.3} \]

The term \( \sum_{j=0}^{k} (B_\omega((k+1)e) - B_\omega(ke))^2 \) in Equation 3.3 can be shown to have expected value \( t \) and variance of \( O(et) \). Hence, for small \( et \) we expect this random variable to be close to the constant \( t \). Therefore, the limiting semantics of the net for \( s \) is \( s^* = \lambda t. \lim_{t \to 0} s(t) = \frac{1}{2} (B_\omega^2(t) - t) \), which corresponds to the exact solution of \( s = \int_0^t B_\omega dB_\omega \).

Note that limiting semantics only applies to a closed parameterized net and is not composite. For a probabilistic constraint net with multiple temporal integrators, we will use a single infinitesimal for all the integrator transductions.

Furthermore, note that an Itô equation might not have a unique solution on the whole interval \([0, t]\). For example, Girsanov [Gir62] has shown that the one-dimensional Itô equation
\[ x(t) = \int_{t_0}^t |x(s)|^\alpha dB(s) \]
has infinitely many solutions when $0 < \alpha < 1/2$. In the next chapter, we will address the issue of existence and uniqueness of the solution of a stochastic differential equation and discuss the conditions under which a probabilistic constraint net produces the “correct” solution.

Analogously to the approach in the CN framework, we can define three variations of stochastic temporal integration: (1) stochastic temporal integration with bounds, (2) stochastic temporal integration with reset, and (3) stochastic integration against another trace on domain $\mathbb{R}$. As the extension of these variations of temporal integration is straightforward and their names are self-explanatory, we refer the interested reader to [Zha94] for a detailed description of these alternate temporal integrals which can prove very useful when modeling complex dynamical systems.
Chapter 4

Modeling in PCN

As defined in the previous chapter, a probabilistic dynamical system is built on a dynamics structure $D(T, A)$ where $T$ and $A$ denote a time structure and a domain structure, respectively. The time and domain structures are abstracted so that they can be either continuous or discrete or hybrid. In Table 4.1, we present examples of the most commonly used models for probabilistic/stochastic dynamical systems in each of the four basic situations that can occur with discrete/continuous time and domain. In this chapter, we will discuss the issues related to modeling various types of systems with the PCN framework. In the next chapter, we will demonstrate how each of the models enumerated in Table 4.1 is subsumed by the PCN framework and show how to translate them into an equivalent PCN.

4.1 Events

We are interested in modeling the larger class of probabilistic dynamical systems encompassing components of more than one basic type. These systems are referred to as hybrid systems. We have

<table>
<thead>
<tr>
<th>Time</th>
<th>Domain</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Discrete</td>
<td>Continuous</td>
</tr>
<tr>
<td>Discrete</td>
<td>Markov chains</td>
<td>Stochastic Difference Equations</td>
</tr>
<tr>
<td></td>
<td>Dynamics Bayesian Networks</td>
<td></td>
</tr>
<tr>
<td>Continuous</td>
<td>Continuous Time Markov Chains</td>
<td>Stochastic Processes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stochastic Differential Equations</td>
</tr>
</tbody>
</table>

Table 4.1: Generic Types of probabilistic/stochastic models
developed Probabilistic Constraint Nets (PCN) as a formal model for probabilistic hybrid dynamical systems. Within the PCN paradigm, a probabilistic hybrid dynamical system consists of modules with different time structures, with its domain structure multi-sorted and with a set of probabilistic generators, as basic transductions, which allows for the modeling of the uncertain components of these modules.

As mentioned in the previous chapter, event-driven modules constitute an important part of our framework as they allow us to model systems with modules that are associated with different clocks. Hence, we can unify, within the same model, modules with different sample time structures generated by event traces. There are two ways in which an event trace can be generated: either with a fixed sampling rate, or by an event generator that reacts to changes in its inputs. Moreover, we can also combine multiple event traces, yielding new event traces.

Typically, event traces are combined using event logic which allow various asynchronous components within a given set of modules to be coordinated. Common logical interactions are "event or", "event and", and "event select". With event logic modules, asynchronous components can be coordinated. As event control logic, and event generators and synchronizers in PCN are analogous to those introduced in CN, we will simply summarize the most important concepts here. For more details, the reader is referred to the original work on the Constraint Net framework [Zha94].

4.1.1 Event generators

Events generators are one way in which events traces can be produced in the PCN model. Formally, an event generator is a transduction whose output is an event trace. Consider for example the transport delay $e = \Delta(t_s)(0)(-e)$. Recall that "$(0)$" indicated the initial output value of the delay is 0. This transduction is an event generator whose output is an event trace with a fixed sampling rate.

In Figure 4.1, we show two examples of basic event generator modules which were implemented as follows:

- Module NotEqual$(i,o)$ (Figure 4.1(a)) is composed of a unit delay and a transliteration notEqual where notEqual : $\mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$ is defined as $notEqual(x, y) = cond(x, y, 0, 1)$.

- Module $F(i,o)$ (Figure 4.1(b)) is composed of a unit delay and a transliteration $f$ where $f : \mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$ is defined as $f(x, y) = cond(x, 0, y, -y)$. 87
where \( \text{cond} \) is the conditional function defined in Equation 2.4. Although transliterations \( \text{notEqual} \) and \( f \) are applied to a different second input \( y \), one can easily deduce that both these transliterations act as an "exclusive or", \( \oplus \).

An important property of event generators is that any cascade connection to an event generator is also an event generator. For example, an event generator that generates an event whenever its input changes from 0 to 1, usually referred to as a rising transition, is a cascade connection of \( \text{NotEqual} \) to \( F \), i.e., \( F \circ \text{NotEqual} \).

Obviously, event generators can also include uncertainty by simply encompassing generators in the module. We now introduce an event generator that will prove very useful for modeling stochastic systems with a continuous time structure.

Let us consider an event generator process in which events occur randomly in time. The phrase events occur randomly in time is generic and could represent, for example:

- The times when a piece of radioactive material emits particles
- The times when customers arrive at a service station
- The times when requests arrive at a server computer
- The times when accidents occur at a particular intersection

Consider the following counting process: \( \{X(t) \mid t \in \mathbb{R}^+ \cup \{0\}\} \). Such a process is called a Poisson process (named after Simeon Poisson) [Poi37] with parameter \( \lambda \) if

Figure 4.1: Two basic modules for event logics: \( \text{NotEqual} \) and \( F \).
1. The probability that at least one Poisson arrival occurs in a time period of duration \( \tau \) is

\[
P(\tau) = \lambda \tau + o(\tau),
\]

where \( o(\tau) \) denotes a term that goes to zero faster than \( k\tau \) as \( \tau \) goes to zero (for any constant \( k \)). Mathematically, \( \lim_{\tau \to 0} \frac{o(\tau)}{\tau} = 0 \).

2. Let \( N(t) \) be the total number of Poisson arrivals occurring in the interval \([0, t]\). We assume that \( N(0) = 0 \). For the interval \( \{t_1, t_2\} \), the number of Poisson-type arrivals \( [N(t_2) - N(t_1)] \) for \( t_2 > t_1 \geq 0 \) is dependent only on \( (t_2 - t_1) \) and not on \( t_1 \) or \( N(t_1) \).

3. If \( 0 \leq t_1 < t_2 < t_3 < t_4 \leq \ldots \), the numbers of arrivals occurring in disjoint time intervals \( [N(t_2) - N(t_1)], [N(t_4) - N(t_3)], \ldots \) are mutually independent random variables.

4. The probability that two or more Poisson arrivals occur in a time interval of length \( \tau \) is \( o(\tau) \).

The basic underlying assumption of a Poisson process is that the behavior of the process after an arrival should be independent of the behavior before the arrival and probabilistically be like the original process. This property is often referred to as regeneration [Fel68]. In particular, the general regeneration assumption means that the times between arrivals, known as inter-arrival times, must be independent, identically distributed random variables. Furthermore, regeneration should occur at a fixed time \( t \). In particular, if the first arrival has not occurred by time \( t \), then the time remaining until the arrival occurs has the same distribution as the first arrival time itself. This is known as the memoryless property and can be stated in terms of a generic inter-arrival time \( X \) as follows:

\[
P(X > t + s | X > s) = P(X > t), \forall s, t \geq 0.
\]

One can also easily show that the only probability distribution which possesses these properties is the exponential distribution.

Based on these results, one can easily show that the occurrence time of events follows the iteration:

\[
t_{n+1} = t_n - \frac{\log(U(0, 1))}{\lambda}
\]

where \( U(0, 1) \) is a random variable following a Uniform distribution with parameter \((0, 1)\) and \( \lambda \) is the rate of the exponential distribution. Hence, an event generator following a Poisson process will yield events at uncertain times \( t_i, i = 0, \ldots \), following Equation 4.1.
4.1.2 Event synchronizers

A possible way of handling events is to modify existing event traces. For this purpose, *event synchronizers* were developed. Event synchronizers are transductions that map event traces to new event traces. For example, *event or* as presented in Figure 2.2 is an event synchronizer that generates an event if and only if no two events happen at the same time. Other event synchronizers that can be modeled in PCN include *event and*, *event filter*, *event select* and any event logic elements described in [Sut89a], such as *Switch, Event-Controlled Storage Element (ECSE), Toggle, Arbiter*.

Let us now briefly review the types of uncertainty arising in physical systems, and thus the types that an efficient framework should be able to model. We will then proceed to a discussion of the types of computation that are possible within the PCN framework.

4.2 Types of Uncertainties

When choosing a model for an uncertain system, it is important to capture the essential features of the real system and the uncertainties in that system so that the trajectories of the model mimic the behavior of the real system. There are many different types of models for uncertainty and the model to be used depends not only on the type of uncertainty expected within the system of interest but also depends upon what kind of analysis one wishes to perform on the model. Indeed, a particular uncertainty model might reflect the physical system's uncertainty very closely but might also be too complex to analyze, thus rendering the model useless. For this reason, it is often necessary to enlarge the class of uncertainties to ensure that we get a tractable model. For example, in the problem of optimal control, one might want to ensure that the model of the uncertainty is tractable so that methods for control synthesis can be applied. Unfortunately, this can lead to conservative control system designs.

In this thesis, we will focus mainly on modeling uncertainty via stochastic processes such as Markov chain for discrete time systems and Wiener processes for continuous systems. Note that stochastic modeling of uncertainty has been widely studied in the recent years and many alternative models have been suggested. For example, the notion of stochastic uncertain systems which relies on *integral quadratic constraints* has been developed to handle various types of uncertainty and is especially suited for analysis related to stochastic stability and robust control of systems [PUS00,
4.3 Computation in Probabilistic Constraint Nets

No computational model is suitable for every type of computation. A given model is developed to handle a certain type of computation and (hopefully) provides advantages over other models for this type of computation. For example, analog circuits are used to represent parallel and continuous computations while Turing machines and Markov chains are used for sequential computations, the latter when in the presence of uncertainty.

The PCN framework is no different in that it is inherently designed for parallel computation of uncertain dynamical systems with its main advantage over existing models being its abstraction of time and domain. However, PCN also allows for sequential computation to be modeled. In this section, we will introduce the reader to the notion of sequential computation by means of the PCN framework and will conclude with an introduction to continuous computation in PCN.

4.3.1 Sequential and Analog computation

It is possible, in PCN, to represent sequential computation using events to coordinate the order of computation. In fact, just as for the CN framework, we model sequential computation as a module with an event input indicating the start of a computation and an event output indicating the end of the computation (see Figure 4.2 taken from [Zha94]). The time duration between the start and the end of the computation is variable, depending on the input data. We call such a module a sequential module.\footnote{A basic transliteration $f$, described as a pointwise extension of a function in previous chapters, can be seen as a sequential module with $End = Start$ and $Data\_Out = f(Data\_In)$, i.e., computations happen instantly within a transliteration.}

Figure 4.2: A sequential module

Ugr98].
An important result is that given a set of basic functions and their sequential modules, the set of functions closed under functional composition, recursive schemes and minimization operations can be computed by sequential modules. In fact, this set is large enough to include all the computable functions given a small set of basic functions [Zha94]. However, many functions that are not easy to model in sequential computation are easy to compute as traces which, conveniently, are at the core of the PCN framework.

**Example 4.1** As an example of modeling uncertain continuous time systems with a Brownian motion, consider the following function on continuous time structures: $\lambda \omega, t.Ce^{(\lambda - \frac{1}{2} \mu^2)t} + \mu B_{\omega}(t)$. This function (or stochastic process) is the solution of a probabilistic constraint net $x = \int (C)(\lambda x) + \int (C, B_{\omega})(\mu x)$ representing the geometric Brownian motion [Mao97]

$$dX_t = \lambda X_t dt + \mu X_t dW_t.$$  \hspace{1cm} (4.2)

This SDE is of great importance in the field of economics and finance as the well-known Black-Scholes partial differential equation can be derived from it.

Let us now raise the following important question: Given a set of stochastic differential equations modeled in probabilistic constraint nets over a continuous time structure, what is the relationship between the semantics of the probabilistic constraint nets and the solutions of the stochastic differential equations? This question is equivalent to asking if the PCN has a well-defined semantics, and if the set of stochastic differential equations has a unique solution. With a positive answer to both of these questions, we have that a trace, obtained as a solution of a set of stochastic differential equations, can be computed as the limiting semantics of the PCN representing the set of SDEs.

### 4.3.2 Stochastic Taylor expansion

Let us now present an introduction to stochastic Taylor expansions. Stochastic Taylor expansions will be at the core of the limiting semantics for PCN on continuous time structure. Similarly to the case of ordinary differential equations and Taylor expansions, stochastic Taylor expansions are at the core of the numerical integration schemes for stochastic differential equations. First, let us define

$$\nabla_x = \left( \frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_d} \right)^T$$
as the d-dimensional gradient in the x-direction and $C^k(U, V)$ as the family of continuous functions from $U$ to $V$ with continuous derivatives up to order $k$.

We also need to introduce the famous Itô formula [Itô51]

$$\mathcal{L}^0 = \frac{\partial}{\partial t} + < f(t, x), \nabla_x >_d + \frac{1}{2} \sum_{j=1}^{m} \sum_{i,k=1}^{d} g_i^j(t,x)g_k^j(t,x) \frac{\partial^2}{\partial x_k \partial x_i}$$

and $\mathcal{L}^j = < g^j(t,x), \nabla_x >_d$ where $j = 1, 2, \ldots, m$.

Moreover, before formally introducing the notion of stochastic Taylor expansion to define the limiting semantics of a probabilistic constraint net of stochastic differential equations, we have to formulate what is meant by multiple indices, hierarchical sets, remainder sets, coefficient functions and multiple integrals in the Itô sense. This will allow us to present the results in a clearer and more compact fashion.

**Definition 4.1** A multiple index has the form $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_{l(\alpha)})$ where $l(\alpha) \in \mathbb{N}$ is called the length of the multiple index $\alpha$, and $n(\alpha)$ is the total number of zero entries of $\alpha$. The symbol $\nu$ denotes the empty multiple index with $l(\nu) = 0$. The operations $\alpha^- = (\alpha_1, \ldots, \alpha_{l(\alpha)-1})$ and $-\alpha = (\alpha_2, \ldots, \alpha_{l(\alpha)})$ are called right- and left-subtraction, respectively (in particular, $(\alpha_1)^- = -(\alpha_1) = \nu$). The set of all multiple indices is defined to be:

$$\mathcal{M}_{k,m} = \{ \alpha = (\alpha_1, \alpha_2, \ldots, \alpha_{l(\alpha)}) : \alpha_i \in \{k, k+1, \ldots, m\}, i = 1, 2, \ldots, l(\alpha), \text{ with } l(\alpha) \in \mathbb{N} \}.$$

A hierarchical set $Q \subset \mathcal{M}_{0,m}$ is any multiple indices $\alpha \in \mathcal{M}_{0,m}$ such that $\nu \in Q$ and $\alpha \in Q$ implies $-\alpha \in Q$. The hierarchical set $Q_k$ denotes the set of all multiple indices $\alpha \in \mathcal{M}_{0,m}$ with length smaller than $k \in \mathbb{N}$, i.e., $Q_k = \{ \alpha \in \mathcal{M}_{0,m} : l(\alpha) \leq k \}$.

The set $R(Q) = \{ \alpha \in \mathcal{M}_{0,m} Q : \alpha^- \in Q \}$ is called the remainder set $R(Q)$ of the hierarchical set $Q$. A multiple Itô integral $I_{a,s,t}[V(\cdot, \cdot)]$ is defined to be

$$I_{a,s,t}[V(\cdot, \cdot)] = \begin{cases} \int_s^t I_{-\alpha,s,u}[V(\cdot, \cdot)]dW_u^{a(\alpha)} & \text{if } l(\alpha) > 1 \\ \int_s^t V(u, X_u)dW_u^{a(\alpha)} & \text{otherwise} \end{cases}$$ (4.3)

for a fixed $\alpha \in \mathcal{M}_{0,m} \{\nu\}$ and for a given process $V(t, X_t)$ where $V \in C^{0,0}([0,T] \times \mathbb{R}^d, \mathbb{R}^k)$. A multiple Itô coefficient $V_\alpha \in C^{0,0}([0,T] \times \mathbb{R}^d, \mathbb{R}^k)$ for a given mapping $V = V(t, x) \in$
$C^{l(\alpha),2l(\alpha)}([0,T] \times \mathbb{R}^d, \mathbb{R}^k)$ is defined to be

$$V_{\alpha}(t,x) = \begin{cases} L^{l(\alpha)}V_{\alpha-}(t,x) & \text{if } l(\alpha) > 0 \\ V(t,x) & \text{otherwise} \end{cases} \quad (4.4)$$

Now that the notational background has been introduced, we are able to state a general form of the Itô-Taylor expansion. Stochastic Taylor expansions for Itô processes have been introduced and studied originally by Wagner and Platen [WP78].

An Itô-Taylor expansion\(^2\) for the standard Itô\(^3\) SDE

$$dX_t = f(t, X_t)dt + \sum_{j=1}^{m} g^j(t, X_t)dW^j_t,$$

where $f, g^j : [0, T] \times \mathbb{R}^d \rightarrow \mathbb{R}$ are the drift and the diffusion parts, and where $\{W^j_t : 0 \leq t \leq T\}$ represent $m$ mutually independent Wiener processes on the complete probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t \in [0,T]}, P)$, is of the form:

$$V(t, X_t) = \sum_{\alpha \in Q} I_{\alpha,s,t}[V_{\alpha}(s, X_s)] + \sum_{\alpha \in R(Q)} I_{\alpha,s,t}[V_{\alpha}(\cdot, \cdot)] \quad (4.5)$$

for a given mapping $V = V(t, x) : [0, T] \times \mathbb{R}^d \rightarrow \mathbb{R}^k$ which is smooth enough. For completeness, we restate Theorem 5.1 of [KP99].

**Theorem 4.1 Wagner-Platen Expansion**

Let $\rho$ and $\tau$ be two $\mathcal{F}_t$-adapted stopping times with $t_0 \leq \rho \leq \tau \leq T < \infty$ (a.s.). Assume $V : [0, T] \times \mathbb{R}^d \rightarrow \mathbb{R}^k$. Take any hierarchical set $Q \in \mathcal{M}_{0,m}$. Then, each Itô SDE with coefficients $f, g^j$ possesses a Itô-Taylor expansion of Equation 4.5 with respect to the hierarchical set $Q$, provided that all derivatives of $V, f, g^j$ (related to $Q$) exist.

Based on these notions we can present the following result.

**Proposition 4.1** Given a probabilistic constraint net made of the stochastic differential equations

$$\dot{x}_k = f_k(\bar{x}) + g_k(\bar{x})N_k, k = 1, \ldots, n \text{ with } x_k(t_0) \in \mathbb{R} \text{ and } f_k, g_k : \mathbb{R}^n \rightarrow \mathbb{R} \text{ as partial or total functions, and given that all } f_k \text{ and } g_k \text{ are sufficiently smooth at } \bar{x}(t_0),$$

the limiting semantics of...
the probabilistic constraint net, based on the Euler-Maruyama method, is well-defined over $T = [t_0, t_1]$ for some $t_1 > t_0$. In particular, the results of Theorem 4.1 apply for the smooth function $V(t, x) = x$. The reader should note that if $f_k$ and $g_k$ are polynomial functions, then both $f_k$ and $g_k$ are smooth over $\mathbb{R}^n$ and hence this results holds.

**Example 4.2** Recall the Geometric Brownian motion denoted by Equation 4.2, and introduced briefly in Example 4.1. Let us calculate the stochastic Taylor expansion of this process. Applying the Itô-Taylor expansion of Equation 4.5 to the SDE 4.2 we get

$$X_t = X_{t_0} \left(1 + \sum_{\alpha \in \mathcal{M}_{0,1}\setminus\nu} \lambda^{n(\alpha)} \mu^{l(\alpha)-n(\alpha)} I_{\alpha,t_0,t}\right)$$

$$= X_{t_0} \left(1 + \lambda(t - t_0) + \mu(W_t - W_{t_0}) + \sum_{\alpha \in \mathcal{M}_{1}\setminus\nu(0),(1)} \lambda^{n(\alpha)} \mu^{l(\alpha)-n(\alpha)} I_{\alpha,t_0,t}\right)$$

$$= X_{t_0} \sum_{i,j=0}^{+\infty} \frac{[\lambda-e^2](t-t_0)^i}{i!} [\mu(W_t-W_{t_0})]^j$$

$$= X_{t_0} \exp \left(\lambda - \frac{e^2}{2}(t-t_0) + \mu(W_t - W_{t_0})\right)$$

where the coefficient functions are $V(t, x) = x$, $V_{\alpha}(t, x) = \lambda^{n(\alpha)} \mu^{l(\alpha)-n(\alpha)} x$ with $n(\alpha)$ as the total number of zeros of $\alpha \in \mathcal{M}_{0,1}$, $\nu$ as the empty index, and where $I_{\alpha,s,t}$ without the argument $[\cdot]$ is understood to be $I_{\alpha,s,t}[1]$. \(\square\)

So far, we have provided a meaning for the rightmost integral of Equation 2.2, but did not mention whether we can obtain existence and uniqueness theorems for such equations. Moreover, one might be interested in knowing what kind of properties these solutions have, if any. We reproduce here the well known existence and uniqueness theorem for stochastic differential equations (SDEs) from §5.2 of [Oks98]. For simplicity we present the one-dimensional case but the results can be generalized to more complex SDEs. We will omit the proof here and refer the interested reader to the original work.

**Theorem 4.2** Existence and uniqueness theorem for SDEs

Let $T > 0$ and $b(\cdot, \cdot) : [0, T] \times \mathbb{R}^n \to \mathbb{R}^n$, $g(\cdot, \cdot) : [0, T] \times \mathbb{R}^n \to \mathbb{R}^{n \times m}$ be measurable functions satisfying

$$|b(t, x)| + |g(t, x)| \leq C(1 + |x|); \quad x \in \mathbb{R}^n, t \in [0, T]$$

(4.7)

for some constant $C$, and such that

$$|b(t, x) - b(t, y)| + |g(t, x) - g(t, y)| \leq D|x - y|; \quad x, y \in \mathbb{R}^n, t \in [0, T]$$

(4.8)
for some constant D. Let Z be a random variable which is independent of the σ-algebra \(\mathcal{F}_\infty^{(m)}\) generated by \(B_s(\cdot), s \geq 0\) and such that \(E[|Z|^2] < \infty\).

Then, the stochastic differential equation

\[dX_t = b(t, X_t)dt + g(t, X_t)dB_t, \quad 0 \leq t \leq T, X_0 = Z\]  

(4.9)

has a unique \(t\)-continuous solution \(X_t(\omega)\) with the property that \(X_t(\omega)\) is adapted to the filtration \(\mathcal{F}_t^Z\) generated by \(Z\) and \(B_s(\cdot); s \leq t\)

\[E[\int_0^T |X_t|^2 dt] < \infty\]  

(4.10)

where Equation 4.7 is a linear-polynomial boundness condition on \(f\) and \(g\) while Equation 4.8 is the well-known Lipschitz condition.

The Lipschitz Condition 4.8 guarantees that \(f\) and \(g\) do not change faster with change in \(x\) than does the function \(x\) itself. This implies in particular the continuity of \(f(t, \cdot)\) and \(g(t, \cdot)\) for all \(t \in [t_0, T]\). It is easy to show that if \(f\) and \(g\) are linear functions, i.e., \(f(x) = Ax\), then both the Lipschitz and linear-polynomial boundness conditions are satisfied. Hence, linear SDEs always have a unique solution.

Another important question is whether the solution \(X_t\) of Equation 4.9 is a stationary Markov process. A necessary and sufficient condition for stationarity is that \(X_t\) must be homogeneous along with a series of more complex analytical conditions. We will omit the details here but the interested reader should consult [Kha69], p.97 for more details. In this case, the limiting semantics lead to a stochastic process with a stationary distribution for all \(t \in [t_0, T]\).

To summarize, in this chapter we have demonstrated the modeling power of PCN by showing it can model sequential as well as analog computation. Moreover, we introduced the notion of event generators and event synchronizers, central concepts that allow us to handle system with multiple clocks. Finally, we provide a formal limiting semantics for analog computation based on the well known results of SDEs and stochastic Taylor expansions.
Chapter 5

Models Subsumed by PCN

In this chapter we look at the various probabilistic modeling frameworks that are special cases of the PCN framework. Table 4.1 of Chapter 4 highlights the classification of the models based on the time and domain structure. For each of those four cases, we will show how these commonly used models can be equivalently represented by the PCN framework, hence demonstrating the flexibility of our framework.

5.1 Discrete time and discrete domain systems

The simplest case of a PCN model is one with discrete time structure along with a discrete domain structure: henceforth referred to as a DD-PCN. Note that a PCN model, even if restricted to these discrete-time/discrete-domain constraints, is still powerful and allows the modeling of a large class of systems. Discrete asynchronosity (multiple discrete clocks), as well as deterministic, non-deterministic or probabilistic behaviors are all properties that can be modeled within the DD-PCN framework. Furthermore, we will show that widely used models such as Markov Chains and Markov Decision Processes (in a later chapter) are in fact instantiations of the DD-PCN class of models.

5.1.1 DTMC to PCN conversion

The most commonly used discrete time/domain model for stochastic systems is the Discrete Time Markov Chains model (DTMC). Showing that every possible DTMC can be represented as a DD-PCN model is a trivial exercise. Indeed, given a discrete time Markov chain $M$ represented as
the tuple \( (S, s_0, \mathcal{P}) \), where \( S \), \( s_0 \) and \( \mathcal{P} \) represent the finite set of states (assume \(|S| = n\)), the initial state and the probability transition respectively, the equivalent DD-PCN \( M_{pcn} \) is simply \( (\{S\}, \delta(s_0), \mathcal{P}_{pcn}, C_{pcn}) \). As it can be seen, the set of locations contains one and only one location, \( S \), with domain \( \{1, 2, \ldots, n\} \), where each value of the location represents the encoding of each state in the DTMC. \( M_{pcn} \) has only one deterministic transduction, a unit delay \( \delta(s_0) \), and one generator following the probability distribution \( \mathcal{P} \). The unit delay \( \delta(s_0) \) is not only essential in avoiding an algebraic loop but it also allows us to model the Markovian property of the Markov chain within \( M_{pcn} \). The unit delay guarantees that the next state of the DD-PCN can only depend on the value of location \( S \) (state \( S_t \) depends only on state \( S_{t-1} \)). Note that the PCN framework also allows the modeling of Markov chains of Markovian order greater than one by simply using longer time delays: \( \delta_n, n > 1, n \in \mathbb{N} \). Furthermore, observe that the initial state \( s_0 \) is only used to set the value of \( S \) in the unit delay \( \delta(s_0) \) of the DD-PCN \( M_{pcn} \).

The only generator in \( M_{pcn} \) is equivalent to the probability transition matrix of the Markov chain \( \mathcal{M} \). That is, given the state value \( S_t \), the generator represents a probability distribution of the next possible state \( S_{t+1} \).

The set \( C_{pcn} \) contains three connections: 1) connecting \( \mathcal{P}_{pcn} \) to its output location \( S \); 2) connecting location \( S \) to the unit delay \( \delta(s_0) \); and 3) connecting the output of the unit delay to the generator \( \mathcal{P}_{pcn} \). Figure 5.1 shows the graphical representation of a DTMC as a DD-PCN.

### 5.1.2 DD-PCN equivalence to DTMC

We can show that any DD-PCN with the following characteristics can be converted into a DTMC:
1. all locations have finite domains
2. all delays are bounded
3. all transductions have finite memory

The transformation of a DD-PCN into a DTMC simply consists of obtaining a state space from the set of locations of the DD-PCN (along with their respective domains) and generating a set of transition probabilities from the set of generators of the DD-PCN.

We can show that a naive approach to transforming a DD-PCN into a DTMC will most likely yield a state space much larger than in fact is needed. An unnecessarily large state space would result in a transition probability matrix which is extremely sparse. We will present the intuition behind reducing the state space to a smaller size, while still keeping the correspondence between the DD-PCN and the DTMC.

Within the DD-PCN class, each model is represented by a finite set of locations \(Lc\) (each location with a finite domain), a finite set of deterministic transductions \(Td\), a finite set of generators \(G\), and a set of connections \(Cn\), linking locations with transductions and generators. To keep things interesting, we will assume that the set of generators is non-empty. Indeed, in the absence of probabilities, the system is purely deterministic and the current state of the system fully determines its next state, hence removing the need for a probabilistic analysis altogether.

Formally, in order to see if a DD-PCN is equivalent to a DTMC, one needs to obtain, from the components of the DD-PCN model, a state space \(S\), and a probability transition matrix which amounts to the transition probability per state \(s: P(s, s')\). The initial state of the DTMC will be dictated by the initial values of the delays within the DD-PCN.

The naive approach to generating the state space is to make it the cross-product of the domains of all locations. For instance, given a simple DD-PCN \(XYZ\) with three locations \(X, Y\) and \(Z\), and with respective domains \(D_x = \{0, 1, 2\}, D_y = \{1, 2\}, D_z = \{1, 2, 3, 4\}\), one would get a state space consisting of 24 states, each state equivalent to an element in \(D_x \times D_y \times D_z\). The first requirement on the structure of the DD-PCN, the one stipulating that the domain of each location be finite is essential to guarantee that the resulting cross-product (state space) is in fact finite.

Once one has a state space to work with, the only thing missing is the transition probability matrix. Since we are given the probabilistic transductions for each location, it becomes easy to extract
the transition probability for the induced state space. In order to maintain Markovian dynamics, we will assume that the DD-PCN of interest only possesses unit delays and transliterations (primitive transductions without memory nor internal state). This assumption is fairly restrictive and will soon be relaxed (while retaining the Markov property), once the intuition behind the concept of generating the transition matrix has been presented.

By using the information provided by the generators of the DD-PCN, we can obtain the transition probabilities quite easily. Let us demonstrate this by revisiting the DD-PCN $XYZ$ introduced above. This DD-PCN is shown in Figure 5.2.

In this system, two random variables ($X$ and $Y$) are summed and the result is called $Z$. This is a very simple system as neither probabilistic transduction depends on previous time steps, and it is made of only one deterministic transduction. Real probabilistic systems will often be composed of hundreds or even thousands of components, with the presence of unit and transport delays. However, the intuition gained from looking at the DD-PCN $XYZ$ will apply to those as well.

In order to obtain the transition probabilities, we first need the state space of the DTMC associated with this DD-PCN. As shown above, we simply compute the cross-product of the domains of locations $X$, $Y$ and $Z$. This gives us a state space of 24 states. Now to compute the transition probabilities, we need to take into account the fact that we are in the presence of a deterministic transduction, namely the Add transliteration, which simply returns the sum of its two inputs. This

\[ P(X=0) = 0.15 \]
\[ P(X=1) = 0.60 \]
\[ P(X=2) = 0.25 \]

\[ P(Y=1) = 0.4 \]
\[ P(Y=2) = 0.6 \]

\[ P(Z=1) = 0.06 \]
\[ P(Z=2) = 0.33 \]
\[ P(Z=3) = 0.46 \]
\[ P(Z=4) = 0.15 \]
transduction has the effect that it renders location \( Z \) fully dependent on locations \( X \) and \( Y \). Indeed, given the values of \( X \) and \( Y \), the value of location \( Z \) is fully determined (with probability 1). It becomes obvious that the probability that a state \( \{x, y, z\} \) will transition to another state \( \{x', y', z'\} \) is equivalent to the probability that at the next time step, \( X = x' \) and \( Y = y' \), multiplied by the indicator function \( I_z(z' = x' + y') \) (which yields 1 for \( z' = x' + y' \) and 0 otherwise). Hence the transition probability is simply the product of \( I_z \) with specific probabilities obtained from the probabilistic transductions of \( XYZ \).

It should be noted that most PCN models contain both types of transductions (probabilistic and deterministic). An interesting situation arises in the presence of deterministic transductions. In this case, there is full dependence between two locations which makes some transitions impossible. For example, within the DD-PCN \( XYZ \), we already mentioned that location \( Z \) is dependent on both locations \( X \) and \( Y \). Furthermore, as briefly discussed above, this dependence creates some constraints on the transitions of the systems. However, for this example as is the case for most DD-PCN models, we have more than one choice for a possible state space: in fact, for this example, we have three distinct choices. One could consider a state space consisting of all three locations (as presented above). One can also consider \( X \) and \( Y \) as defining the state space or alternatively consider only location \( Z \). Although the first choice appears to be the intuitive one, as it provides an easy way of generating a state space, it comes with the price of having to handle impossible transitions.

However, depending on the interests of the system designer, it might be valuable to only consider a subset of locations as being worthy of defining the state space. Assume for example that for the \( XYZ \) system, the designer is in fact only interested in the value of location \( Z \), and knowing the specific values of the two random variables \( X \) and \( Y \) are irrelevant to her.\(^2\) In this situation, it would make sense for her to build a state space based only on location \( Z \), hence reducing the state space from 24 states to merely 4 states, without any loss of information (at least what the designer considers relevant information). The transition probabilities for this new state space are easily obtained from the product of transition probabilities of locations \( X \) and \( Y \), under the assumption

\(^2\)This is only the case if location \( X \) and \( Y \) are both independent of any previous values. In the case where transduction \( P_X \) would have a delayed value of \( X \) as input, then the state space would require \( X \) to be present, as the transition probability for \( Z \) would be conditioned on the value of \( X \). Hence not all locations can be omitted from the state space, even if they are not considered important by the system's designer.
Observed Variables

Figure 5.3: Bayesian Network representation of the temporal dependence

that \( X \) and \( Y \) are independent at time \( t \).

In our example, it is easy to see that locations \( X \) and \( Y \) are in fact independent since none of the transductions depend on past values. However, even if both probabilistic transductions depended on the values of \( X \) and \( Y \) at the previous time step, \( X \) and \( Y \) would still be independent at time \( t \), given their values at time \( t - 1 \).

To show this, we can simply represent the temporal behaviour of the DD-PCN as a (Dynamic) Bayesian Network (Figure 5.3) and see that since the values of \( X_{t-1} \) and \( Y_{t-1} \) have been observed (evidently since we are now at time \( t \)), then the values of \( X_t \) and \( Y_t \) must be independent. Hence, to obtain the distribution of \( Z \), we can proceed using simple probabilistic rules.

So far, we have assumed that DD-PCNs are only composed of unit delays and transliterations. This was done in order to ensure that the transition probabilities possess the Markov property. However, it is possible to maintain the Markov property while greatly relaxing the assumptions on the components of the DD-PCN model. This relaxation is specified by the second and third characteristics of DD-PCN specified at the beginning of this sub-section.

Obviously, in the presence of unbounded delays, the Markov property would be totally lost, since the system would depend on an unbounded number of previous time steps. The same reasoning holds for transductions with infinite memory as this would assume that the transitions of the
system not only depend on the current state but also on the whole history up to the current time.

One may wonder why we only restrict the DD-PCN to bounded delays and finite memory transductions. Intuitively, a delay of 2 time steps appears not to respect the Markov property: the system depends on the current state but also on the previous state, and similarly with transductions that are not solely transliterations. However, there is a way around that problem. By including part of the history in the state space, one can convert a seemingly non-Markovian DD-PCN into a Markovian one. For instance, going back to our previous example of the DD-PCN with three locations, let us assume this time that the generators driving $X$ and $Y$ are dependent on $X_{t-2}$ and $Y_{t-1}$ respectively. This system, although very similar to the initial $XYZ$ system, is represented in Figure 5.4 for sake of clarity.

This new situation implies the presence of delays of three units for $X$ and two units for $Y$. By using a state space obtained from the cross-product of $D_x \times D_x \times D_x \times D_y \times D_y$, one now gets a state containing information for $X$ at three different times $\{X_t, X_{t-1}, X_{t-2}\}$ and for $Y$ at two different times $\{Y_t, Y_{t-1}\}$. The probability of transition from a given state $S_t = \{X_t, X_{t-1}, X_{t-2}, Y_t, Y_{t-1}\}$ to another state $S_{t+1} = \{X_{t+1}, X_t, X_{t-1}, Y_{t+1}, Y_t\}$ is simply the product of the two probabilities $P_x(X_{t+1}|X_{t-2})$, $P_y(Y_{t+1}|Y_{t-1}$, which can easily be obtained from the generators for $X$ and $Y$ and a set of indicator functions ensure that $X_t$ is now equal to $X_{t-1}$ once the transition has occurred.
5.2 Discrete time and continuous domain systems

5.2.1 DTMP to PCN conversion

Similarly to a DTMC, the PCN equivalent to a Discrete Time Markov Process (DTMP)\(^3\) is very simple. The only difference is that instead of a probability transition matrix, we have a probability measure over sets of states. The representation of such a DTMP is shown in Figure 5.5 where the domain of location \(S\) is continuous rather than discrete as it was the case for the DTMC conversion. Notice that the generator is now defined on a set of states \(A\), given the present state \(s'\), which demonstrates the fact that the location’s domain is continuous.

5.2.2 DC-PCN equivalence to DTMP

The class of discrete time, continuous domain PCNs, called DC-PCN, can be shown, in a similar way as with DD-PCN, to be equivalent to the class of discrete time Markov processes. We omit the details here as the extension to continuous domains, although more complex due to the presence of measures and dense domains, is straightforward.

\(^3\)The term Markov chains is used for discrete domains while Markov processes is the accepted terminology for continuous state spaces
5.3 Continuous time and discrete domain systems

In this section we will discuss the relationship between our framework and the paradigm most commonly used to model continuous time systems over discrete domains: Continuous Time Markov Chains (CTMC).

However, first, let us discuss the meaning of probabilities when in the presence of a continuous time structure. For continuous domains where the uncertainty is modeled as a stochastic process such as a Gaussian process or Brownian motion, the combination of the time continuum and probabilities is meaningful and easily understood since uncertainty evolves continuously with time. However, if one is studying systems with non-continuous domain structures, then one needs to clearly understand the meaning of probabilistic transitions. What does it mean for a system to evolve uncertainly, continuously over time, over a discrete set of values? Some desired properties of such probabilistic systems include:

1. The numbers of transitions in non-overlapping time intervals should be independent for all intervals. The occurrence of a specific transition should in fact be independent of when the previous one occurred.

2. The probability of two or more transitions occurring in a sufficiently small interval \( h \) should be \( o(h) \).

A way to handle such systems while guaranteeing that the above properties are satisfied is to have event-driven transductions, where events trigger a probabilistic transduction, i.e., a transduction containing a generator. Formally, we can show that the event generator should be modeled as a Poisson process, with the event being the arrival time. Recall that we introduced the Poisson process event generator in § 4.1.1 as represented by Equation 4.1.

For example, consider the simple probabilistic system \( XYZ \) of Figure 5.2, but now with a continuous time structure. The probabilistic transductions for \( X \) and \( Y \) are the same as before. However, since time is continuous, we have to ensure that we have a finite number of transitions within any finite interval \([t_1, t_2]\). Without this constraint, the system would be ill-behaved since locations would continuously transition from one value to another.

We therefore model the transition times for each probabilistic transduction \( X \) and \( Y \), as a Poisson process. Note that both \( X \) and \( Y \) can have different event generators, which in turn means that
only a subset of the variables would in fact be transitioning at one given time. Note also that in
general the event generator can be dependent on locations of the model. This allows for the rate of
transition to vary according to a subset of the variables constituting the state space of the system.
Hence a system can stay longer on average in one state while it might stay for a very short period
of time in another state. A PCN of such a probabilistic model can be shown to be equivalent to a
CTMC. Let us first define formally the notion of CTMCs, and then present how to obtain them from
a PCN model and vice-versa.

Formally, a continuous time Markov chain is a tuple \( M = \langle S, s_0, R \rangle \) where \( S \) is a finite set
of states, \( s_0 \) is the initial state, and \( R : S \times S \to \mathbb{R}^+ \) is known as the rate or intensity matrix.
Note that this general definition does not preclude the system from having self-loops. Self-loops are
represented by \( R(s, s) > 0 \) and denote the fact that a transition can occur while the system remains
in the same state. The inclusion of self-loops in the model might appear to diverge from the classical
definition of CTMC\(^4\). However, while they do not impair the expressibility of the paradigm, self-
loops allow the usual interpretation of linear-time operators from temporal logic. We will expand
on this notion in a later chapter when we introduce the probabilistic verification of PCN models.
Moreover, for simplicity, we will assume that the initial state of the system, \( s_0 \), is known with
probability one. This restriction can easily be lifted and the following discussion generalizes to the
case of a distribution over the initial state.

Intuitively, there exists a possible transition from state \( s \) to \( s' \) of \( M \) iff we have \( R(s, s') > 0 \).
The delay in transition between two states \( s \) and \( s' \) is governed by an exponential distribution with
rate \( R(s, s') \). Hence, the probability that a transition \( s \rightarrow s' \) happens within \( t \) time units is denoted
by the expression \( 1 - e^{-R(s, s') \cdot t} \). When the system can transition from a state \( s \) to multiple states
with different rates, i.e., \( R(s, s') > 0 \) for more than one \( s' \in S \), a race condition ensues and the race
is won by the minimum delay among the exponential distributions. Hence, \( E(s) = \sum_{s' \in S} R(s, s') \)
denotes the total rate at which any transition can occur at state \( s \) and leads to the conclusion that
the probability of leaving state \( s \) within \( t \) time units is \( 1 - e^{-E(s) \cdot t} \). With this, we can infer that the
probability of moving from state \( s \) to another state \( s' \) is determined by the probability that the delay
in evolving from \( s \) to \( s' \) is completed before the delays of the other possible transitions. Therefore,
we obtain \( P(s, s') = R(s, s')/E(s) \).

\(^4\)Although many other bodies of work allow the modeling of self-loops in their definition of CTMC,
among other, [BDH99]
5.3.1 CMTC to PCN conversion

Similarly to the previously discussed Markovian models, the conversion of a CTMC to a PCN is very straightforward. Given an $n$ state CTMC $M = (S, s_0, R)$, we can obtain an equivalent PCN $M_{pcn} = (\{S\}, \{Rate, P\}, C)$ where $S$ is the sole location of the system with domain($S$) = \{$s_0, s_1, \ldots, s_{n-1}$\} encoding the $n$ states of the system; $Rate$ is the event generator following an exponential distribution with state dependent rate $E(s)$ that triggers an event when the race condition has been completed; $P$ is the generator following the distribution $P(s, s') = R(s, s')/E(s)$ for all $s$ in domain($S$) which causes the system to transition probabilistically to a new state $s'$ when: 1) the system is in state $s$, and 2) an event signifying the completion of the race condition has occurred. This general situation, which applies to any CTMC with discrete state space, is represented in Figure 5.6.

5.3.2 PCN to CTMC

Let us now look at what class of PCN can in fact be mapped into a CTMC. This conversion is more complex, since for a general PCN, each location (one variable out of the set of variables constituting the state space of the system) can have a different rate of delay. When there is more than just one exponentially distributed event generator, only a subset of the locations can transition at one given time, hence causing a localized transition of the system. To keep things as general as possible, let us assume for the remainder of this section that each PCN location is obeying its own
exponentially distributed event generator, resulting in transitions in only one location for each event. Indeed, since time is continuous, we will assume that there cannot be multiple events arising from the exponentially distributed generators of the model. This assumption will become important when we combine the rate matrices of all the locations of the system. The case where multiple locations are related to the same event generator can easily be dealt with based on similar reasoning.

Without loss of generality, we can assume that, when building the state space of the equivalent CTMC, we only consider the locations that are outputs from a generator (i.e., locations equivalent to random variables). Any location which is an output of a deterministic transduction can be ignored as its value is merely a deterministic relabeling of the input values and hence does not affect the state space if ignored. Moreover, we assume that each generator \( g_i \in \mathcal{G} \) is event-driven and the event generators follow a Poisson process with an exponential distribution with rate \( \lambda_i \).

Assume that we are given a general PCN model of the form \( M_{pcn} = (Lc, Td, Cn) \) where \( Lc \) is a finite set of output locations, \( Td \) is a finite set of transduction labels (with \( \mathcal{G} \subset Td \) being a finite set of generators) and \( Cn \) is the set of connections between transductions, generators and locations. As mentioned above, we will only consider the set \( L_g \subset Lc \) of locations that are outputs of generators and we assume that \( |L_g| = n \). Therefore, it is easy to show that the state space of the equivalent CTMC \( M \), denoted by \( S_M \), will consist of the cross product of all the locations in \( L_g \), i.e., \( \times_{i \in L_g} i = 1, \ldots, n \). We then need to show that there exist a rate matrix \( R(\cdot, \cdot) \), which is equivalent to the dynamics of the PCN model.

Based on these assumptions on \( M_{pcn} \), we have each location of interest modeled in a way similar to the single location of the PCN shown in Figure 5.6. We can thus extract, for each location \( L_{g_i} \in L_g \), \( i = 1, \ldots, n \), the following information:

1. The event generator \( \text{Rate}(L_{g_i}) \) provides the conditional total rate at which any outgoing transition from location \( L_{g_i} \) is taken. Hence, in the CTMC framework, we have that \( \text{Rate}(L_{g_i}) = E(I(\text{Rate}(L_{g_i})), \text{Rate}(L_{g_i})) \) is the total transition rate for the \( L_{g_i} \) component of the state space \( S_M \), where \( I(\cdot) \) denotes the inputs of the transduction or generator.

2. From the distribution of the generator \( P_{L_{g_i}} \), one can obtain the probability of moving from the current state of the system to another value (possibly the same value since self-loops are allowed by design),
Therefore, knowing the relationship between the total rate $E(\cdot)$, the rate matrix $R(\cdot, \cdot)$ and the probability of transition $P(\cdot, \cdot_{\cdot})$, namely $P(s, s') = R(s, s')/E(s)$, we can easily obtain the conditional rate matrix, $R(\cdot, \cdot)$, for each $L_{gi} \in L_g$.

To complete the conversion, we need to combine the individual conditional rate matrices to obtain a rate matrix that would allow us to see the system as a single process. The method used to calculate the global rates is similar in essence to the amalgamation operator developed in the context of continuous time Bayesian Networks [NSK02], with the exception of a small modification needed to handle self-loops.

To obtain the whole system’s rate matrix, we first note that any transition which involves a change in more than one location will have an intensity of zero as no two locations can transition simultaneously. Then, similarly to the amalgamation operator, for a non-diagonal element of the rate matrix, one simply uses the intensity of the rate matrix for the associated location. Diagonal elements, which are equivalent to a self-loop need to be treated separately. Since any single transition from a state value to that same state value causes a self-loop, the transition rate corresponds to the minimum delay in which the locations constituting the state space will transition. Due to the fact that the transition delays are exponentially distributed, the minimum delay is hence obtained by summing all the rates for self-loops in each location. In this manner, we obtain the rate matrix for the system as a whole. Since we already built the state space as the cross-product of the domains of the PCN locations, this completes the conversion of a PCN into a standard CTMC.

5.4 Continuous time, continuous domain

In the previous chapter we discussed the notion of analog computation within PCN. Analog computation is modeled via stochastic differential equations (SDEs) which is a general model for continuous time and continuous domain systems. The general class of SDEs includes Markov processes and diffusion processes which, as mentioned earlier, can be modeled within the PCN framework. The wide area of SDE has been extensively studied over the years. For more details on the field, the reader is referred to the following books on SDEs and their applications [Oks98, Arn74, Mao97].
5.5 Other models of interest

Markov Decision Processes (MDPs) and their noisy observation counterpart Partially Observable Markov Decision Processes (POMDPs) are models that have received growing attention recently, especially in the field of artificial intelligence [Put94, BDH99]. Similarly to the other Markovian models presented in this chapter, these paradigms are also special cases of the PCN framework. However, since our main interest in MDPs and POMDPs lies in their use in obtaining optimal policies from the models, we will delay their introduction until the chapter on control synthesis of PCN models.
Chapter 6

Introduction to Behavioural Verification

The online satisfaction of the constraints imposed on the dynamics of the system by the model ensures that the system's behaviour will behave according to the stochastic process that corresponds to its solution. However, such constraint satisfaction does not guarantee that the behaviour of the system will satisfy global behavioural constraints. For example, consider the dynamics of an elevator system as described in Appendix B. Equation B.1 represents the constraints on the dynamics, essentially stating that the elevator must move obeying Newtonian laws of motion. Although this equation can represent very accurately the local behaviour of the system, it does not preclude the elevator from stopping halfway between two floors nor does it guarantee that a service request will always be successful in a timely manner. None of these requirements are explicitly specified in the model of the dynamics. In fact, such restrictions are global constraints on the behaviour of the system and cannot easily be represented within the PCN modeling framework. However, such global behavioural constraints are absolutely necessary when designing a dynamical system that will be used in practice. Hence, we need to define an appropriate requirements specification language that would allow the designer to specify global behavioural constraints. Moreover, we need to develop a formal method to verify the behaviour of a system with respect to a given requirements specification, thus guaranteeing that the system will in fact behave as it is designed to.

In Chapter 7 and 8 respectively, we develop average-timed \( \mathcal{A} \)-automata, and the Probabilistic Arbitrary-time Timed Temporal Logic (PATTL), two requirements specification languages for which
we provide formal verification methods.

6.1 Average-Timed $\forall$-Automaton

A popular method for representing behavioural constraints of systems is automata. This method is also well suited for the PCN framework as we can view traces as a generalization of infinite sequences. A desired property of the systems (hence the traces) can be specified by an automaton. That is, a trace of a system would satisfy the behavioural constraints iff the associated automaton accepts the trace.

Manna and Pnueli [MP87] first proposed $\forall$-automata and applied it to the specification and verification of concurrent programs. An extension to $\forall$-automata, timed $\forall$-automata, was proposed in [Zha94] and applied in the context of behaviour verification of dynamical hybrid systems.

In the next chapter, we begin by briefly introducing the notion of $\forall$-automata, adopted from the definition given in [MP87] and of timed-$\forall$-automata as introduced by Zhang and Mackworth [ZM96]. Second, we extend discrete timed-$\forall$-automata to discrete average-timed $\forall$-automata, by augmenting the automaton states with average time bounds. With this addition, we can reason about the average temporal behaviour of systems. Finally, we generalize our definitions to accept traces of arbitrary time structures.

6.2 Our Probabilistic Temporal Logic: PATTL

An alternative to automata for specifying behavioural constraints is temporal logic. Probabilistic logics provide a simple yet powerful specification methodology for (dynamical) systems with temporal behaviour. We develop the Probabilistic Arbitrary-time Timed Logic (PATTL) where Timed refers to the notion of temporal evolution of the systems and Arbitrary-time denotes the ability to handle systems with arbitrary clocks. This logic is intended to specify quantitative probabilistic behavioural constraints applied to a PCN model of a system. For example, consider a coffee-delivery robot which upon request from his master, must fetch and deliver coffee. The robot's master might want to reason about the overall quality level of his robotic butler. This level could be defined by the waiting time to receive a cup of coffee once the robot has been summoned. Furthermore, the master might be interested in maintaining a certain basic quality level at all times but might agree, as a cost
saving strategy, to accept that the probability of the quality level dropping below the basic level for a short period of time be less than 10%. In industrial applications, this requirement is often referred to as quality of service and is a very important requirement for many applications. It is essential to be equipped with a specification language that is powerful enough to allow us to explicitly state such requirements but also offers computational efficiency so that one can verify that the desired requirements for real-world practical applications are indeed satisfied.

The PATTL logic is based on a well-known logic and extends an already existing logic to account for arbitrary time and domains. Specifically, PATTL is a probabilistic temporal logic arising from the Computational Tree Logic (CTL) [Eme90, EC82], a branching time logic, on which some of the most used probabilistic logics are based. While CTL is not expressive enough to specify quantitative properties of uncertainty, it has a natural correspondence with the computation of concurrent and non-deterministic programs. The fact that CTL is a branching time logic rather then a linear time logic makes it a good candidate when investigating probabilistic temporal logics. Indeed, the underlying structure of time being tree-like provides an ideal framework to see each branch as a different probabilistic (or non-deterministic) successor of the current node. In our probabilistic approach to systems, instead of seeing time as having a tree structure, we view the branching property of such logic as a representation of each possible event \( \omega \) taken from the event space \( \Omega \). We will elaborate on this notion later in subsequent chapters. However, for a more in depth (and classical) comparison of linear time logic and branching time logic, the reader is referred to [Eme90].

Recently, with the increasing number of practical applications believed to exhibit inherent uncertainty, the need to specify quantitative, rather than qualitative, probabilistic properties has lead to the development of full probabilistic logic, most of them directly extending the CTL framework. We will base PATTL on the Probabilistic Computational Tree Logic (PCTL) of [HJ94], which we generalize to obtain a logic for arbitrary time and domain structures to allow formulae to be interpreted over Probabilistic Constraint Nets.

### 6.3 Behavioural Verification

In the previous chapters, we presented the syntax and semantics of Probabilistic Constraint Nets, a framework for modeling probabilistic hybrid systems. We showed that the PCN framework models a large class of systems and subsumes some of the most commonly used modeling framework such
as Markov processes and (PO)MDP. We argued that the PCN framework is a general and representationally very powerful modeling framework which can be of great use for a system modeler. In the following chapters, we introduce the notion of average-timed V-automata and extend the logics CTL and PCTL to define the PATTL temporal logic. Both average-timed V-automata and PATTL constitute specification languages which we will use to specify behavioural constraints on systems.

At this time, the difference between system modeling and the specification of behavioural constraints might still be unclear. Although they might appear similar, these two notions are inherently very different. The modeling task focuses mainly on the dynamics of the systems and how different components interact together. Essentially, it imposes local constraints on the systems dynamics. On the other hand, the specifications of a system impose global constraints on its behaviours. For example, the dynamics of a mobile robot can be modeled by differential equations following basic laws of physics such as the relation between velocity and acceleration ($v = at$ or $s = 1/2at^2$). These laws represent the constraints on the dynamics. However, although these represent well the local behaviour of the system, it does not preclude the robot from hitting people as it is roaming around nor does it guarantee that a delivery robot will always be successful when attempting to deliver coffee to its "master's" office. Such restrictions are global constraints on the behaviours of the system and cannot easily be represented within the PCN modeling framework. They can, however, be compactly expressed with a PATTL or a $\forall$-automaton specification. For example, the coffee delivery requirement can be represented in PATTL as $\Box\Diamond \text{Deliver\_Coffee}$, meaning that the robot will always ($\Box$) eventually ($\Diamond$) be successful at delivering coffee. The $\forall$-automaton specification is represented in Figure 6.1.
Once one is equipped with a model of the dynamics of a system (via the PCN framework in our case although many other modeling languages have emerged over the years) and with a requirements specification of the global behaviour of the system (either in $\forall$-automaton or PATTL), a key question is to ask whether the behaviour of the system satisfies these requirements. This is called \textit{behavioural verification}. In the next two chapters, we will present behavioural verification procedures when armed specifically with average-timed $\forall$-automaton and PATTL specifications.
Chapter 7

Behavioural Verification with Average Timed $\forall$-Automaton

In this chapter, we augment the notion of $\forall$-automata behavioural verification to *average-timed* $\forall$-automata for stochastic dynamical systems. We first discuss the relation between $\forall$-automaton and stochastic systems. The notion of behavioural verification in this context is not as straightforward as with deterministic systems. We will then provide a simple introduction to timed $\forall$-automaton for discrete time structure and proceed to augment it to average-timed $\forall$-automaton for a similar time structure by specifying a set of *on average* constraints on automata states. Finally we generalize discrete average-timed $\forall$-automaton to average-timed $\forall$-automaton whose time structure can be arbitrary.

### 7.1 $\forall$-Automata

$\forall$-automata are non-deterministic finite state automata which can be used to specify requirements for concurrent programs [MP87] or time traces from deterministic dynamical systems [Zha94]. We present the definitions surrounding $\forall$-automaton along with the classical notion of acceptance of traces (the definitions are reproduced from Section 10.1 of [Zha94]). We then carry on by defining the acceptance of traces induced by a stochastic dynamical system. Note that this is not meant to be a comprehensive survey of $\forall$-automata and their related notions. For more details the reader is referred to Chapter 10 of [Zha94].
Definition 7.1 (Syntax of \( \forall \)-automata) A \( \forall \)-automaton \( A \) is a quintuple \( \langle Q, R, S, e, c \rangle \) where \( Q \) is a finite set of automaton states, \( R \subseteq Q \) is a set of recurrent states and \( S \subseteq Q \) is a set of stable states. With each \( q \in Q \), we associate a state proposition \( e(q) \), which characterizes the entry condition under which the automaton may start its activity in \( q \). With each pair \( q, q' \in Q \), we associate a state proposition \( c(q, q') \), which characterizes the transition condition under which the automaton may move from \( q \) to \( q' \).

\( R \) and \( S \) are the generalization of accepting states to the case of infinite inputs. We denote by \( B = Q - (R \cup S) \) the set of non-accepting (bad) states.

Let \( T \) be a discrete time structure, \( A \) be a domain and \( v : T \rightarrow A \) be a trace. A run of \( A \) over \( v \) is a mapping \( r : T \rightarrow Q \) such that (1) \( v(0) \models e(r(0)) \); and (2) for all \( t > 0 \), \( v(t) \models c(r(\text{pre}(t)), r(t)) \).

We call a \( \forall \)-automaton complete iff we have

- \( \forall_{q \in Q} \ e(q) \) is valid
- For every \( q \in Q \), \( \forall_{q' \in Q} \ c(q, q') \) is valid.

For the remainder of this chapter, we will only consider complete automata as any automaton can be transformed into a complete automaton by introducing an error state. When displaying incomplete \( \forall \)-automaton, we will assume that the error state is implicitly present, thus simplifying the representation. Obviously, any complete automaton guarantees that any discrete time trace has a run over it.

If \( r \) is a run then let \( \text{Inf}(r) \) denote the set of automaton states which appears infinitely often in \( r \). That is, \( \text{Inf}(r) = \{ q | \forall t, \exists t_0 \geq t, r(t_0) = q \} \). If \( T \) has a greatest element \( t_0 \) then we define \( \text{Inf}(r) = \{ r(t_0) \} \). Therefore, \( \text{Inf}(r) \) can be seen as a generalization of the "final value" of a system.

Let \( A \) be a \( \forall \)-automaton. A run \( r \) of \( A \) is defined to be accepting iff it satisfies one of the two conditions:

1. \( \text{Inf}(r) \cap R \neq \emptyset \), i.e., some of the states appearing infinitely many times in \( r \) belong to \( R \), or
2. \( \text{Inf}(r) \subseteq S \), i.e., all the states appearing infinitely many times in \( r \) belong to \( S \).
Essentially, the notion of acceptance of traces states that in the long run, the system either always returns to the set of recurrent states $R$ or it will remain forever within the stable set $S$. Systems which cannot guarantee this are deemed unsatisfactory for the requirements specified by the behavioural constraints. Based on this requirement, the semantics of $\forall$-automata follow:

**Definition 7.2 (Semantics of $\forall$-automata)** A $\forall$-automaton $A$ accepts a trace $v$, written $v \models A$, iff all possible runs of $A$ over $v$ are accepting.

One should note that these semantics differ in the way they handle non-determinism from the semantics of conventional automata, with which the reader might be more accustomed. A conventional automata $C$, which could, in this context, also be called a $\exists$-automata, accepts a language if there exists at least one run over $C$ which is accepting. However, in the context of behaviour verification, having at least one run satisfying the requirements is obviously not a strong enough statement as in the case of a safety requirement, this is generally not what we define as a safe system.

For deterministic systems, which are defined completely by a single trace, it is meaningful to require the trace be accepted. However, when modeling a stochastic system, asking for all traces to be accepted (which we referred to as satisfying the requirements at level $\alpha = 1$) might be too demanding. Indeed, there might be a very small probability that the system will move into a set of absorbing bad states, hence never satisfying the behavioural constraints. However, if this probability (which is equivalent to the measure of all sample traces leading to the absorbing bad states) is small enough so that these events rarely occur, one might be willing to accept the risk to work with a system which satisfy the requirements at a level $\alpha$ where $\beta \leq \alpha < 1$ and $\beta$ is the safety threshold.

Although, for certain systems, we will be interested in satisfying some behavioural constraints at levels $\alpha < 1$, we will postpone doing so until we discuss verification procedures with PATTL. The techniques for verifying $\forall$-automata specifications which will be presented shortly are better suited for level $\alpha = 1$ verification. Therefore, we will limit ourselves to systems for which verification at level $\alpha = 1$ is meaningful.

At this point it might be helpful to motivate the notion of verification at level $\alpha = 1$ of a stochastic dynamical systems. What type of restrictions on the system itself does this create? Intuitively, perfect satisfaction of a set of behavioural constraints amounts to the system not possessing any absorbing bad states. By absorbing we refer to the case where the system enters this bad state and
never leaves it. In practice, for a system to not possess any absorbing bad states requires that for any state of the stochastic dynamical system associated with a bad automaton state, there must exist a path with positive probability which leads to an accepting state (associated to either $R$ or $S$). We will formally prove this result when introducing the verification rules later in this section.

While requiring that systems do not have absorbing bad states may appear to be overly restrictive, we can apply a simple transformation to the state space of the system to remove such states. Indeed, for a large class of systems with absorbing bad states, these states correspond to a situation where the robotic agent is down in one way or another. Hence, repair or restart would be needed to ensure that the system can continue operating. One could take this “repair” into account and modify the state space so that once the agent arrives to a absorbing bad state, a transition occurs with probability one which relocates the agent to “restart” state. This simple modification removes absorbing bad states and thus allows the verification method to be applied to a vast class of systems.

For economy of space, we refer the reader to §10 of [Zha94] for a comprehensive introduction to behavioural verification with $\forall$-automata.

Let us now present some common behavioural constraints of dynamical systems. Figure 7.1(a) represents a specification which accept the traces of a system which eventually will always satisfy the goal condition $G$. This is equivalent to $\Diamond \Box G$ in temporal logic. Figure 7.1(b) is a safety constraint which states that an accepted system should never satisfy the unsafe condition $B$. An equivalent temporal logic representation would be $\Box \neg B$. Finally, Figure 7.1(c) is a bounded response constraint. It states that whenever event $E$ occurs, the response $R$ will occur in bounded time. This requirement is slightly more complicated as indicated by the temporal logic equivalent: $\Box (E \rightarrow \Diamond R)$.

Although $\forall$-automata are not equivalent in expressive power to temporal logic such as TLTL, these small examples demonstrate the simplicity and intuitiveness of $\forall$-automata for the specification of behavioural constraints.

### 7.1.1 Average-timed automata

Meaningful behavioural constraints often encompass temporal components. Consider Figure 7.1(c), where one might be interested in a system satisfying a bounded response specification where the time bound is a known finite constant. In order to represent timeliness of behavioural constraints,
Timed $\forall$-automata were proposed [ZM96]. Timed $\forall$-automata augment basic $\forall$-automata with timed automaton states and time bounds. This approach, however, is ill-suited for stochastic dynamical systems. Since we are interested in solving behavioural constraints on stochastic systems, we cannot talk about satisfying a given time constraint in an absolute way but rather we need to reason about satisfying that time constraint on average. We mentioned earlier that in order for a system to be accepting by a $\forall$-automata specification, it needs not have any absorbing bad states. This is characterized by the limiting behaviour: $\lim_{t \to \infty} \Pr(X_t \in S \cup R | X_0 \in B) = 1$. Although this requirement is sufficient to guarantee that for any run $r$ over a given trace, $\inf(r) \cap R \neq \emptyset$ or that $\inf(r) \subseteq S$, it does not guarantee that it will happen in a finite time for every trace. However, for systems without any absorbing bad states, we are assured that the average time will be bounded, as stated in the proposition below. We prove the result for finite state space, but the result can be extended to countable state space at the price of a slightly more complicated proof.

**Proposition 7.1** Assume a finite state space $S$, and assume that, for all the bad states $B$, there is a positive probability of moving toward an accepting state $R$ or $S$, i.e., the set of bad states is irreducible. Define $\zeta_{s'}^s$ as the time needed to reach state $s'$ from state $s$. Then, $E(\zeta_{s'}^s | b \in B, s \in R \cup S) < \infty$.

A logical extension of time constraints is average time constraints. The idea behind average time constraints is that although we cannot prove that a stochastic dynamical system can always satisfy some given time constraint, we can show that the average behaviour of the system does satisfy the constraints. This is similar to the well-known concept of sample paths and expected sample paths of stochastic analysis. For completeness of this discussion, we summarize the notion of timed-$\forall$-automata prior to introducing the definitions of average-timed $\forall$-automata.
Definition 7.3 (Syntax of timed $\forall$-automata) A timed $\forall$-automaton $TA$ is a triple $(A, T, \tau)$ where $A = (Q, R, S, e, c)$ is a $\forall$-automaton, $T \subseteq Q$ is a set of timed automaton states and $\tau : T \cup \{\text{bad}\} \rightarrow \mathbb{R}^+ \cup \{\infty\}$ is a time function.

It is easy to show that any $\forall$-automaton is equivalent to a special timed $\forall$-automaton with $T = \emptyset$ and $\tau(\text{bad}) = \infty$. Graphically, a $T$-state is denoted by a nonnegative real number indicating its time bound. The conventions for complete $\forall$-automata are adopted for timed $\forall$-automata.

Let $v : T \rightarrow A$ be a trace. We define a run $r$ of $TA$ over $v$ has being a run of $A$ over $v$; $r$ is accepting for $TA$ iff

1. $r$ is accepting for $A$ and
2. $r$ satisfies the time constraints. If $I \subseteq T$ is an interval of $T$ and $q^* : I \rightarrow Q$ is a segment of run $r$, i.e., $q^* = r|_I$, let $\mu(q^*)$ denote the measure of $q^*$, i.e., $\mu(q^*) = \mu(I) = \sum_{t \in I} \mu(t)$ since $I$ is discrete. Furthermore, let $\mu_B(q^*)$ denote the measure of bad automaton states in $q^*$, i.e., $\mu_B(q^*) = \sum_{t \in I, q^*(t) \in B} \mu(t)$. Let $Sg(q)$ be the set of segments of consecutive $q$'s in $r$, i.e., $q^* \in Sg(q)$ implies $\forall t \in I, q^*(t) = q$. Let $BS$ be the set of segments of consecutive $B$ and $S$-states in $r$, i.e., $q^* \in BS$ implies $\forall t \in I, q^*(t) \in B \cup S$. The run $r$ satisfies the time condition iff

(a) (local time constraint) $\forall q \in T, q^* \in Sg(q), \mu(q^*) \leq \tau(q)$ and
(b) (global time constraint) $\forall q^* \in BS, \mu_B(q^*) \leq \tau(\text{bad})$.

The first condition stipulates that for a local time constraint, the system will not stay continuously in a given state $q \in T$ for longer than its local time bound $\tau(q)$. The second condition requires the system to leave the set of bad states within $\tau(\text{bad})$ time units.

Definition 7.4 (Semantics of timed $\forall$-automata) A timed $\forall$-automaton $TA$ accepts a trace $v$, written $v \models TA$, iff all possible runs of $TA$ over $v$ are accepting.

We now present the syntax and semantics of average-timed $\forall$-automata, which extend the definitions for timed $\forall$-automata presented above. Average-timed $\forall$-automata allow for the verification of behavioural constraints for systems exhibiting uncertainty.
Definition 7.5 (Syntax of average-timed \( \forall \)-automata) An average-timed \( \forall \)-automaton \( \mathcal{ATA} \) is a triple \( (A,T,\tau) \) where \( A = (Q,R,S,e,c) \) is a \( \forall \)-automaton, \( T \subseteq Q \) is a set of average-timed automaton states and \( \tau : T \cup \{\text{bad}\} \rightarrow \mathbb{R}^+ \cup \{\infty\} \) is an average-timing function.

Once again, we can easily show that any \( \forall \)-automaton is equivalent to a special average-timed \( \forall \)-automaton with \( T = \emptyset \) and \( \tau(\text{bad}) = \infty \). A \( T \)-state is denoted by a nonnegative real number indicating its average-time bound. However, unlike with typical \( \forall \)-automata, or even timed-\( \forall \)-automata, we cannot define the acceptance of a single trace by an average-timed \( \forall \)-automata. In fact, due to the stochastic nature of the systems of interest, we are no longer interested in the behaviour exhibited by individual traces but rather in the behaviour of a set of traces. Expected time constraints should be satisfied by the average behaviours of systems, hence we need to look at the ensemble of traces induced by those systems.

Let \( B \) be the behaviour of a system. We define a run \( r \) of \( \mathcal{ATA} \) over \( B \) has being a run of \( A \) over every trace \( v : T \rightarrow A \) in the behaviour \( B \). A run \( r \) is accepting for \( \mathcal{ATA} \) iff

1. \( r \) is accepting for \( A \) and
2. \( r \) satisfies the expected time constraints. If \( I \subseteq T \) is an interval of \( T \) and \( q^* : I \rightarrow Q \) is a segment of run \( r \), i.e., \( q^* = r_{|I} \), let \( \mu(q^*) \) denote the measure of \( q^* \), i.e., \( \mu(q^*) = \mu(I) = \sum_{t \in I} \mu(t) \) since \( I \) is discrete. Furthermore, let \( \mu_B(q^*) \) denote the measure of bad automaton states in \( q^* \), i.e., \( \mu_B(q^*) = \sum_{t \in I,q^*(t) \in B} \mu(t) \). Let \( Sg(q) \) be the set of segments of consecutive \( q \)'s in \( r \), i.e., \( q^* \in Sg(q) \) implies \( \forall t \in I, q^*(t) = q \). Let \( BS \) be the set of segments of consecutive \( B \) and \( S \)-states in \( r \), i.e., \( q^* \in BS \) implies \( \forall t \in I, q^*(t) \in B \cup S \). The run \( r \) satisfies the time condition iff
   
   (a) (local time constraint) \( \forall q \in T, q^* \in Sg(q) \), \( \mathbb{E}(\mu(q^*)) \leq \tau(q) \) and
   
   (b) (global time constraint) \( \forall q^* \in BS \), \( \mathbb{E}(\mu_B(q^*)) \leq \tau(\text{bad}) \).

where \( \mathbb{E}(\cdot) \) denotes the expectation over all traces \( v \) of \( B \).

Definition 7.6 (Semantics of average-timed \( \forall \)-automata) An average-timed \( \forall \)-automaton \( \mathcal{ATA} \) accepts a set of traces \( B \), written \( B \models \mathcal{ATA} \), iff all possible expected runs of \( \mathcal{ATA} \) over \( B \) are accepting.
As an example, Figure 7.2 depicts the real-time response constraint which states that $R$ will be reached within 40 time units of $B$, where the time in $S$ is not accounted for.

### 7.2 Model-Checking Approach

Before formally discussing the notion of behavioural constraint verification, it is necessary to discuss the relationship between stochastic dynamical systems and their behaviours. Intuitively, the behaviour of a stochastic dynamical system is the set of observable input/output traces of a given system. Let $\mathcal{P}(I, O)$ be a PCN module, where $(I, O)$ is the tuple of input and output locations of the module. Formally, an input/output pair $(i, o)$ is an observable trace of $\mathcal{P}(I, O)$ iff $\exists F \in [\mathcal{P}(I, O)]$ such that $o = F(i)$. The reader should note that, within the PCN framework, the function $F$ can be deterministic, non-deterministic or probabilistic, depending on what type of transductions (deterministic or probabilistic) and locations (hidden or not) are used to model the system. We define the behaviour of $\mathcal{P}(I, O)$ as the set of all observable traces and we denote it as $[\mathcal{P}(I, O)]$. We will abbreviate $[\mathcal{P}(I, O)]$ to $[\mathcal{P}]$ if $I = I(\mathcal{P})$, $O = O(\mathcal{P})$ and no ambiguity arises.

The notion of equivalency of PCN modules stems directly from their behaviour. Two PCN modules, $\mathcal{P}_1$ and $\mathcal{P}_2$, are equivalent, denoted $\mathcal{P}_1 \simeq \mathcal{P}_2$, iff they exhibit the same behaviour: $[\mathcal{P}_1] = [\mathcal{P}_2]$.

Now that we have formally defined the behaviour of a system, we introduce the definitions of time-invariant and Markovian behaviours.

**Definition 7.7 (Time-Invariant)** Let $\mathcal{B} = \{v | v : \mathcal{T} \rightarrow A\}$ be a behaviour. $\mathcal{B}$ is a time-invariant
behaviour\(^1\) if for any \(a_1, a_2 \in A, \forall v \in B\) such that \(v(t_1) = a_1\) and \(v(t_1 + s) = a_1\) then \(\Pr(v(t_2) = a_2|v(t_1) = a_1) = \Pr(v(t_2 + s) = a_2|v(t_1 + s) = a_1)\) for \(t_1 < t_2 \in T\).

A PCN model is time invariant iff all transductions (deterministic or probabilistic) from the set of transductions are independent of the time parameter \(t\).

Another important type of behaviour, which has been studied extensively, is called Markovian behaviour.

**Definition 7.8 (Markov Property)** Let \(B\) be a behaviour. Given any time point \(t \in T\), \(v \in B\) and \(a \in A\), such that \(v(t) = a\), the behaviour is called Markovian (of order 1) if the probability distribution of the next state, \(v(t + 1)\), is independent of the past history of the system except for the state \(v(t) = a\).

More generally, a behaviour is Markovian of order \(n\) if the probability of the next state depends only on the \(n\) previous states. We will say that a system is strictly Markovian if it is Markovian of order 1.

**Proposition 7.2 (Markovian PCN)** Observe a PCN model at the finest level of details, i.e., looking at the network created by the connections between locations and basic transduction (transliterations, delays and generators). A PCN model induces a Markovian behaviour of order \(n\) iff all the delays, for which the output location is included in the state space, are of finite length of at most \(n\) or if the sum of the strictly consecutive delays with output location not included in the state space is at most \(n\).

Recall that any Markovian system of finite order can be transformed into a strictly Markovian system. We can do so by augmenting the domains of the locations to contain a finite history of length equal to the order of the original system. Hence this allows us to solely consider strictly Markovian systems without loss of generality.

These two type of behaviours constitute a very important class of behaviours and a vast number of physical systems exhibit them. In this chapter, we are interested in analyzing the behaviour of time invariant and Markovian systems. As we mentioned earlier, the PCN framework is very general and encompasses a much larger class of behaviour. We will thus restrict ourselves to the sub-class of PCN which are time invariant and Markovian.

\(^1\)This is sometimes also referred to as time homogeneous behaviour.
The class of discrete time stochastic dynamical systems exhibiting a time-invariant Markovian behaviour is very vast. In fact, any discrete-time, time invariant Markovian stochastic dynamical system corresponds to what we call a stochastic transition system. Since we are interested in the verification of properties specified as an automaton, we restrict ourselves to discrete-time.

A stochastic state transition system is a tuple \( \langle S, P, \Theta \rangle \) where \( S \) is a set of states, \( P : S \times S \) is an evolution kernel representing the transition probability distribution between two states, i.e., \( P(s_1, s_2) \) is the probability of a transition occurring between \( s_1, s_2 \in S \) and \( \Theta \) represents the distribution of the initial state of the system\(^2\). Notice that due to the time invariance and Markovian properties, \( P \) is independent of the time parameter and transitions depend only on the current state. For any discrete time structure \( T, v : T \rightarrow S \) is a trace of \( \langle S, P, \Theta \rangle \) iff \( \forall t > 0, P(v(\text{pre}(t)), v(t)) > 0 \), where \( \text{pre}(t) \) represents the time value preceding \( t \). We will denote an allowed transition from \( v(\text{pre}(t)) \) to \( v(t) \) by \( v(\text{pre}(t)) \rightsquigarrow v(t) \). A behaviour \( B \) corresponds to a stochastic state transition system \( \langle S, P, \Theta \rangle \) iff \( B \) is equal to the set of all traces of \( \langle S, P, \Theta \rangle \). Stochastic state transition systems constitute a compact representation of time invariant Markovian behaviours.

Let us now introduce the notion of probability measure on a behaviour. To do so, we introduce the notion of a Borel space on traces of a system, which follows [HJ94, BHHK03]. Let \( B \) be the behaviour of a stochastic state transition system \( STS = \langle S, P, \Theta \rangle \). A distribution \( \Theta \) on the initial state of \( STS \) induces a probability measure \( \mu_\Theta \) on its traces in the following way. First let \( s_0, s_1, \ldots, s_n \in S \) with \( s_i \rightsquigarrow s_{i+1} \), \( 0 \leq i < n \). Then denote \( C(s_0, s_1, \ldots, s_n) \) by the cylinder set of all traces \( v \in B \) such that \( v(0) = s_0, \ldots, v(n) = s_n \). Define \( \mathcal{F}(B) \) to be the smallest \( \sigma \)-algebra on the behaviour \( B \) which contains all cylinder sets \( C(s_0, \ldots, s_n) \). The probability measure \( \mu_\Theta \) on \( \mathcal{F}(B) \) is the unique measure defined by induction on \( n \) with base case \( n = 0 : \mu_\Theta(C(s_0)) = \Theta(s_0) \) and induction hypothesis for \( n \geq 0 \):

\[
\mu_\Theta(C(s_0, \ldots, s_n, s_{n+1})) = \mu_\Theta(C(s_0, \ldots, s_n)) \cdot P(s_n, s_{n+1})
\]

We define a behavioural constraint (or requirements specification) \( B_C \) for a stochastic system \( STS = \langle S, P, \Theta \rangle \) as a set of allowable input/output traces of the system, i.e., \( B_C \subset \times_{t \in T} A_t \). Let \( B = [STS] \) be the behaviour of \( STS \). We say that \( STS \) satisfies the behavioural constraints \( B_C \) at a level \( \alpha \), denoted by \( B \models_{\alpha} B_C \) iff \( \mu(B \cap B_C) = \mu(\{v \in B | v \models B_C\}) \geq \alpha \), where \( v \models B_C \)

\(^2\)This does not preclude the initial state of the system to be set in advance.
is the predicate for \( v \) satisfying the behavioural constraint \( B_C \). Perfect satisfaction of behavioural constraints is indicated by \( \alpha = 1 \), which means that all traces of \( B \) (with the possible exception of a zero measure set) are allowable traces. Satisfaction at a level of \( \alpha < 1 \) results in a subset of the traces of \( B \) being undesirable.

Obviously, a satisfaction at level \( \alpha = 0 \) is equivalent to a total absence of satisfaction, i.e., *none* of the possible traces of the system will ever satisfy the constraint \( B_C \) (\( \mu = 0 \Rightarrow (B \cap B_C) = \emptyset \)). Behavioural constraint satisfaction of deterministic system is equivalent to satisfaction at level \( \alpha = 1 \). However, in the presence of probabilistic behaviour, requiring satisfaction at level \( \alpha = 1 \) might be too strict as one might be willing to accept a small risk of not satisfying the requirement (e.g., \( \alpha = 0.95 \)), rather then flat out rejecting the system. For the remainder of this chapter, we will elaborate a method which performs behavioural constraint satisfaction at level \( \alpha = 1 \) for untimed constraints while temporal behavioural constraints will be satisfied on average. Methods for behavioural specification of average-time \( \forall \)-automata properties at any level \( \alpha < 1 \) are being investigated and constitute future work. In Chapter 8, we will develop methods for verification at arbitrary level \( \alpha \).

Now that we formally defined the notion of behavioural verification at level \( \alpha \), we can introduce the notion of robustness and complexity of systems. The *robustness* of a system is a notion defined on parameterized probabilistic constraint nets. We say that a parameterized system \( P_1^P \) is less robust than a second system \( P_2^P \) with respect to a behavioural constraint \( B_C \), denoted by \( P_1^P \preceq_{B_C} P_2^P \), iff \( \forall p \in \times P D_P, \llbracket P_1^P \rrbracket (p) \models_\alpha B_C \Rightarrow \llbracket P_2^P \rrbracket (p) \models_\beta B_C \), for \( \alpha \leq \beta \). The two systems above are equivalent w.r.t. \( B_C \), written by \( P_1^P \equiv_{B_C} P_2^P \), iff \( P_1^P \preceq_{B_C} P_2^P \) and \( P_2^P \preceq_{B_C} P_1^P \). In this case, both systems would satisfy the behavioural constraint at level \( \alpha = \beta \). Note that this definition of equivalence is somewhat more subtle than for deterministic systems. Equivalence of deterministic systems requires that the behaviours of the two systems be the same. This implies that the traces of the two systems are exactly the same. For probabilistic systems, we relax this assumption by requiring only that the measure over allowable traces be the same for both systems. However, it is easy to construct two equivalent systems for which their respective set of allowable traces is different, even though both sets have equal measure.

Let us now define the complexity of a behaviour. Behavioural complexity is defined with respect to a given measurement on the size of a stochastic dynamical system. This measurement could be
the number of transductions, the number of delays or the maximum number of delay element in any path (which is equivalent to the order of the Markovian property of the system). Therefore, given a measurement $\kappa$, denote $|P|_\kappa$ as the size of the system $P$ with respect to $\kappa$. We then define the complexity of the behaviours satisfying the requirements specification $B_C$, w.r.t. $\kappa$ and level $\alpha$, written $|B_C|^\alpha_\kappa$, to be the smallest stochastic dynamical system which respects $B_C$. That is, $|B_C|^\alpha_\kappa = min\{|P|_\kappa | P =_{\alpha} B_C\}$.

We have defined above the concepts of the behaviour of a system and of behavioural constraints. Given the behaviour $B$ of a stochastic dynamical system and a behavioural constraint $B_C$, the behaviour satisfies the requirements at level $\alpha$, written as $B =_\alpha R_S$, iff $\mu(\{v \in B | v = R_S\}) = \alpha$. As mentioned before, we will restrict ourselves to the satisfaction of the behavioural constraints of a system at “average” level only. Therefore, given the probabilistic constraint net model of a system and an average-timed $\forall$-automaton specification of behavioural constraints, we say that the behaviour of the system satisfies the behavioural constraints if and only if the all traces of the system are accepting for the average-timed $\forall$-automaton.

The formal behaviour verification method consists of a set of model-checking rules. The rules are a generalization of the rules for dynamical systems developed by [Zha94], which themselves extended the rules developed for concurrent programs [MP87].

### 7.2.1 Behavioural Constraint Verification Rules for Discrete-time

For sake of simplicity, we will introduce the verification rules for the simplest possible situation: discrete time and discrete domain. A generalization of the rules for arbitrary time and domain will follow shortly.

As we mentioned earlier, any time-invariant Markovian behaviour $B$ in discrete time corresponds to a stochastic state transition system $<S_B, P, \Theta>$ for which we denoted an allowed transition from state $s$ to state $s'$ by $s \sim s'$. We also write $\{\varphi\}B\{\psi\}$ iff the consecutive condition: $\varphi(s) \land (s \sim s') \rightarrow \psi(s')$ is valid. This relation is different from the one defined for deterministic systems in that it is valid not if there is a transition from $s$ to $s'$ but more generally if there is an allowed transition (non-zero probability of transition) from $s$ to $s'$.

Our verification method is composed of three types of rules: Invariance rules (I), Stability (Lyapunov-based) rules (S) and Average Timeliness rules (AT). Assume $ATA$ is a $\forall$-automaton.
\( \langle A, T, \tau \rangle \) representing the behavioural constraints for the stochastic dynamical system: \( \langle S, \mathcal{P}, \Theta \rangle \).

(I) Invariance Rules

We define a set of propositions \( \{ \alpha_q \}_{q \in Q} \) as a set of invariants for the behaviour \( B \) and specification \( A \) iff

1. Initiality: \( \forall q \in Q, \Theta \land e(q) \rightarrow \alpha_q \), and

2. Consecution: \( \forall q, q' \in Q, \{ \alpha_q \} B \{ e(q, q') \rightarrow \alpha_{q'} \} \).

**Proposition 7.3** Let \( \{ \alpha_q \}_{q \in Q} \) be invariants for \( B \) and \( A \). If \( r \) is a run of \( A \) over a trace \( v \in B \), then \( \forall t \in T, v(t) \models \alpha_{r(t)} \).

Note that this proposition stipulates that no matter which (uncertain) transition occurs, the destination state must always satisfy the invariant condition. This is consistent with the notion of invariants, regardless of whether the dynamics of the underlying systems are deterministic or stochastic.

(S) Stability Rules

Let \( \{ \alpha_q \}_{q \in Q} \) be a set of invariants for \( B \) and \( A \) as defined above. A set of partial functions \( \{ \rho_q \}_{q \in Q} \) is called a set of Lyapunov functions for \( B \) and \( A \) iff \( \rho_q : S_B \rightarrow \mathbb{R}^+ \) satisfies the following conditions:

1. Definedness: \( \forall q \in Q, \alpha_q \rightarrow \exists w \in \mathbb{R}^+, \rho_q = w \).

2. Non-increase: \( \forall q \in S, q' \in Q, \{ \alpha_q \land \rho_q = w \} B \{ e(q, q') \rightarrow E(\rho_{q'}) \leq w \} \).

3. Decrease: \( \exists \varepsilon > 0, \forall q \in B, \exists q' \in Q, \{ \alpha_q \land \rho_q = w \} B \{ e(q, q') \rightarrow \rho_{q'} - w \leq -\varepsilon \} \).

Those three conditions are derived from [ZM96]. However, the last two have been adapted for stochastic dynamical systems. Condition (S2) requires that for each stable state \( q \in S \), the transitions from \( q \) lead on average to a state for which the value of the Lyapunov function is less than or equal to the current value. Condition (S3) is similar in that it requires that for each bad state \( q \in B \), there exists at least one allowed transition (i.e., with positive probability) leading to a state with strictly smaller Lyapunov value. This is a formal requirement that can only be satisfied if there are no absorbing bad states in the system under study, as discussed previously.
Proposition 7.4 Let \( \{\alpha_q\}_{q \in Q} \) be a set of invariants for \( B \) and \( A \). Let \( r \) be a run of \( A \) over a trace \( v \in B \). Also, let \( V_B = \{ r \mid r \text{ is a run of } A \text{ over } v \in B \} \) be the set of runs induced by \( B \). If \( \{\rho_q\}_{q \in Q} \) is a set of Lyapunov functions for \( B \) and \( A \), then

- \( \forall t \in T, \exists r^*, v^* (\rho_{r^*}(v^*(t))) \leq \rho_{r(\text{pre}(t))}(v(\text{pre}(t))), \) where \( r^* \) and \( v^* \) denote all \( r' \in V_B \) and \( v' \in B \) such that \( v(\text{pre}(t)) \sim v'(t) \) and \( c(r(\text{pre}(t)), r'(t)), \) when \( r(\text{pre}(t)) \in S; \)

- \( \exists \varepsilon > 0, \forall t \in T, \exists v' \in B, r' \in V_B, (|\rho_{r^*}(v^*(t)) - \rho_{r(\text{pre}(t))}(v(\text{pre}(t)))| \leq \varepsilon) \land (v(\text{pre}(t)) \sim v'(t)) \wedge c(r(\text{pre}(t)), r'(t))) \) when \( r(\text{pre}(t)) \in B. \)

Theorem 7.1 Let \( \{\alpha_q\}_{q \in Q} \) be a set of invariants for \( B \) and \( A \). Let \( r \) be a run of \( A \) over a trace \( v \in B \). If \( \{\rho_q\}_{q \in Q} \) is a set of Lyapunov functions for \( B \) and \( A \), then

- if \( BS \) is the set of segments of consecutive \( B \)-states and \( S \)-states in \( r \), then \( \forall q^* \in BS, q^* \) has a finite number of \( B \)-states;

The results of Proposition 7.4 need to be applied to a set of traces, and thus to a set of runs. In the context of stochastic dynamical systems, we cannot guarantee that for every trace a transition from a bad (or stable) state to any other state will yield an immediate decrease (or non-increase) in the value of the Lyapunov function; nevertheless, we can show that there is a positive probability of this happening at any time point. Hence, we know that at any time \( t \) there exists at least one trace whose transition from \( v(t) \) to \( v(t+1) \) will yield a decrease in the value of the Lyapunov function.

(AT) Average-Timeliness Rules

Let \( \mathcal{ATA} = (\mathcal{A}, T, \tau) \) be an average-timed \( \forall \)-automata. Assume, without loss of generality, that time is encoded in the stochastic state transition system. We assume that it is defined in a general sense as \( \lambda : S_B \rightarrow T; \) i.e., as a function of time measure on states returning the time until the next transition. Note that for the special case of discrete time systems on \( \mathbb{N} \), \( \lambda \equiv 1 \) uniformly. We now define two different types of timing functions, associated with the local and global average-time bounds respectively.

Once again, let \( \{\alpha_q\}_{q \in Q} \) be a set of invariants for \( B \) and \( A \). A set of partial functions \( \{\gamma_q\}_{q \in T} \) is called a set of local timing functions for \( B \) and \( \mathcal{ATA} \) if \( \gamma_q : S_B \rightarrow \mathbb{R}^+ \) satisfies the following conditions:
(L1) **Boundedness:** \( \forall q \in T, \alpha_q \to \lambda \leq \gamma_q \leq \tau(q) \).

(L2) **Decrease:** \( \forall q \in T, \{\alpha_q \land \gamma_q = w \land \mathbb{E}(\lambda) = l\} B\{c(q, q) \to \mathbb{E}(\gamma_q) - w \leq -l\} \).

A set of partial functions \( \{\eta_q\}_{q \in Q} \) is called a set of **global timing functions** for \( B \) and \( \mathcal{A}T\mathcal{A} \) iff \( \eta_q : S_B \to \mathbb{R}^+ \) satisfies the following conditions:

(G1) **Definedness:** \( \forall q \in Q, \alpha_q \to \exists w \in \mathbb{R}^+, \eta_q = w \).

(G2) **Boundedness:** \( \forall q \in B, \alpha_q \to \eta_q \leq \tau(\text{bad}) \).

(G3) **Non-increase:** \( \forall q \in S, q' \in Q, \{\alpha_q \land \eta_q = w\} B\{c(q, q') \to \mathbb{E}(\eta_{q'}) \leq w\} \).

(G4) **Decrease:** \( \forall q \in B, q' \in Q, \{\alpha_q \land \eta_q = w \land \mathbb{E}(\lambda) = l\} B\{c(q, q') \to \mathbb{E}(\eta_{q'}) - w \leq -l\} \).

**Proposition 7.5** Let \( \{\alpha_q\}_{q \in Q} \) be a set of invariants for \( B \) and \( \mathcal{A} \), and \( r \) be a run of \( \mathcal{A} \) over a trace \( \nu \in \mathcal{A} \). If there exist local timing functions, \( \{\gamma_q\}_{q \in T} \), and global timing functions, \( \{\eta_q\}_{q \in Q} \), for \( B \) and \( \mathcal{A}T\mathcal{A} \), then

1. if \( Sg(q) \) is the set of segments of consecutive \( q \)'s in \( r \), then \( \forall q \in T, q^* \in Sg(q), \mathbb{E}(\mu(q^*)) \leq \tau(q) \), and

2. if \( BS \) is the set of segments of consecutive \( B \) and \( S \)-states in \( r \), then \( \forall q^* \in BS, \mathbb{E}(\mu_B(q^*)) \leq \tau(\text{bad}) \).

The following is the set of verification rules (Invariance (I), Stability (S) and Average-Timeliness (AT)) for a behaviour \( B \) and an average-timed automaton \( \mathcal{A}T\mathcal{A} = (\mathcal{A}, T, \tau) \):

(I) **Associate with each automaton state** \( q \in Q \) a state formula \( \alpha_q \), such that \( \{\alpha_q\}_{q \in Q} \) is a set of invariants for \( B \) and \( \mathcal{A} \).

(S) **Associate with each automaton state** \( q \in Q \) a partial function \( \rho_q \), such that \( \{\rho_q\}_{q \in Q} \) is a set of Lyapunov functions for \( B \) and \( \mathcal{A} \).

(AT) **Associate with each average-timed automaton state** \( q \in T \) a partial function \( \gamma_q \), such that \( \{\gamma_q\}_{q \in T} \) is a set of local timing functions for \( B \) and \( \mathcal{A}T\mathcal{A} \). Associate with each automaton state \( q \in Q \) a partial function \( \eta_q \), such that \( \{\eta_q\}_{q \in Q} \) is a set of global timing functions for \( B \) and \( \mathcal{A}T\mathcal{A} \).
Let us now present the main result of this section. The following theorem stipulates that if we are equipped with a set of invariants, Lyapunov functions and local and global timing functions, then the behavioural verification is sound and complete.

**Theorem 7.2 (Verification Rules)** For any state-based and time-invariant behaviour $B$ with an infinite time structure and a complete average-timed $\forall$-automaton $ATA$, the verification rules are sound and complete, i.e., $B \models ATA$ iff there exist a set of invariants, Lyapunov functions and timing functions.

### 7.2.2 Automatic Behaviour Verification

The above rules do not guarantee the existence of an automatic verification method. However, for finite domain probabilistic constraint nets, we can fully automate the process in order to verify an average-timed $\forall$-automata constraint on the behaviour. We will briefly describe the algorithm and then will utilize it to verify the elevator system augmented with probabilistic passenger arrivals.

First, let us assume that $PCN = (Lc, Td, Cn)$ is a probabilistic constraint net made solely of transliterations and unit delays. We denote an acceptable state by $PCN(s)$ iff for every equation of the form $l_0 = f(l_1, \cdots, l_n) \rightarrow Pr(s(l_0) = f(s(l_1), \cdots, s(l_n))) > 0$, and denote an acceptable transition by $PCN(s, s')$ if and only if $PCN(s)$ and $PCN(s')$, and if for every delay equation $l'_0 = l$, $s'(l_0) = s(l)$. Let us also denote a reachable pair $(q, s)$ by $r(q, s)$ where $q \in Q$ and $s \in \times LcA_{kl}$. Furthermore, let $K$ be the evolution kernel associated with the set of reachable pairs for $q \in B \cup S$ and let $T$ be the matrix summarizing the time for each transition within the set of reachable pairs with $q \in B \cup S$. In addition, let $K_1$ represent the evolution kernel for the set of reachable states with $q \in B \cup S$ which constitute the $RS$-boundary of the set $B \cup S$. This set is composed of all the bad and stable states which have a direct transition to a state $r \in R$ or to an absorbing $S$ state. Proceed similarly to define $T_1$. Finally, let $L$ and $L_1$ be the matrix $T$ and $T_1$ respectively where the non-zero entries have been replaced by the value 1.

The algorithm follows the verification rules and has four steps which we describe below:

1. **Invariant Generation:** We can show that invariants can be constructed by finding the fixpoint of the sets of Equation A.4. This fixpoint can be obtained with the following two steps:

   (a) **Initiality:** Generate $r(q, s)$ if $\Theta(s), e(q)(s), PCN(s)$. 

131
(b) **Consecution:** Generate $r(q', s')$ if $r(q, s), PCN(s, s'), c(q, q')(s').$

2. **Non-Absorbness and Stability:**

   • Verify that the set of bad states is irreducible. That is, ensure that for every bad state $b \in B$ there is a path with non-zero measure leading to a $R$-state or an $S$-state. If it is not the case, proceed to the modification of the state space discussed earlier such that absorbing bad states are removed.
   
   • For $q \in R$, let $\rho_q = 0$.
   
   • Solve the set of linear equations for the average number of transitions taken to enter the set of recurrent states or absorbing $S$-states. The solution is the set of Lyapunov functions, $\{\rho_q\}_{q \in B \cup S}$. Practically, to solve for $\{\rho_q\}_{q \in B \cup S}$, define $A = -K + I_n$ and $u = diag([T, T_1] * [K, K_1])$, then solve $A\rho = u$. Here $diag$ denotes the diagonal operator, which returns the diagonal elements of a matrix.

3. **Global Average Timing:**

   • For $q \in R$, let $\eta_q = 0$.
   
   • Similarly to the method for stability, solve the set of linear equations for the average time measure to leave the set of bad and non-absorbing stable states, not accounting for time spent in an $S$-state. The solution is the set of Global timing functions $\{\eta_q\}_{q \in B \cup S}$. Verify that $\eta_q < \tau(bad), \forall q \in Q$.

4. **Local Average Timing:**

   • For each $q \in T$, solve the set of linear equations for the average time measure to leave $q$. This is similar to solving for the Lyapunov functions where we only consider states $q \in T$. This leads to the local timing functions $\{\gamma_q\}_{q \in T}$. Verify that $\gamma_q < \tau(q), \forall q \in T$.

   It is possible, given this method, to obtain a bound on the probability that a certain time bound will be exceeded. The bound is obtained from the well known equation $Pr(X > \tau) \leq \mathbb{E}(X^2)/\tau^2$. We can calculate $\mathbb{E}(X^2)$, where $X$ is the average time to reach a $R$-state or an absorbing $S$-state, as obtained in the global and local average timeliness rules. With the equation

$$u = 2[T, T_1] * [K, K_1] * \eta + diag([T, T_1] * [T, T_1] * [P, P_1]',$$
where the operator $\ast$ denotes the element-wise matrix multiplication, solve $AY = u$ for $Y$ to calculate the value of the probability bound.

### 7.3 Generalizing the Approach

In this section, we generalize the concept of average-timed $\forall$-automata on discrete time structures to arbitrary time structures. This allows us to apply our method to behavioural constraint verification of stochastic hybrid dynamical systems, which generate traces on general time structures. Note that the common time structures of continuous and discrete time both act as special cases.

Essentially, the set of verification rules for general time structures follows closely that of discrete time systems, however, the definitions of invariants, Lyapunov functions and timing functions are generalized.

For any trace $v : T \rightarrow A$, let $\{\varphi\}_v \{\psi\}$ denote the validity of the following two consecutive conditions:

- $\{\varphi\}_v \{\psi\}$: for all $t > 0$, $\exists t' \leq t$, $\forall t'', t' \leq t'' < t$, $v(t'') \models \varphi$ implies $v(t) \models \psi$;

- $\{\varphi\}_v^+ \{\psi\}$: for all $t < \infty$, $v(t) \models \varphi$ implies $\exists t' > t$, $\forall t'', t < t'' < t'$, $v(t'') \models \psi$.

If $T$ is discrete, these two conditions are reduced to one, i.e., $\forall t > 0, v(\text{pre}(t)) \models \varphi$ implies $v(t) \models \psi$.

Given $B$ as a behaviour, let $\Theta = \{v(0) | v \in B\}$ denote the set of initial values in $B$. Let $A = (Q, R, S, e, c)$ be a $\forall$-automaton. A set of propositions $\{\alpha_q\}_{q \in Q}$ is called a set of invariants for $B$ and $A$ if:

- **Initiality**: $\forall q \in Q, \Theta \land e(q) \rightarrow \alpha_q$.

- **Consecution**: $\forall v \in B, \forall q, q' \in Q, \{\alpha_q\}_v \{c(q, q') \rightarrow \alpha_{q'}\}$.

**Proposition 7.6** Let $\{\alpha_q\}_{q \in Q}$ be invariants for $B$ and $A$. If $r$ is a run of $A$ over $v \in B$, $\forall t \in T, v(t) \models \alpha_{r(t)}$.

Without loss of generality, we assume that time is encoded in domain $A$ by $\lambda : A \rightarrow T$. Given that $\{\alpha_q\}_{q \in Q}$ is a set of invariants for $B$ and $A$, a set of partial functions $\{\rho_q\}_{q \in Q} : A \rightarrow \mathbb{R}^+$ is called a set of Lyapunov functions for $B$ and $A$ if the following conditions are satisfied:
• Definedness: $\forall q \in Q, \alpha_q \rightarrow \exists w \in \mathbb{R}^+, \rho_q = w$.

• Non-increase: $\forall v \in B, \forall q \in S, q' \in Q,$

$$\{\alpha_q \land \rho_q = w\}v^-\{c(q, q') \rightarrow E(\rho_{q'}) \leq w\}$$

and $\forall q \in Q, q' \in S,$

$$\{\alpha_q \land \rho_q = w\}v^+\{c(q, q') \rightarrow E(\rho_{q'}) \leq w\}.$$

• Decrease: $\forall v \in B, \exists \epsilon > 0, \forall q \in B, q' \in Q,$

$$\{\alpha_q \land \rho_q = w \land E(\lambda) = t\}v^-\{c(q, q') \rightarrow \frac{E(\rho_{q'}) - w}{\mu([0, E(\lambda))]) \leq -\epsilon\}$$

and $\forall q \in Q, q' \in B,$

$$\{\alpha_q \land \rho_q = w \land E(\lambda) = t\}v^+\{c(q, q') \rightarrow \frac{E(\rho_{q'}) - w}{\mu([0, E(\lambda))])} \leq -\epsilon\}.$$

Proposition 7.7 Let $\{\alpha_q\}_{q \in Q}$ be invariants for $B$ and $A$ and $r$ be a run of $A$ over a trace $v \in B$. If $\{\rho_q\}_{q \in Q}$ is a set of Lyapunov functions for $B$ and $A$, then

• $E(\rho_r(t_2)(v(t_2))) \leq \rho_r(t_1)(v(t_1))$ when $\forall t_1 \leq t \leq t_2, r(t) \in B \cup S$,

• $\frac{E(\rho_r(t_2)(v(t_2))) - \rho_r(t_1)(v(t_1))}{\mu([t_1, t_2])} \leq -\epsilon$ when $t_1 < t_2$ and $\forall t_1 \leq t \leq t_2, r(t) \in B$, and

• if $BS$ is the set of segments of consecutive $B$ and $S$-states in $r$, then $\forall q^* \in BS, \mu_B(q^*)$ is finite.

Let $ATA = \langle A, T, \tau \rangle$. Corresponding to the two types of time bounds, we define two timing functions. Let $\{\alpha_q\}_{q \in Q}$ be invariants for $B$ and $A$. A set of partial functions $\{\gamma_q\}_{q \in T}$ is called a set of local timing functions for $B$ and $ATA$ iff $\gamma_q : A \rightarrow \mathbb{R}^+$ satisfies the following conditions:

• Boundedness: $\forall v \in B, \forall q \in Q, q' \in T,$

$$\{\alpha_q\}v^-\{c(q, q') \rightarrow \gamma_{q'} \leq \tau(q')\}$$

and $\forall q \in T, q' \in Q,$

$$\{\alpha_q \land E(\lambda) = t \land \gamma_q = w\}v^-\{c(q, q') \rightarrow w \geq \mu([0, E(\lambda))])\}.$$
A set of partial functions \( \{\eta_q\}_{q \in Q} \) is called a set of global timing functions for \( B \) and \( AT.A \) iff \( \eta_q : A \to \mathbb{R}^+ \) satisfies the following conditions:

- **Definedness**: \( \forall q \in Q, \alpha_q \to \exists w \in \mathbb{R}^+, \eta_q = w. \)
- **Boundedness**: \( \forall q \in B, \alpha_q \to \mathbb{E}(\eta_q) \leq \tau(bad). \)
- **Non-increase**: \( \forall v \in B, \forall q \in S, q' \in Q, \)
  \[ \{\alpha_q \land \eta_q = w\} v^- \{c(q, q') \rightarrow \mathbb{E}(\eta_{q'}) \leq w\} \]
  and \( \forall q \in Q, q' \in S, \)
  \[ \{\alpha_q \land \eta_q = w\} v^+ \{c(q, q') \rightarrow \mathbb{E}(\eta_{q'}) \leq w\}. \]
- **Decrease**: \( \forall v \in B, \forall q \in B, q' \in Q, \)
  \[ \{\alpha_q \land \eta_q = w \land \mathbb{E}(\lambda) = t\} v^- \{c(q, q') \rightarrow \frac{E(\eta_{q'}) - w}{\mu([0, \mathbb{E}(\lambda)])} \leq -1\} \]
  and \( \forall q \in Q, q' \in B, \)
  \[ \{\alpha_q \land \eta_q = w \land \mathbb{E}(\lambda) = t\} v^+ \{c(q, q') \rightarrow \frac{E(\eta_{q'}) - w}{\mu([0, \mathbb{E}(\lambda)])} \leq -1\}. \]

**Proposition 7.8** Let \( \{\alpha_q\}_{q \in Q} \) be invariants for \( B \) and \( A \) and \( \tau \) be a run of \( A \) over a trace \( v \in B \). If there exist local and global timing functions for \( B \) and \( AT.A \), then

- if \( Sg(q) \) is the set of segments of consecutive \( q \)'s in \( \tau \), then \( \forall q \in T, q^* \in Sg(q), \mathbb{E}(\mu(q^*)) \leq \tau(q) \), and
- if \( BS \) is the set of segments of consecutive \( B \) and \( S \)-states in \( \tau \), then \( \forall q^* \in BS, \mathbb{E}(\mu_B(q^*)) \leq \tau(bad) \).

The following theorem is a generalization of the soundness and completeness of the set of verification rules.

**Theorem 7.3** The verification rules (I), (S) and (AT) are sound if the following conditions on \( B \) and \( AT.A \) are satisfied:
• $T$ is an infinite time structure.

• All traces in $B$ are specifiable by $ATA$.

The verification rules are complete if the following conditions on $B$ and $ATA$ are satisfied:

• $\{(v, r)|v \in B, r \text{ is a run over } v\}$ is time-invariant.

• All transitions from $R$ to non-$R$-states are left-closed, i.e., if $r$ is a run, and there is a transition from a $R$-state to a $B$-state or a $S$-state at $t$, then $r(t) \in B \cup S$.

The conditions for the completeness of the rules are imposed so as to be able to define Lyapunov functions for a behaviour and an automaton, as long as the behaviour satisfies the automaton. The second condition for completeness is always satisfied for traces with discrete time structures. More generally, the following proposition may apply.

**Proposition 7.9** All transitions from $R$ to non-$R$-states are left-closed, if the following conditions are satisfied:

• $ATA$ is open and complete.

• $\forall q \in R, q_1 \not\in R \text{ and } q_2 \in R, c(q, q_1) \land c(q, q_2) \text{ is not satisfiable.}$

• All traces in $B$ are right-continuous.

These definitions are essential to provide understand of general behaviours of stochastic hybrid dynamical systems. At the present time, however, we have yet to develop an algorithm, either semi-automatic or automatic, based on these rules. Work in progress includes the development of such algorithms along with the augmentation of the behavioural constraint verification technique to perform quantitative probabilistic verification.
Chapter 8

Behavioural Verification with a Probabilistic Temporal Logic

Probabilistic logics provide a simple yet powerful specification methodology for (dynamical) systems with temporal behaviour. We develop the Probabilistic Arbitrary-time Timed Temporal Logic (PATTL) for specifying the specifications on behaviours of uncertain systems, where timed refers to the notion of temporal evolution of the systems viewed as a metric distance. First we generalize the probabilistic computational tree logic (PCTL) [HJ94] into the Probabilistic Arbitrary-time Temporal Logic (PATL) to incorporate both discrete and continuous time, so that properties of arbitrary traces can be specified and reasoned about. Then we augment the modal operator so that real-time properties can be specified. Finally, we develop a first order PATTL (FPATTL) for arbitrary time and domain structures.

The PATTL logic is intended to specify timed quantitative probabilistic behavioural constraints applied to a PCN model of a system. For example, in an elevator system, the building manager might want to reason about the overall service level of his elevator. He might be interested in maintaining a basic service level at all times where the incoming requests are served within 40 time units but might agree, as a cost saving strategy, to accept that the probability of the service level dropping below the basic level for at most a specified short period of time be less than 10%. This requirement is often referred to as quality of service and is a very important requirement for many applications. Therefore, it is essential to be equipped with a specification language that is powerful enough to allow us to explicitly state such requirements but also offers computational efficiency so that one
can verify that the desired requirements for real-world practical applications are indeed satisfied.

Our PATTL logic arises from the well known logic called Computational Tree Logic (CTL) [Eme90, EC82], a branching time logic, which some of the most important probabilistic logics extend. While CTL is not expressive enough to specify quantitative properties of uncertainty, it has a natural correspondence with the computation of concurrent and non-deterministic programs. The fact that CTL is a branching time logic rather than a linear time logic makes it a good candidate to serve as the basis of our probabilistic temporal logic. Indeed, the underlying structure of time being tree-like provides an ideal framework as one can view each branch as a different probabilistic (or non-deterministic) successor of the current node. In our probabilistic approach to systems, we view the branching property of such logic as a representation of each possible event \( \omega \) taken from the event space \( \Omega \). We will elaborate on this notion later. However, for a more in depth (and classic) comparison of linear time logic and branching time logic, the reader is referred to [Eme90].

Recently, with the increasing number of practical applications where the inherent uncertainty is modeled, the need to specify quantitative, rather than qualitative, probabilistic properties has lead to the development of full probabilistic logics, most of them directly extending the CTL framework. PATTL is such an extension and is related to the Probabilistic Computational Tree Logic (PCTL) of [HJ94]: it generalizes PCTL into a logic for arbitrary time and domain structures which allows formulae to be interpreted over a general probabilistic constraint net model.

8.1 Introduction to CTL and PCTL

As mentioned above, our logic emerges from one of the simplest branching time logics: Computational Tree Logic (CTL)\(^1\) first introduced by [EC82]. It introduces formulae where one of the two path quantifiers, A (for all future paths) and E (for some future paths), are followed by the usual linear temporal operators that can be found in most temporal logics: \( X \) (next time), \( U \) (until), \( \square \) (always) and \( \diamond \) ( sometime or eventually).

CTL was developed as a modal (temporal) logic for reasoning about qualitative program correctness. Typical sentences in CTL expressing properties are: "\( p_1 \) will hold continuously on some future execution path" (\( E \boxdot p_1 \)), "\( p_2 \) will eventually hold on all future executions paths" (\( A \diamond p_2 \)),

\(^1\)CTL can also be extended to a more expressive logic CTL* which allow the path operators to be followed by a boolean combination or nesting over formulae made with F, G, X or U. Here we will restrict ourselves to CTL. For more details on the extended logic CTL*, the reader is referred to [Eme90].
and "p₃ will always hold on all future execution paths" (A □ p₃). Emerson et al. [EMSS92] have extended CTL with RTCTL which is suited for specification of hard deadlines. Alur extended CTL in a way similar to RTCTL, but in their logic (TCTL) formulae are interpreted over models with continuous time [ACD90].

CTL formulae are generally interpreted over a structure $M = \langle S, R, L \rangle$ where $S$ is a set of states, $R$ is a binary relation on $S \times S$ and $L : S \rightarrow 2^\mathcal{A}$ is a labeling function which associates with every state in $S$ a subset of atomic propositions $\mathcal{A}$ which are known to be true in that state. Intuitively $M$ can be seen as a directed labeled graph with $S$ representing the set of nodes, $R$ being the arc set representing transitions and $L$ labeling every node of the graph. It is important to note that CTL formulae can also be interpreted on more general structures, a property that we will use when developing the logic to represent properties of a PCN model.

The PCTL logic [HJ94] is based on Emerson, Clarke and Sistla's Computation Tree Logic (CTL) [CES86], an extension of the above mentioned CTL logic ([EC82]) with the added property of fairness included into the model. However, unlike with CTL, in PCTL one is interested in soft deadlines such as: after a request for service is received, there is at least a 95% probability that the service will be carried out within 5 seconds. Note that using probabilities of 0 and 1 makes PCTL also suitable for expressing hard deadlines.

Within PCTL, time is assumed to be discrete, with one time unit corresponding to one transition along an execution path. To allow for reasoning about soft deadlines, the authors have replaced path quantifiers by probabilities. Examples of PCTL properties are: with at least 35% probability, property $p_1$ will hold within 10 time units ($\diamond \leq 10_{0.35}$) and with at least 95% probability, property $p_2$ will hold continuously for at most 20 time units ($\Box \leq 20_{0.95}$). PCTL uses the same operators as CTL with the exception that the operators are augmented with probabilities and time intervals. PCTL properties are interpreted over structures that are discrete time Markov Chains.

Figure 8.1 summarizes the relationship between the aforementioned logics. All these logics extend CTL to handle either hard or soft deadlines, hence quantitative properties. The arrow from PCTL to PATTL highlights the relationship between the logic introduce in this dissertation and its ancestor logic PCTL.
8.2 Probabilistic Arbitrary-time Temporal Logic (PATL)

As mentioned above, we are interested in developing a temporal probabilistic logic that will be used in the specification of behavioural constraints for systems modeled as Probabilistic Constraint Nets. Since one of the PCN framework's main advantages is that one can consider abstract time and domain structures, we would like our logic to assert semantics appropriate for such abstraction while also being powerful enough to specify temporal and probabilistic constraints. As PCTL allows for temporal and probabilistic reasoning, it is a perfect candidate to base our logic on. Moreover, a powerful verification method for PCTL behavioural constraints exists and this method can be extended to operate on a certain class of PCN structures. The algorithm will be described in the following sections.

First of all, let us start by assuming that we are equipped with a finite set of atomic propositions $\mathcal{A} = \{\alpha_1, \cdots, \alpha_n\}$ which will represent atomic properties of states. Such atomic properties could be, in an elevator system for example, $\alpha : \text{the number of passengers in the elevator at a given time does not exceed the maximum number allowed.}$

Formulae in our logic are built from a combination of atomic propositions with the propositional, temporal and probabilistic operators introduced below.

**Definition 8.1 (PATL Syntax)** Let $\mathcal{A}$ be a set of atomic propositions, $p \in [0,1]$ be a real number
known as a probability bound (or threshold) and $\odot \in \{\geq, >, <, \leq\}$ be a comparative operator defined on the reals. The syntax of PATL, given $\mathcal{A}$, is obtained inductively via the following rules:

- true is a state formula;
- each atomic proposition $\alpha \in \mathcal{A}$ is a state formula;
- if $\varphi_1$ and $\varphi_2$ are state formulae, then so are $\neg \varphi_1$ and $\varphi_1 \land \varphi_2$;
- if $\varphi_1$ and $\varphi_2$ are state formulae, then $\varphi_1 S \varphi_2$, $\varphi_1 U \varphi_2$ and $X \varphi$ are path formulae, and,
- if $\varphi$ is a path formula, then $\mathcal{P}_\odot(\varphi)$ is a state formula.

Here $U$, $S$ and $X$ denote the until, since and next temporal operators, while $\mathcal{P}$ denotes the probabilistic operator, with $\mathcal{P}_\odot(\varphi)$ asserting that the probability measure of the paths that satisfy the path formula $\varphi$ satisfies the bound $\odot p$.

In this definition, the logic operators $\neg$ and $\land$ hold the usual meaning of negation and conjunction. The probabilistic formula $\mathcal{P}_\odot(\varphi)$ specifies that the probability of the path formula $\varphi$ being true satisfies the bound $\odot p$. As we will see later in this section, this probability is obtained by computing the measure of the paths over which $\varphi$ is satisfied. The reader should note that the operator $\mathcal{P}$ replaces the common CTL quantifiers $A$ and $E$. Instead of reasoning on the qualitative satisfaction of a given formula on one, some or all computation paths, we adopt a quantitative approach which allows us to infer about the probability of a formula being satisfied. Note that this quantitative approach does not preclude us from specifying qualitative properties of systems. We keep the ability to represent "there exists" and "for all" by using specific values for $p$ within $\mathcal{P}_\odot p$. The CTL requirement $(E \varphi)$ which means "there exists a path on which $\varphi$ is satisfied" can be expressed with $\mathcal{P}_{\geq 0}(\varphi)$ while the CTL requirement $(A \varphi)$, which means "$\varphi$ is satisfied for all possible paths (except, perhaps, for a zero measure set of paths)", can be represented by $\mathcal{P}_{\geq 1}(\varphi)$, given a fair interpretation of the CTL formula $(A \varphi)$ [EL87]. In a fair interpretation of CTL, one only considers the set of paths that satisfy a certain fairness constraint (e.g., visit every state infinitely often). Hence, the satisfaction of the formulae is only with respect to those fair paths. For an elaborate discussion on the relation between fairness and probability, the reader is referred to [BK98a].
It is important to note that in the presence of uncertainty, the meaning of trace differs significantly from its usual deterministic meaning. Indeed, when dealing with systems behaving unpredictably, even if all the parameters of the system are fixed and no inputs are hidden, a trace does not represent the exact behaviour of the system but rather one of the possible behaviours. It is for this reason that instead of talking about the behaviour of a system, one often refers to its average behaviour. Therefore, to be able to assess the behaviour of a given system, one needs to observe many traces so that an average behaviour can emerge.

Our goal in defining an interpretation function is to be able to relate every state (value in the domain $A$) to the set of atomic propositions that is satisfied in that state. We say that a pair $(v(t), \varphi)$ is an element of the satisfaction relation $\models$ if and only if the state formula $\varphi$ is valid in state $v(t)$. We denote the satisfaction of $\varphi$ in state $v(t)$ by $v(t) \models \varphi$. To introduce the semantics of a PATL path formula $\psi$ augmented with the probabilistic operator $\mathcal{P}$, we need to define, for each PCN structure $\mathcal{K}$ and state $v(t)$, a probability measure $\mu_{v(t)}^{\mathcal{K}}$ on the set of traces starting from $v(t)$.

Definition 8.2 (Probability distribution induced by a PCN) Assume the following PCN structure $\mathcal{K} = \langle L, T, C \rangle$ with $n$ locations, $L = \{L_i\}_{i=1}^{n}$, each with domain $A_{L_i}$. Let us denote the set of probabilistic transductions by $P \subseteq T$. The set of probabilistic transductions$^2$ induces a probability distribution $\mathbb{P}^{\mathcal{K}} : (\times_1 A_i) \times (\times_1 A_i) \rightarrow [0,1]$ which is a well-defined probability distribution due to the well-definedness of each probabilistic transduction in $T$. Hence, $\mathbb{P}^{\mathcal{K}}(a_t, a_{t+1}) = P_{L_1}(a_{t+1} | L_1(t_1)) \times \cdots \times P_{L_n}(a_{t+1} | L_n(t_1))$ denotes the probability of the cross-product $\times_1 L_i$ having values $a_{t+1}$ at time $t+1$ while having values $a_t$ at time $t$, where $I_{L_i}$ denotes the input locations of transduction $P_{L_i}$.

Based on this definition, let us consider the well-definedness of a PCN trace.

Definition 8.3 (Well-definedness of a trace) Let $v : T \rightarrow \times_1 A_i$ be a trace and let $\mathcal{K} = \langle L, T, P, C \rangle$ be a Probabilistic Constraint Net structure with $n$ locations and $P \subseteq T$ as defined above. We say that a trace $v$ is well-defined, given $\mathcal{K}$, if $v(t)$ is well-defined for all $t \in T$ (i.e., $v(t) \in \times_1 A_i$, $\forall t \in T$), and for every consecutive time point $t_i, t_{i+1} \in T$ we have $\mathbb{P}^{\mathcal{K}}(v(t_i), v(t_{i+1})) > 0$.

$^2$The set of deterministic transductions also comes into play but each deterministic transduction induces a probability distribution with 0 and 1 values only, hence not affecting directly the result.
Example 8.1 (Undefined trace) Consider the PCN $\mathcal{X} = \langle \{x\}, \{\delta_x\}, P(x_t, x_{t+1}), C \rangle$, which is defined on $T = \mathbb{N}$ and $A = \{1, 2\}$ with sole deterministic transduction being the unit delay $\delta_x$ and unique probabilistic transduction defined as: $P(1, 1) = 0.4$, $P(1, 2) = 0.6$, $P(2, 1) = 1$ and $P(2, 2) = 0$. For this simple PCN model, we have $\mathbb{P}_\mathcal{X} = P$. Now consider the partial trace $v = \{1, 2, 2, 1, 1, 2, 1\}$. Although this trace is defined for every value it takes over $t = \{1, \ldots, 7\}$, the transition from $v(2)$ to $v(3)$ is ill-defined with respect to the probability distribution $\mathbb{P}_\mathcal{X}$. Indeed, transitions from domain value 2 to 2 has probability 0 in the model, and hence is not a valid transition. In fact, it is easy to show that such a trace has measure 0. □

Definition 8.4 (Probability measure of trace) Let $B^\mathcal{X}_a$ denote the set of traces of the PCN $\mathcal{X} = \langle L, T, P, C \rangle$ starting at $a \in \times_t A_t$, i.e., $v(0) = a$. In accordance with measure theory, we define:

- For any sequence $v_0, v_1, \ldots, v_n$, with $v_i \in \times_t A_t$,
  $$\mu^\mathcal{X} v_0 \{v \in B^\mathcal{X}_a \mid v|_n = \{v_0, \ldots, v_n\}\} = \mathbb{P}_\mathcal{X}(v_0, v_1) \times \cdots \times \mathbb{P}_\mathcal{X}(v_{n-1}, v_n)$$
  where $v|_n$ denotes the restriction of $v$ onto $\{t' \in T \mid 0 \leq t' \leq n\}$. That is, the measure of a cylinder set of all traces $v \in B^\mathcal{X}_a$ such that $v[i] = v_i$, $(i \leq n)$ is equal to the product $\prod_{i=0}^{n} \mathbb{P}_\mathcal{X}(v_i, v_{i+1})$.

- For $n = 0$
  $$\mu^\mathcal{X} v_0 \{v \in B^\mathcal{X}_a \mid v|_0 = \{v_0\}\} = 1.$$

- For any countable set $\{S_i\}_{i \in I}$ of disjoint subsets of $B^\mathcal{X}_v$
  $$\mu^\mathcal{X} v_0 (\bigcup_{i \in I} S_i) = \sum_{i \in I} \mu^\mathcal{X} v_0 (S_i)$$
  The sum is well-defined. This follows because it is bounded by 1 and each term in the summation is non-negative.

- The measure of the complement set $B^\mathcal{X}_v \setminus S$ where $S$ is a subset of $B^\mathcal{X}_v$ can be obtained via
As stated in the preceding section on the relationship between uncertainty and time, we will assume non-Zenoness of the probabilistic transductions. However, given a PCN model with continuous time and discrete domain structures, where the transport delays are of positive duration and the probabilistic transductions are event-based on exponential distributions, we can show that this assumption is superfluous. Indeed, the following proposition shows that the set of infinite traces where all the transitions are taken in a finite amount of time has probability measure 0. The proof of this result is similar in essence to the proof of Proposition 1 of [BHHK03]. This result can also be obtained from a not so simple application of Theorem 3 of [dA97a].

### Proposition 8.1 (Measure of non-time-divergent infinite traces)
Assume a well-defined continuous time $T$ and discrete domain $\times_1 A_i$ PCN model (with positive transport delays to avoid algebraic loops) where all probabilistic transductions $P_i$ are event-based with exponential distribution $\exp(\lambda_i)$. For any initial value $v_0 \in \times_1 A_i$, the probability measure of the set of infinite traces $B : v_0, v_1, v_2, \ldots$ for which time is convergent, i.e., $\sum_{i \geq 0} t_i < \infty$ where $t_i$ denote the time of events $i \geq 0$, converges to 0.

Now let us present the definition of the structures over which PATL formulae will be interpreted. As stated above, we are interested in specifying the behaviour of probabilistic dynamical systems. In order to do so, we introduce a few notions that will be useful for the rest of this section.

### Definition 8.5 (Frame)
A frame of PATL over a PCN structure $\mathcal{X}$ is a tuple $(T, A, \mathbb{P}^{\mathcal{X}}, V)$ where $T$ denotes a time structure, $A$ is a composite domain, $\mathbb{P}^{\mathcal{X}}$ is the probability distribution induced by $\mathcal{X}$ and $V$ is referred to as an interpretation$^4$. Formally, we define an interpretation as a function $V : A \rightarrow 2^{\mathcal{X}}$. We will use $v \models \varphi$ to denote $\varphi \in V(v)$.

### Definition 8.6 (Model)
A model of PATL is a tuple $(F, v)$ where $F = (T, A, \mathbb{P}^{\mathcal{X}}, V)$ is a frame and $v : T \rightarrow A$ is a trace generated by the PCN $\mathcal{X}$.

Formally, the semantics of the PATL logic is defined as follows.

---

$^3$A system is called Zeno if it takes infinitely many discrete transitions in a finite time interval.

$^4$In the temporal logic and model checking literature, $V$ is often referred to as a labeling function.
Definition 8.7 (Semantics of PATL) Let $\mathcal{F} = (\mathcal{T}, A, \mathbb{P}, V)$ be a frame and $(\mathcal{F}, v)$ be a model of PATL. Let $\varphi$ and $\psi$ be PATL state and path formulae respectively. $v \models_t \varphi$ denotes that state $v$ satisfies $\varphi$ at time $t$. Specifically the satisfaction of PATL formulae is defined as follows.

$$
\begin{align*}
&v \models_t \text{true}, \quad \forall t \in \mathcal{T}. \\
&v \models_t \alpha \quad \text{iff } [v(t) \models \alpha] \equiv [\alpha \in V(v(t))] \text{ with } \alpha \in \mathcal{A}.
&v \models_t \neg \varphi \quad \text{iff } v(t) \not\models \varphi.
&v \models_t \varphi_1 \land \varphi_2 \quad \text{iff } v(t) \models \varphi_1 \land v(t) \models \varphi_2.
&v \models_t \Box \varphi \quad \text{iff } \exists t', \forall t'', t < t'' \leq t', v \models_{t''} \varphi.
&v \models_t \varphi_1 \supset \varphi_2 \quad \text{iff } \exists t' \leq t, \forall t'' \leq t, \exists t'' < t', v \models_{t''} \varphi_2 \text{ and } \forall t''', t < t'' < t', v \models_{t'''} \varphi_1.
&v \models_t \varphi_1 \mathcal{U} \varphi_2 \quad \text{iff } \exists t' \geq t, v \models_{t'} \varphi_2 \text{ and } \forall t'', t \leq t'' < t', v \models_{t''} \varphi_1.
&v \models_t \varphi \mathcal{O}_{\varphi}(\psi) \quad \text{iff } \mu_{v(t)}(\{v|v(0) = v(t) \land v \models_t \psi\}) \in p.
\end{align*}
$$

To specify the initial satisfaction of a PATL formula $\varphi$, i.e., $v \models_0 \varphi$, we will use the simplified expression $v \models \varphi$. A PATL formula $\varphi$, not extended with the probabilistic operator, is said to be valid over a frame $\mathcal{F}$, iff for any model $(\mathcal{F}, v)$, $v \models \varphi$. $\varphi$ is valid, iff for any frame $\mathcal{F}$, $\varphi$ is valid over $\mathcal{F}$. $\varphi$ is satisfiable over a frame $\mathcal{F}$, iff for some model $(\mathcal{F}, v)$, $v \models \varphi$. Finally, we say that $\varphi$ is satisfiable, iff for some frame $\mathcal{F}$, $\varphi$ is satisfiable over $\mathcal{F}$.

8.2.1 Common Logical Connectives and Temporal Operators

With our syntax of PATL we are able to represent many other commonly used logical connectives and temporal operators. Here are a few of them represented within the PATL syntax.

- False: $\text{false} = \neg \text{true}$.
- Disjunction: $\varphi_1 \lor \varphi_2 = \neg (\neg \varphi_1 \land \neg \varphi_2)$.
- Implication: $\varphi_1 \to \varphi_2 = \neg \varphi_1 \lor \varphi_2$.
- Equivalence: $\varphi_1 \leftrightarrow \varphi_2 = (\varphi_1 \to \varphi_2) \land (\varphi_2 \to \varphi_1)$.
- Eventually: $\Diamond \varphi \equiv \text{true U} \varphi$.
- Previous: $\Theta \varphi \equiv \varphi \mathcal{S} \varphi$.

The propositional connectives $\lor, \to, \leftrightarrow$, have the usual logical meaning. The reader is referred to [Eme90] for the definitions of stronger and weaker variants of these temporal operators.

The commonly used Always operator ($\Box \varphi$), which is equivalent to $\neg \Diamond \neg \varphi$ in the CTL* logic,
cannot be derived similarly in our logic. However, we can derive it using the duality of lower and upper probability bounds, e.g., $P_{> \pi}(\square \varphi) = P_{\leq 1-\pi}(\Diamond \neg \varphi)$.

Based on the definitions above, we can present the semantics of these commonly used connective and temporal operators. Given a frame $\mathcal{F} = \langle T, A, P^X, V \rangle$, a model $\langle \mathcal{F}, v \rangle$ and a PATL formula $\varphi$, the semantics are specified as follow.

- **False:** $v \not \models_t false$.
- **Disjunction:** $v \models_t \varphi_1 \lor \varphi_2$ iff $v(t) \models \varphi_1 \lor v(t) \models \varphi_2$.
- **Implication:** $v \models_t \varphi_1 \rightarrow \varphi_2$ iff $v \models_t \varphi_1$, implies $v \models_t \varphi_2$.
- **Eventually:** $v \models_t \Diamond \varphi$ iff $\exists t' \geq t, v \models v(t) \models \varphi$.
- **Previous:** $v \models_t \Box \varphi$ iff $\exists t'' < t, \forall t' \leq t'' < t, v \models v(t) \models \varphi$.
- **Always:** $v \models_t \Box \varphi$ iff $\forall t' \geq t, v \models v(t) \models \varphi$.

For the elevator task mentioned earlier and which is described in detail in Appendix B, let $E$ be the proposition denoting the entrance of a passenger in the elevator and let $D$ representing the proposition which is true when the passenger is delivered to the floor of his selection. A desired property for the elevator is $\square (E \rightarrow P_{\geq 1}(true \cup D))$. That is, whenever a passenger enters the elevator (event satisfying $E$), then with probability 1 the passenger will be delivered to his destination floor ($D$).

Other important properties of behaviours mentioned earlier can also be specified using PATL.

- **Safety:** Let $B$ denote a proposition representing an undesirable (bad) situation: $\square \neg B$.
- **Goal achievement:** Let $G$ be a proposition representing a final absorbing goal of a system: $\Diamond \Box G$.
- **Recurrence:** Let $R$ be a proposition of a recurrent condition: $\Box \Diamond R$.

### 8.3 Probabilistic Arbitrary-time Timed Temporal Logic (PATTL)

In the preceding section we have introduced a logic that can represent probabilistic properties over general traces. However, in this logic we did not take time explicitly into account. Many important behavioural requirements of dynamical systems contain a temporal component. For example, the
elevator property $\Box(E \rightarrow \mathcal{P}_{\geq 1}(\text{true}UD))$ would be more meaningful if one could stipulate an upper bound on the time of delivery.

In this section, we develop the PCN timed temporal logic (PATTL), an extension of PATL which allows to specify the metric properties of time. The basic syntax and semantics of PATTL are the same as those of PATL with the exception that we augment PATL with two real-time operators: timed until, $U^\tau$ and timed since, $S^\tau$, where $\tau \geq 0$ is a nonnegative real-number.

Before defining the semantics of these operators, we need to introduce the concept of time evolution. Even though PATTL is based on a branching time logic (CTL), we still maintain our notion of linear time. As mentioned earlier, we see the branching property of the logic as representing the probabilistic choices within the system’s dynamics rather than representing the evolution of time. With this notion of linear time, we need to define the notion of subsets of the time domain. The reason for this definition will become evident once we define the semantics for the real-time logical operators $U^\tau$ and $S^\tau$.

**Definition 8.8 (Time Domain Subset)** Let $\tau \geq 0$ be a nonnegative real number representing a time duration. Let $\mathcal{T}_{t+\tau} = \{t' | t \leq t', d(t, t') \leq \tau\}$ and $\mathcal{T}_{t-\tau} = \{t' | t' \leq t, d(t', t) \leq \tau\}$, where $d(\cdot, \cdot)$ is the usual Euclidean metric between two real numbers.

This definition allows us to reason about the time instances that are included between two fixed times. Remember that PCN is defined on an arbitrary time structure. Hence when reasoning about time intervals, it is useful to be able to assess what other times are included between time $t$ and $t + \tau$. For example, for $\mathcal{T} = \mathbb{N}$ and $t_1 = 1, \tau = 4$, the set $\mathcal{T}_{t+\tau}$ is simply $\{1, 2, 3, 4, 5\}$ while for the time domain $\mathcal{T} = \mathbb{R}$, this set is the closed interval $[1, 5]$.

Given PATTL formulae $\varphi_1$ and $\varphi_2$, the two real-time operators that augment PATL are defined as follows:

real-time Until: $v \models_t \varphi_1 U^\tau \varphi_2$ iff $\exists t' \in \mathcal{T}_{t+\tau}, v \models_{t'} \varphi_2$ and $\forall t''$, $t \leq t'' < t'$, $v \models_{t''} \varphi_1$.  \hspace{1cm} (8.1)

real-time Since: $v \models_t \varphi_1 S^\tau \varphi_2$ iff $\exists t' \in \mathcal{T}_{t-\tau}, v \models_{t'} \varphi_2$ and $\forall t''$, $t' < t'' \leq t$, $v \models_{t''} \varphi_1$.  \hspace{1cm} (8.2)

As one can observe from their syntax, the timed versions $U^\tau$ and $S^\tau$ simply extend $U$ and $S$ to allow for temporal reasoning. The formula $\varphi_1 U^\tau \varphi_2$ expresses the property that $\varphi_2$ will become true within $\tau$ time units, period during which $\varphi_1$ will remain true continuously, while the formula
\( \varphi_1 S^r \varphi_2 \) expresses the property that \( \varphi_2 \) was true at some time value \( t' \) within the interval \( [t - \tau, t] \), and for every subsequent time value, \( \varphi_1 \) held continuously until time \( t \).

Based on those two real-time operators, we can define other real-time operators that can prove useful for specifying real-time properties of systems.

**Eventually:**
\[
\Diamond^r \varphi \equiv \text{true} U^r \varphi.
\]

**Always:**
\[
\Box^r \varphi \equiv \text{true} S^r \varphi.
\]

The real-time Always operators \( \Box^r \varphi \) and \( \Box_r \varphi \) can also be derived using the duality of lower and upper probability bounds: \( \mathcal{P}_{\geq p}(\Box^r \varphi) = \mathcal{P}_{\leq 1-p}(\Diamond^r \neg \varphi) \) and \( \mathcal{P}_{\geq p}(\Box_r \varphi) = \mathcal{P}_{\leq 1-p}(\Diamond_r \neg \varphi) \).

The semantics of these new real-time operators is defined by:

**Eventually:**
\[
v \models_t \Diamond^r \varphi \iff \exists t' \in T_{t+r}, v \models_{t'} \varphi.
\]
\[
v \models_t \Box^r \varphi \iff \forall t' \in T_{t+r}, v \models_{t'} \varphi.
\]

**Always:**
\[
v \models_t \Box^r \varphi \iff \forall t' \in T_{t+r}, v \models_{t'} \varphi.
\]
\[
v \models_t \Box_r \varphi \iff \forall t' \in T_{t-r}, v \models_{t'} \varphi.
\]

Now let us come back to the specification for the elevator task introduced in the PATL section. With the real-time temporal operators, we are now able to express the property: “Whenever a passenger enters the elevator (E), then there is a probability of at least 95% that the passenger will be delivered to his destination floor (D) within \( \tau = 30 \) time units. In PATTL, this specification is denoted by \( \Box (E \rightarrow P_{0.95}(\text{true} U^{30} D)) \).

We define the size of a PATTL real-time logical operator as \( \log(k) \) where \( k \) is the number of time steps included within \( T_{t+r} \), with \( \tau \) being the time threshold of the real-time operator. For instance, assume that we have the PATTL formula \( \varphi_1 U^T \varphi_2 \) with \( T = N \). In this case, \( k = \tau = 7 \), i.e., there is 7 transitions that will occur between time \( t \) and time \( t + \tau \). For general time structures with equally spaced increments of \( \Delta \), there will be \( \max \{ k \Delta \leq \tau \} \) time steps. The logarithmic property of the size of the real-time operators comes from the fact that in matrix representation \( P^k \), which is the probability transition matrix after \( k \) steps, can be computed in \( O(\log k) \) matrix multiplications. Finally, we also define the size \( |\varphi| \) of a PATTL formula \( \varphi \) as the number of propositional connectives.
and temporal operators added to the sum of the sizes of the real-time operators in $\varphi$.

### 8.4 Time Structure and Verification Issues

In order to proceed with the verification of PATL and PATTL formulae we need to first specify the meaning of the time bound $\tau$ in the temporal formulae $U_T$ and $S_T$. Indeed, since PCN models are defined over arbitrary time structures, one needs to have a clear understanding of the time bounds. We view the time bound $\tau$ as the absolute time reference, independent from the time structure of the PCN model under study. For instance, assume PCN with a time structure such as $\mathcal{T} = \{3t | t \in \mathbb{N}\}$. A trace on such a time structure would have a value for every time point in $\mathcal{T} = \{0, 3, 6, 9, \ldots\}$. If one desires to verify a property such as $\varphi_1 U^2 \varphi_2$, it is important to note that one should only be interested in the first value of the trace, as any other value occurs after time $t = 2$ and hence has no effect in the satisfaction (or dissatisfaction as it may be the case) of the behavioural constraint. However, one could ask what is the value of the trace at time $t = 2$ since it may have a crucial effect on the satisfaction of the formula $\varphi_1 U^2 \varphi_2$. Thus, one needs to pay close attention to the meaning of time along with the various time structures that a system can have.

#### 8.4.1 The Structure of Time

Time is an essential component of dynamical systems. Hence, understanding time is absolutely necessary to understanding dynamics. Our approach consists of using an abstract structure to formalize time so that it encompasses its most important aspects. In general, we view a time structure as a linearly ordered set with a well-defined start point, a metric associated to the distance between any two time points and a measure defined over the duration of an interval of time. This approach is analogous to the results obtained for the CN framework [Zha94], which we summarized in Chapter 2.

As mentioned earlier, the PCN framework allows for the modeling of hybrid systems. A hybrid system consists of subsystems acting on different time structures. Hence when considering the verification of temporal logic formulae, one needs to ensure that the systems’ traces are well defined for any value of the time bound $\tau$, which is independent of the time structures of the system. For systems modeled with multiple clocks, sampling and extending are common transductions that are used to synchronized the various components.
When verifying multi-clock systems, we will assume that the trace of the system has been extended to the most dense time structure that takes into account all the various clocks of the system. Based on this assumption, we present the following proposition which guarantees that no matter what reference time is used, the verification method is still sound.

**Proposition 8.2** Assume a PCN trace \( v : T \rightarrow A \) which satisfies a PATTL formula \( \varphi \). Let \( T_r \) be any reference time of \( T \) with a reference time mapping \( h : T \rightarrow T_r \). Then the extension trace \( \bar{v} : T_r \rightarrow A \) of \( v \) onto \( T_r \) also satisfies \( \varphi \).

The effect of an extending transduction on the transition probability of a given location is very simple. For every extended value, the probability that a location remains in the same state is one, while the probability that the location takes on any other value in its domain is 0. Hence, we generalize the notion of probability distribution induced by a PCN with multiple clocks, which we denote with \( P^{\mathcal{X}}_r \), where \( r \) refers to the most dense reference time structure of the system.

**Definition 8.9** Transition probability distribution induced by a PCN with multiple clocks
Assume a PCN \( \mathcal{X} = (L, T, P, C) \), a frame \( (T, A, P^{\mathcal{X}}, V) \) and a reference time \( T_r \) of \( T \) with a reference time mapping \( h \). The probability distribution induced by the PCN over the reference time structure \( T_r \) is defined as follows:

\[
P^{\mathcal{X}}_r(a_{t_1}, a_{t_i+1}) = \begin{cases} 
P^{\mathcal{X}}(a_{t_i}, a_{t_{i+1}}) & \text{if no locations are extended at } t_{i+1} \\ 0 & \text{if any } L_j \text{ are extended at } t_{i+1} \text{ and } a_{t_i}^j \neq a_{t_{i+1}}^j \\ P_{L_1} \times \cdots P_{L_k} & \text{if } L_1, \ldots, L_k \text{ are not extended at } t_{i+1} \text{ and for all extended locations } L_j, j = k + 1, \ldots, n, \\
\text{we have } a_{t_i}^j = a_{t_{i+1}}^j & \end{cases}
\]

We need to add an extra parameter to the transition probability distribution to take into account the current time value. Indeed, the temporal location of the system will influence the value of the transition probabilities. If a location is getting extended at time \( t \), then its value will be the same than at the previous time step with probability one. This will be important when describing the verification algorithms as it enable us to verify systems with arbitrary non-continuous time structures. The reader should note that these results not only apply to multi-clocked systems but also to event-driven systems. An event-driven transduction works in a way that is very analogous...
to a transduction with input of different clocks. Indeed, for an event-driven transduction, the input trace with reference time $T$ is first sampled onto the sample time $T_e$ generated by the event trace $e$. Secondly, the primitive transduction is performed on the time structure $T_e$. Finally, the output trace is extended from $T_e$ back to the initial time structure $T$.

Another important fact is that the total number of different probability distributions induced by a PCN with multiple clocks is always finite. The following proposition summarizes this result.

**Proposition 8.3** For any PCN with multiple clocks, the number of different induced probability distribution is finite and at most $2^n - 1$, where $n$ is the size of the location set $L$.

### 8.5 Model Checking of PATTL over finite PCN and non-continuous time structure

In this section we present an iterative model checking algorithm, which, for a given frame $F = \langle T, A, \mathcal{P}, V \rangle$ and a PATTL formula $\varphi$ determines whether a PCN model with finite domain satisfies the formula $\varphi$. The algorithm terminates when each state is labeled with the set of sub-formulae of $\varphi$ that are true at that state. We will focus on PATTL formulae since as we will show, the verification of PATL formulae collapses into the PATTL verification procedure with $\tau = \infty$.

As mentioned previously, the PCN framework is designed to handle arbitrary time structures, whether, discrete, continuous or event-based. Although our algorithm is based on CTL model checking [CES86] and on the PCTL model checker of [HJ94], it is more general as it can handle arbitrary discrete and event-based time structures. For PCN models with a continuous time structure, we will present another algorithm, based on the CSL model checker of [ASSB00, BHHK03], which can handle continuous-time systems where the time delay between transition events is distributed exponentially. In this setting, we will also discuss another probabilistic operator, $\mathcal{L}_{\ominus p}(\varphi)$, as introduced within the CSL logic, which allows for the specification of steady-states requirements.

Similarly to other model checking algorithms, in order to proceed with the iterative labeling of each state, our algorithm introduces a variable $valid(v_t)$ that indicates which sub-formulae of $\varphi$ have been shown to be true at state $v_t$. As a base case, each state $v_t$ is labeled with the atomic propositions that are true in $v_t$, that is, we have at first $valid(v_t) = \forall v_t \in A, \forall v_t \in A, where V(\cdot)$ is the interpretation function obtained from the frame $F$. Then, the labeling of the states is performed.
starting with the smallest sub-formulae of \( \varphi \) that remain to be labeled until the algorithm labels the states with \( \varphi \) itself. Obviously, the labeling of composite formulae depends on the labeling of their parts. For instance, the label \( \varphi_1 \land \varphi_2 \) is added to \( \text{valid}(v_t) \) if \( \varphi_1 \in \text{valid}(v_t) \) and \( \varphi_2 \in \text{valid}(v_t) \). The other logical connectives such as \( \neg, \lor, \rightarrow, \leftrightarrow \) are handled similarly.

### 8.5.1 Verifying modal formulae in PATTL

Let us now present the algorithms for handling the labeling of states with modal formulae of the form \( \mathcal{P}_{\oplus}(X\varphi), \mathcal{P}_{\oplus}(\varphi_1 U^\tau \varphi_2) \) and \( \mathcal{P}_{\oplus}(\varphi_1 S^\tau \varphi_2) \). We will verify these formulae for the different sets of values of \( p: p > 0, 0 < p < 1 \) and \( p \geq 1 \) and for the different possible values of \( \tau: \tau = 0, 0 < \tau < \infty \) (\( \tau \) is finite) and \( \tau = \infty \). One should note that, as mentioned before, the cases of \( p = 0 \) and \( p \geq 1 \) correspond to the existential and universal quantifiers, respectively, with the exception that these operators are restricted to non-measure zero sets of paths. The case of \( \tau = \infty \) collapses to the PATL version of the operators. Hence PATTL verification encompasses PATL verification as one of its sub-case.

For the remaining of this section, we will assume that the system's coarsest time structure \( T_r \) is known and the increments of the time structure are regularly spaced by \( \Delta_r > 0 \). The increments of an event-based time structure, or even multi-clocked, might not be equally spaced. However, the results provided here will also hold for this case with the exception that the notation is more complicated and harder to follow. Hence, without loss of generality, we only depict the case for regular increments.

<table>
<thead>
<tr>
<th>Operator</th>
<th>( \tau )</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X\varphi )</td>
<td>(-)</td>
<td>use ( P_r^x )</td>
</tr>
<tr>
<td>( \varphi_1 U^\tau \varphi_2 )</td>
<td>( 0 )</td>
<td>equivalent to ( v \models \varphi_2 )</td>
</tr>
<tr>
<td></td>
<td>( (0, \infty) )</td>
<td>compute ( \mathcal{R}(\tau, v(t), t) )</td>
</tr>
<tr>
<td></td>
<td>( \infty )</td>
<td>simple clock: compute ( \mathcal{B}(\infty, v(t)) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>multiple clocks with cycles: compute ( \mathcal{B}(\infty, v(t), n) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>multiple clocks without cycles: approximate ( \mathcal{B}(\infty, v(t), t) )</td>
</tr>
<tr>
<td>( \varphi_1 S^\tau \varphi_2 )</td>
<td>(-)</td>
<td>reversible distribution: see ( \varphi_1 U^\tau \varphi_2 )</td>
</tr>
<tr>
<td></td>
<td>( (0, t) )</td>
<td>non-reversible distribution: compute ( \mathcal{D}(\tau, v_t) )</td>
</tr>
</tbody>
</table>

Table 8.1: Comparing the diverse cases

Table 8.5.1 summarizes the different cases that we will consider, along with a brief description
of the method used.

### 8.5.2 Model checking of the $X$ operator

Given a PCN frame $(T, A, \mathbb{P}^X, V)$, the verification of a PATTL formula of the form $\mathcal{P}_{\exists p}(X \varphi)$ directly involves the transition probability distribution matrix $\mathbb{P}^X$.

To obtain the set of states which satisfy $\mathcal{P}_{\exists p}(X \varphi)$, simply construct a column vector $n^\varphi: A \rightarrow \{0, 1\}$ given by $n^\varphi(v) = 1$ if $\varphi \in valid(v)$ and 0 otherwise. Then, the set of satisfactory states is obtained via $\{v \in A | x_v \cap p\}$ where $x_v = \mathbb{P}^X_x \cdot n^\varphi$.

Note that in the case of multi-clocked systems, the next operator is exclusively defined at the coarsest time structure of the system, i.e., the time at which the next transition occurs in one of the locations of the system. Hence we need to use the generalized transition probability matrix $\mathbb{P}^X$ in our calculations.

### 8.5.3 Model checking of the $U^\tau$ operator when $\tau = 0$

In this case, it is easy to show that the satisfaction of $\mathcal{P}_{\exists p}(\varphi_1 U^0 \varphi_2)$ collapses to showing $v \models \varphi_2$, regardless of the value of the probability threshold $p$ [HJ94].

### 8.5.4 Model checking of the $U^\tau$ operator when $0 < \tau < \infty$

Here we consider the case where the time threshold $\tau$ is finite and the probability threshold $p$ is comprised between 0 and 1 inclusively. The special cases for $p = 0$ and $p = 1$ can be solved more efficiently if handled separately as they correspond to the RTCTL model checking for the formula $E(\varphi_1 U^{\leq \tau} \varphi_2)$ and $A(\varphi_1 U^{\leq \tau} \varphi_2)$, respectively [EMSS92]. For an example of such improvements for systems modeled as Markov chains, the reader is referred to [HJ94].

Assuming that we have done the labeling of the states for formulae $\varphi_1$ and $\varphi_2$, we now give an algorithm for labeling states with the PATTL formula $\mathcal{P}_{\exists p}(\varphi_1 U^\tau \varphi_2)$, for $0 < \tau < \infty$. We generalize the approach of Hansson and Jonsson [HJ94], by extending the notion of measure of paths satisfying until formulae to handle arbitrary non-continuous time structures over a PCN model.

**Definition 8.10** *Path Measure for until formulae satisfaction*
Assume a PCN structure $\mathcal{X}$, a frame $\langle T, A, \mathbb{P}_r^T, V \rangle$ and a real value $\tau \geq 0$. For a state $v_t$ of the system, the function $B(\tau, v_t, t)$ defines the measure of the set of traces $v$ in $B_v^T$ for which $v \models_t \varphi_1 U^T \varphi_2$. For negative values of $\tau$, we define $B(\tau, v_t, t) = 0$.

**Proposition 8.4** For $\tau \geq 0$, $B(\tau, v_t, t)$ is obtained via the following recurrence relation:

$$
B(\tau, v_t, t) = \begin{cases} 
1 & \text{if } \varphi_2 \in \text{valid}(v_t) \\
0 & \text{elseif } \varphi_1 \notin \text{valid}(v_t) \\
\sum_{v' \in A} \mathbb{P}_r^T(v_t, v', t + \Delta_r) \times B(\tau - \Delta_r, v', t + \Delta_r) & \text{otherwise}
\end{cases}
$$

Note that since $\Delta_r$ is positive, we are guaranteed that the recurrence relation will be well-defined and always terminate (after $k > 0$ steps when $\tau - k\Delta_r$ becomes negative).

Based on the result above, we have that $B(\tau, v_t, t)$ provides the probability with which a certain state, namely $v_t$, satisfies the PATTL formula $\varphi_1 U^T \varphi_2$. Hence, we label state $v_t$ with $\varphi_1 U^T \varphi_2$ if $B(\tau, v_t, t) \geq p$.

In Figure 8.5.4, we provide an algorithm, based on Equation A.5 for calculating the value of $B(\tau, v_t, t)$. Let us assume that time is equally spaced by $\Delta_r$ and that $k \in \mathbb{N}$ is the maximal value such that $k\Delta_r \leq \tau$.

The worst case complexity of the above algorithm is $((k + 1) \times (|A| + 1) \times 2 \times |A|)$, where $|A|$ is the size of the state space of the PCN model. The most outer loop is run $k + 1$ times while the for all loops are run through $|A|$ times, with an additional assignment statement within the first if statement. Finally, the factor 2 arises from the two additional assignments performed in the second for all loop. Therefore, we can conclude that our algorithm requires $O(k \times |A|^2)$ operations. However, for most PCN, the diverse deterministic transductions will be such that a large portion of the state space (the cross-product of all location domains) will generate transitions with probability 0. Indeed, consider an addition transduction which adds up the value of location $x$ and $y$ and for which the domain of the output location $z$ is $\mathbb{N}$. In this case, the probability of reaching any state of the form $\{x = 1, y = 1, z = z'\}$, will be 0 for every $z' \in \mathbb{N}\{2\}$. Hence, we can significantly improve the performance of the algorithm. In fact, if one considers the transitions with non-zero probability, the algorithm now requires $O(k \times (|A| + |B|))$, where $|B|$ denotes the number of
for (i := 0 to k) {
    for all (v ∈ A) {
        if (φ₂ ∈ valid(v)) {
            \( \mathcal{B}(iΔ_r, v, t + (i + 1)Δ_r) := 1 \)
        } else {
            \( \mathcal{B}(iΔ_r, v, t + (i + 1)Δ_r) := 0; \)
            if (φ₁ ∈ valid(v)) {
                for all (v′ ∈ A) {
                    \( \mathcal{B}(iΔ_r, v, t + (i + 1)Δ_r) := \mathcal{B}(iΔ_r, v, t + (i + 1)Δ_r) + \mathcal{B}(iΔ_r, v′, t + (i + 1)Δ_r) \times \mathcal{B}((i - 1)Δ_r, v, t + (i + 2)Δ_r) \)
                }
            }
        }
    }
}

Figure 8.2: Algorithm for calculating \( \mathcal{B}(\cdot) \)

transitions in \( \mathbb{P}^X_r \) with positive probability. One can easily see that, for a PCN where the whole state space \( A \) is reachable from any state \( v ∈ A \), the two complexity expressions are equivalents as in that case \( |B| = |A|^2 \).

Using the expression for the size of PATTL formulae and real-time operators, we can now define the complexity of deciding whether a PCN model satisfies a formula of the form \( φ = φ_1 U^rφ_2 \). It is easy to show that the worst case complexity for this problem is \( O(k_{max} \times (|A| + |B|) \times |φ|) \), where \( |φ| \) is the size of the PATTL formula \( φ \), \( k_{max} \) is the maximum number of transitions between the current time and the time threshold \( r \) and \( |A| \) and \( |B| \) are as previously defined.

### 8.5.5 Model checking of the \( U^r \) operator when \( r = \infty \)

Any PATTL formula with \( r = \infty \) is equivalent to the PATL counterpart. Hence model checking of \( φ = φ_1 U^\infty φ_2 \) is also applicable to the model checking of \( φ_1 U φ_2 \).

However, it is easy to see that the preceding algorithm cannot be used to model check formulae with \( r = \infty \). Indeed, in this case, the algorithm would require infinite calculations hence rendering the model checking procedure infeasible. Therefore, we have to adopt a different approach to solve formulae with infinite time threshold (or absence of temporal component as in PATL). Let us define \( \mathcal{B}(\infty, v_t) \) as the measure of the set of paths \( w ∈ \mathcal{B}^X_v \) which satisfy \( φ \).
In this section, we develop algorithms to model check PCN models of systems with arbitrary non-continuous time structures. We will consider three distinct families of time structure, each of which will require a different algorithm to be solved. The three families under consideration are:

1. single-clock system, which amounts to stationary transition distributions;
2. multi-clock systems where the induced transitions distribution are cyclical, i.e., the order of the probability distribution repeats itself according to a finite period;
3. multi-clock systems with no structure in the order of the probability transition distribution.

**Single-clock systems**

For a system within the family of single-clocked systems, a sole probability transition distribution \( \mathbb{P}^\mathcal{X} \) is induced. This distribution results from the cross-product of all the probability distribution of the locations of the system.

Let \( \mathcal{X} = \{v_1, \ldots, v_n\} \) denote the state space of a PCN \( \mathcal{X} \). We partition the space \( \mathcal{X} \) into three subsets, denoted by \( A_v^\mathcal{X}, A_f^\mathcal{X} \) and \( A_e^\mathcal{X} \) in the following way:

- \( A_v^\mathcal{X} \): The valid states which satisfy \( \varphi \) with probability exactly 1.
- \( A_f^\mathcal{X} \): The failure states which satisfy \( \varphi \) with probability exactly 0.
- \( A_e^\mathcal{X} = \mathcal{X} \setminus (A_v^\mathcal{X} \cup A_f^\mathcal{X}) \). The equivocal states are states for which a conclusion cannot be reached but which have a positive probability of reaching a valid state. For every state \( v \in A_e^\mathcal{X} \), we have that \( \varphi_1 \in \text{valid}(v) \) and \( \varphi_2 \notin \text{valid}(v) \) with the extra requirement that there exists a path with non-zero measure from \( v \) to a state \( w \in A_v^\mathcal{X} \). Hence, these states will form the paths that lead to a valid state.

The first problem is the identification of the partitions. It has been showed that \( A_v^\mathcal{X} \) and \( A_f^\mathcal{X} \) can be determined by ordinary fix point computation [CY95]. Once \( A_v^\mathcal{X} \) and \( A_f^\mathcal{X} \) are known, it is easy to compute \( A_e^\mathcal{X} \) by using set difference. Once these partitions have been identified, the next step consists of solving the set of linear equations defined by:
The solution of the linear equation systems can be computed by any direct method such as \textit{Gaussian elimination}, or iterative method like \textit{Jacobi} or \textit{Gauss-Seidel}. For Gaussian elimination, the problem can be solved with a complexity of $O((|A| - |A_f^X| - |A_v^X|)^2) \ [\text{AHU74}].$

**Multi-clock systems with cycles**

In the general case of multi-clock PCN models, we have a non-stationary set of linear equations with a parameter of infinite domain. Hence Gaussian elimination is not feasible in this case. However, for systems where the order of the induced transition probability distributions is cyclical, we can still use iterative methods such as Gaussian elimination. Let us assume that the period of the cycle is equal to $n_p$. Then, using the same partitions as above, we need to solve the following set of linear equations:

$$\mathcal{B}(\infty, v_t, p) = \begin{cases} 
1 & \text{if } v_t \in A_v^X \\
0 & \text{elseif } v_t \in A_f^X \\
\sum_{v' \in A^X} P^X_r(v_t, v') \times \mathcal{B}(\infty, v') & \text{otherwise}
\end{cases}$$

where $\mod$ is the usual modulo operator.

In fact, any system, for which its clocks have equally spaced increments, will generate a cycle in the order of the induced transition probability distributions. The following proposition demonstrates this fact and indirectly also shows that we can use the set of linear Equations 8.3 to solve for $\mathcal{B}(\infty, v_t, n).$

**Proposition 8.5** Let $Cl = \{c_1, c_2, \ldots, c_k\}$ be the set of all clocks of a PCN $X$. If for every clock in $Cl$ the time increments are equally spaced, then there exist a finite cycle in the order of the transition distributions $P^X_r$. 

157
Multi-clock systems without cycles

For fully non-stationary systems without any well-defined structure such as cycles, the model checking problem is more complicated. In fact, the following theorem shows that for such systems, there does not exist a finite iterative method that can solve the following model checking task exactly.

\[
\mathcal{B}(\infty, v_t, t) = \begin{cases} 
1 & \text{if } v_t \in A^X_i \\
0 & \text{elseif } v_t \in A^X_j \\
\sum_{v' \in A^X} \mathbb{P}_\tau^X(v_t, v', t + \Delta_\tau) \times \mathcal{B}(\infty, v', t + \Delta_\tau) & \text{otherwise}
\end{cases}
\] (8.4)

Let us first introduce the corollary which will be used in proving the theorem below.

**Corollary 8.1** \( \forall p \in (0, 1), \rho \in \mathbb{R}, \text{ and any } k \in \mathbb{N}, \text{ there exist a subset } I \subseteq \{1, \ldots, k\} \text{ such that the following inequality holds:} \)

\[ p - \frac{1}{2^k} \leq \sum_{i \in I} \frac{1}{2^i} \leq p \] (8.5)

**Theorem 8.1** There does not exist a finite iterative procedure to solve the set of linear Equations 8.4 for any order on the induced probability distributions.

The results of Theorem 8.1 indicates that the measure of the paths satisfying \( \varphi \) cannot be computed exactly. Hence, we need to proceed with an approximate procedure. To do so, we will once again refer to the partition of the state space that was described in the previous section. Based on these three partitions, we define the \(|A| \times |A|\)-matrices \( \mathcal{M}_t \) by

\[
\mathcal{M}_t[v_i, v_j] = \begin{cases} 
\mathcal{P}_\tau^X(v_i, v_j, t + \Delta_\tau) & \text{if } v_i \in A^X_e \\
1 & \text{if } v_i \notin A^X_e \land i \neq j \\
0 & \text{otherwise}
\end{cases}
\] (8.6)

For nonnegative values of \( t \), we define \( \mathcal{B}(\tau, t) \) as the column vector of size \(|A|\) whose \( i \)th element \( \mathcal{B}(\tau, t)_i \) is \( \mathcal{B}(\tau, v_i, t) \). Therefore, we have that \( \mathcal{B}(0, t)_i \) is 1 if \( v_i \in A^X_e \) and 0 otherwise. From this, we to obtain the following proposition:
Proposition 8.6 For any \( r > 0 \), we have

\[
\mathcal{B}(\tau, t) = M_t \times M_{t+\Delta r} \times \cdots \times M_{t+\tau \Delta r} \times \mathcal{B}(0, t) = \overline{M} \times \mathcal{B}(0, t)
\]  \hspace{1cm} (8.7)

For finite values of \( \tau \), it is easy to show that Equation 8.7 yields the same result as recurrence equation of Proposition 8.4 [HJ94]. However, for \( \tau = \infty \), it would take an infinite number of matrix multiplication to obtain the exact result (as it was shown in Theorem 8.1). However, we can proceed to an number of multiplication, knowing that in the limit, the result will converge to the true path measure. In fact, \( \lim_{r \to \infty} \mathcal{B}(\tau, t)_i = \mathcal{B}(\infty, v_i, t) \). Using this method, we can bound the result from both above and below, hence obtaining a good assessment of the value of the approximation.

Proposition 8.7 Bounds on path measure

Let \( M \) be the number of matrix multiplication such that we have \( \mathcal{B}(\infty, t) = M_t \times M_{t+\Delta r} \times \cdots \times M_{t+M\Delta r} \times \mathcal{B}(0, t) = \overline{M} \times \mathcal{B}(0, t) \). Furthermore, let \( \delta_i^M = \sum_{|u_j \in A^e_i \overline{M}[i,j], \text{ with } i \in \{k|v_k \in A^e_j}\}} \).

Then, for every equivocal state \( v_i \), the true measure \( \mathcal{B}(\infty, v_i, t) \) of the paths satisfying \( \varphi \) lies between the following bounds:

\[
\mathcal{B}(\infty, t)_i \leq \mathcal{B}(\infty, v_i, t) \leq \mathcal{B}(\infty, t)_i + \delta_i^M
\]  \hspace{1cm} (8.8)

8.5.6 Model checking of the \( S^\tau \) operator

In this section, we will show how to model check the PATTL formula \( \varphi_1 S^\tau \varphi_2 \). Note that the untimed PATL formula \( \varphi_1 S \varphi_2 \) is equivalent to the PATTL formula with \( \tau = t \). Indeed, because the since operator looks back in the past, the maximum value of \( \tau \), for the formula to be well-defined, is \( t \). Hence, we will always deal with finite time threshold when model checking a formula with the since operator.

One interesting case arises when the induced probability distribution of the PCN model is time stationary and a limiting distribution \( \pi \) exists. In this case, and for large enough \( t \) values, the induced probability distribution is reversible and we can solve the formula \( P_{\mathcal{O}_p}(\varphi_1 S^\tau \varphi_2) \) using
the method of the until formula with the reversed induced transition probability distribution $\mathcal{Q}_X$, which is obtained via the following equation:

$$\mathcal{Q}_X(i, j) = \frac{\pi_j \mathcal{Q}_X(j, i)}{\pi_i}$$

(8.9)

Note that we are using the fact that the system is running at stationary level, hence the assumption of a large $t$ value. It is also well-known that if a system is reversible, then the rate of transitions $i \rightarrow j$ is equal to the rate of the inverse transition $j \rightarrow i$. This result comes from the fact that the operators $\mathcal{U}$ and $\mathcal{S}$ are symmetric. Hence for reversible systems, both formulae are solved by the same technique.

**Example 8.2 Reversible system**

As an example of a system that has reversible dynamics, consider a PCN model $X$ of a random walk on the set $\{0, 1, 2, \ldots, N\}$. A random walk on the integers (or a subset of the integers) is defined as moving either one step up or one step down during each time step. In general, the transition probabilities are of the form

$$\mathcal{Q}_X(i, i + 1) = \alpha_i = 1 - \mathcal{Q}_X(i, i - 1)$$

$$\mathcal{Q}_X(0, 1) = \alpha_0 = 1 - \mathcal{Q}_X(0, 0)$$

$$\mathcal{Q}_X(N, N) = \alpha_N = 1 - \mathcal{Q}_X(N, N - 1)$$

(8.10)

To show that this system is reversible, consider a process that moves up from position $0 < i < N$. If the process is to move up from $i$ once again, then it had to have moved down from $i + 1$ since there is only one way back to state $i$ and that state is via $i + 1$. Therefore, for each move up at $i$, there must have been a move down from $i + 1$. Hence, the rates up and down across the $i \leftrightarrow i + 1$ boundary are equal. $\square$

For non-reversible systems, the problem of model checking the formula $\mathcal{P}_{\mathcal{U}p}(\varphi_1 S^\tau \varphi_2)$ amounts to finding the measure of all the paths for which a $\varphi_2$ state is reached within $[t - \tau, t]$ and, for all subsequent time step up to $t$, the paths never enter a $\neg\varphi_1$ state.

Let us define $\mathcal{D}(\tau, v_t, t)$ as the measure of the paths, ending at state $v_t$ at time $t$ which satisfy the PATTL formula of the form $\varphi_1 S^\tau \varphi_2$.

**Definition 8.11 Path Measure for since formulae satisfaction**
Assume a PCN structure $\mathcal{H}$, a frame $\langle T, A, P_r^X, V \rangle$ and a real value $\tau \geq 0$. Also assume that the time structure of the system has regular time increments, namely $\Delta_r$. Furthermore, for sake of simplicity, let us assume that the induced transition probability distribution is stationary. The extension to the non-stationary case is straightforward although it is notationally more involved.

For a state $v_t$ of the system, the function $\mathcal{D}(\tau, v_t)$ defines the measure of the set of traces $v$ in $B^X$ for which $v \models_\tau \varphi_1 S^\tau \varphi_2$. For negative values of $\tau$, we define $\mathcal{D}(\tau, v_t) = 0$.

The following proposition presents how to calculate the results for the paths measure defined above.

Proposition 8.8 Let us consider two separate cases for the value of the threshold $\tau$.

- For $\tau < \Delta_r$ or $t = 0$, $\mathcal{D}(\tau, v_t)$ is obtained via the following equation:

$$\mathcal{D}(\tau, v_t) = n^{\varphi_2}$$  \hspace{1cm} (8.11)

- For $\tau \geq \Delta_r$, the paths measure is obtained with

$$\mathcal{D}(\tau, v_t) = \left[ P_0^X \times (P_r^X)^{M_1} \times \left[ \sum_{k=1}^{M_2} (P_r^X)^{k-1} M_{\varphi_2 \rightarrow \varphi_1} M_{t \rightarrow \varphi_1}^{M_2-k} \right] \right]' + n^{\varphi_2}(i)$$  \hspace{1cm} (8.12)

and where $P_0^X$ is the vector of the system's initial state's distribution, $M_1 = \left[ \frac{\max(0, t-\tau)}{\Delta_r} \right]$, $M_2 = \left[ \frac{\min(t, \tau)}{\Delta_r} \right]$, and the matrices $M_{\varphi_k \rightarrow \varphi_1}$ are defined as follows:

$$M_{\varphi_k \rightarrow \varphi_1}(v_i, v_j) = \begin{cases} P_r^X(v_i, v_j) & \text{if } v_i \models_\tau \varphi_k \text{ and } v_j \models_\tau \varphi_1 \\ 0 & \text{otherwise} \end{cases}$$  \hspace{1cm} (8.13)

8.6 Model checking of PATTL formulae with continuous time structures

When dealing with a system acting on a finite state space and a continuous time structure, some properties of the probabilistic behaviour are required. For instance, the numbers of transitions in non-overlapping intervals should be independent for all intervals. The occurrence of a transition
should in fact be independent of when the previous one occurred. Furthermore, to ensure fairness, the probability of two or more transitions in a sufficiently small interval should converge to 0 as the interval's width diminishes.

As we have seen earlier, a way to ensure that the above properties are satisfied is to model the system via event-driven probabilistic transductions where the events are subject to an exponentially distributed random delay. We explained how one arrives to the choice of the exponential distribution as the model for transitions delays in Chapter 4

Within a PCN model, the event-driven probabilistic transductions can be triggered by the same event generator or there can be one event generator per transductions. This is characterized by the respective behaviour of the system: 1- the value of every locations changes at every event, 2- a transition of the system only implies one location changing value. Hence in the second case, we are faced with a local transition within the state space.

Let us now approach the problem of verifying PATTL formulae for continuous time systems. As we have shown previously, given a PCN model where the probabilistic transductions are driven by events with exponentially distributed random delays, we can construct a rate matrix $\mathcal{R} : A \times A \rightarrow \mathbb{R}^+$ which denotes the rate at which the system will transition to a new state. Once the rate matrix is generated, the verification method is essentially analogous to model checking continuous-time Markov chains (CTMC). Hence, in the remainder of this section, we will restrict ourselves to a brief description of the method. For a thorough introduction to the methods, the reader is referred to the seminal work on verifying temporal formulae on CTMCs [BHHK00, ASSB96].

Using the rate matrix $\mathcal{R}$, let us define the matrix

$$\mathcal{E}(v) = \sum_{v' \in A} \mathcal{R}(v, v'). \quad (8.14)$$

$\mathcal{E}(v)$ denotes the the rate with which the system will be taking a transition from state $v$ to any other state (including state $v$ itself). Given the exponential nature of the transition delays, it is easy to show that the probability that the system will take a transition from state $v$ within $t$ time units is equal to $1 - e^{-\mathcal{E}(v)t}$. The probability of the destination state $v'$ is obtained via what is called a race condition. That is, the probability $\mathbb{P}(v, v')$ for the system to transition from state $v$ to $v'$ equals the probability that the delay of the transition for going to $v'$ ends before any other transition delay. Hence we have that $\mathbb{P}(v, v') = \mathcal{R}(v, v')/\mathcal{E}(v)$ if $\mathcal{E}(v) > 0$. Otherwise, $\mathbb{P}(v, v) = 1$ since the state $v$
is absorbing and the transition delay is infinite.

In this setting, the time between transition is unknown (exponentially distributed). Therefore, when proceeding to the verification of PATTL formulae, we need to consider the time between transition explicitly. Hence we define a path in a PCN with exponentially distributed transition delays as an augmented trace $v_0 t_0 v_1 t_1 v_2 \ldots$ where $\mathcal{R}(v_i, v_{i+1}) > 0$ and $t_i \in \mathbb{R}^+$ for all $i \geq 0$. Note that since a transition can lead from $v_i$ to $v_i$ itself, $t_i$ does not represent the time spent in $v_i$ but rather the time spent in state $v_i$ before a transition occurs. The need to explicitly take self-transition comes from the construction of the rate matrix $\mathcal{R}$ from the underlying PCN model, as it was explained earlier.

Denoting the set of infinite traces starting in state $v$ by $\mathcal{B}_v$, we define the probability measure as in [BKH99]. Assuming that the states $v_0, \ldots, v_n \in A$ satisfy the condition $\mathcal{R}(v_i, v_{i+1}) > 0$, for all $0 \leq i < n$ and that the time intervals $I_0, \ldots, I_{n-1}$ are non-empty, nonnegative intervals of the real line, then we can define the cylinder set $C(v_0, I_0, \ldots, I_{n-1}, v_n)$ as the set containing all traces $v_0 t_0 v_1 t_1 v_2 \ldots$ where $t_i \in I_i$ for $i < n$. This measure is made unique by simply completing the cylinders to the least $\sigma$-algebra.

With this notion of path measure, one can use the PATTL logic introduced earlier to analyze the transient (state of the system at a specific time instant) behaviour of systems, as it was in the previous sections. Moreover, to analyze the steady-state (in the long-run) behaviour of the system, the CSL logic [ASSB96, BKH99] introduced a steady-state operator $\mathcal{S}$, with syntax $\mathcal{S}_{\text{op}}(\varphi)$. The semantics of the steady-state operator is defined as follows

$$v \models \mathcal{S}_{\text{op}}(\varphi) \iff \sum_{v' \models \varphi} \pi(v') \odot p$$

(8.15)

As one can see from Equation 8.15, in order to model check the steady-state operator formula, we must compute the steady-state probabilities $\pi_v(v')$ for all states $v$ and $v'$. These probabilities are independent of the initial state if the system is ergodic. Otherwise, the solution is more involved as it necessitate the computation of bottom strongly connected components. It is a standard result [Ste94] that steady state probabilities can be obtained by solving a system of linear equations:

$$\bar{\pi} \cdot \mathbf{Q} = \bar{0} \quad \text{and} \quad \sum_{v' \in A} \pi(v') = 1$$

(8.16)
where \( Q \) is called the *generator matrix* and is obtained via the following equation:

\[
Q(v, v') = \begin{cases} 
R(v, v') & \text{if } v \neq v' \\
- \sum_{v'' \neq v} R(v, v'') & \text{otherwise}
\end{cases} \tag{8.17}
\]

Equation 8.16 can be solved using standard iterative or direct method [Ste94].

Let us now turn to model checking the PATTL formula \( \mathcal{P}_{\text{eq}}(\varphi_1 U \varphi_2) \), for finite values of \( \tau \).

The path measure can be obtained via a technique called uniformization, which is also known as Jensen's method [BHHK00]. This method proceeds to the transformation of the original CTMC (which our PCN is equivalent to in this case) to a *uniformized* discrete-time Markov chain with transition matrix \( P \), yielding an infinite summation to compute the transient probability vector \( \pi_{s,t} \) obtained via:

\[
\pi_{s,t} = \pi_{s,0} \cdot \sum_{i=0}^{\infty} \gamma_i q \cdot t \cdot P^i \tag{8.18}
\]

where \( \gamma_i q \cdot t = e^{-q \cdot t} \cdot (q \cdot t)^i / i! \) denotes the \( i \)th Poisson probability with parameter \( q \cdot t \).

The complexity of PATTL augmented with the steady-state operator model checking has been showed to be linear in the size of the formula, polynomial in the state space, linear in the maximum time threshold in the formula, and linear in the largest entry of the generator matrix \( Q \) [BHHK00].

### 8.7 First Order PCN timed temporal logic (FPATTL)

We can define a first-order PCN timed temporal logic (FPATTL). The syntax of FPATTL is obtained by combining the rules of the propositional PATTL logic with a multi sorted first-order language. That is, in addition to atomic propositions, logical connectives, temporal and probabilistic operators, we now also have *predicates, functions, individual constants* and *individual variables* interpreted over an appropriate domain. Intuitively, the structure on which the propositional language is used is extended in such a way that each state is associated with an interpretation of *local* and *global* symbols. The semantics is obtained via the usual (Tarskian) definition of truth while the validity and satisfaction are defined in the usual way. Here, we present briefly the details of the technical formulation of FPATTL, which extend those introduced in [Eme90, Zha94] for a non-probabilistic linear propositional logic.
In order to define a first order language, we need to define the notion of terms which relies on the concepts of signature and \( \Sigma \)-domain structure defined in Chapter 2. In addition to the notions of signature and \( \Sigma \)-domain, we also need to specify a set of global variables on which the first-order quantifiers will apply, and a set of local variables. In the PCN framework, global variables, \( X_g \), will be represented by parameter variables while local variables, \( X_l \), will be trace variables of the model.

The set of global and local variable will add up to the set of \( S \)-sorted variables \( X = X_l \cup X_g \). The set of terms of sort \( s \in S \) induced by \( \Sigma \) and \( X \), denoted by \( T(\Sigma, X)_s \), is the least set of strings that satisfies one of the following:

- If \( x \in X_g \), then \( x \in T(\Sigma, X)_s \).
- If \( x \in X_l \cap X_g \), then \( \text{pre}(x), x - \tau \in T(\Sigma, X)_s \) for \( \tau > 0 \).
- If \( f \in F \) with type \( \rightarrow s \), then \( f \in T(\Sigma, X)_s \).
- If \( f \in F \) with type \( s^* \rightarrow s \) where \( s^* : I \rightarrow S \), then \( f(T) \in T(\Sigma, X)_s \) where \( T : I \rightarrow T(\Sigma, X) \) with \( T_i \in T(\Sigma, X)_{s_i} \).

Given \( \Sigma = (S, F) \) as a signature, let \( \Phi \) be a set of \( S \)-sorted predicate symbols, such that for each \( p \in \Phi \), the type of \( p \) is a tuple \( s^* : I \rightarrow S \). The syntax of FPATTL can be defined given \( \Sigma \) and \( \Phi \).

**Definition 8.12 (Syntax of FPATTL)**

The basic syntax of FPATTL can be defined as:

\[
\phi ::= \text{true} \mid T^1_s \mid T^2_s \mid p(T) \mid \neg \phi \mid \phi_1 \land \phi_2 \mid \phi_1 \cup \phi_2 \mid \phi_1 \, S \, \phi_2 \mid \varphi_1 \, U \, \varphi_2 \mid \varphi_1 \, U^T \, \varphi_2 \mid \varphi_1 \, S^T \, \varphi_2 \mid \mathcal{P}_{\mathcal{P}}(\phi) \mid \mathcal{P}_{\mathcal{P}}^x
\]

where \( T_s \in T(\Sigma, X)_s \) is a term of sort \( s \), \( p \in \Phi \) is a predicate symbol with type \( s^* : I \rightarrow S \), \( T : I \rightarrow T(\Sigma, X) \) with \( T_i \in T(\Sigma, X)_{s_i} \) and \( x \in X_g \).

For defining the quantifiers in FPATTL, we adopt a rather different approach in that instead of specifying the syntax and semantics for the Universal (\( \forall x \in X_g \)) and Existential (\( \exists x \in X_g \)) quantifiers, we generalize those notions to introduce a *Probabilistic quantifier* \( \mathcal{P}_{\mathcal{P}}^x(\phi) \), \( x \in X_g \), i.e., \( \phi \) is quantified over global variables.
To properly define this quantifier, we need to introduce the notion of prior on the domain of the global variables $x \in X_g$. Therefore, within FPATTL, we will assume the existence of a prior probability distribution $\Gamma_x : A \to [0,1], x \in X_g$, with the restriction that $\sum_{a \in A} \Gamma_x(a) = 1$. Intuitively, when quantifying over a global variable $x$, we assume that $\Gamma_x$ is a well-defined probability distribution (a prior) that assess a probability to each value that $x$ can take within its domain.

A frame of FPATTL over a PCN structure $X$ is a tuple $(T, A, P_X, V)$ where $T$ is a time structure, $A$ is a $\Sigma$-domain structure, $P_X$ is the probability distribution (possibly parameterized) induced by $X$ and $V$ is an interpretation that assigns to each predicate symbol $p \in \Phi$ a subset $V(p)$ of $\times_I A_{s^*}$, given that the type of $p$ is $s^* : I \to S$.

A model of FPATTL is a pair $(F, \sigma)$ where $F = (T, A, P_X, V)$ is a frame and $\sigma = \langle \sigma_l, \sigma_g \rangle$ is a valuation for $X = X_l \cup X_g$, i.e., $\sigma_g : X_g \to A$ and $\sigma_l : X_l \to A^T$, where $A^T$ is the trace space of the system. By extending the valuation $\sigma$ from variables to terms, we have $\sigma : T(F, X) \to A^T$, such that for any $t \in T$:

- $\sigma(x)(t) = \sigma_g(x)$ for any $x \in X_g$,
- $\sigma(x)(t) = \sigma_l(x)(t), \sigma(pre(x))(t) = \sigma_l(x)(pre(t)), \sigma(x \equiv \tau)(t) = \sigma_l(x)(t - \tau)$ for any $x \in X_l$,
- $\sigma(f(t))(t) = f^A(\sigma(T(t)))$ for any $f \in F$.

Definition 8.13 (Semantics of FPATTL) Let $F = (T, A, V)$ be a frame and $\langle F, \sigma \rangle$ be a model of FPATTL. Let $F$ be an FPATTL formula, $\sigma \models_t F$ denotes that $\sigma$ satisfies $F$ at time $t$:

- $\sigma \models_t \text{true} \quad \forall t \in T$.
- $\sigma \models_t T^1_s = T^2_s \quad \text{iff } \forall v_1 \in \sigma(T^1_s), v_2 \in \sigma(T^2_s), v_1(t) = v_2(t)$.
- $\sigma \models_t p(T), p \in \Phi \quad \text{iff } \forall v \in \sigma(T), v(t) \in V(p)$.
- $\sigma \models_t \neg \varphi \quad \text{iff } \sigma \models_t \varphi \quad \text{iff } \sigma \models_t \varphi_1 \land \varphi_2 \quad \text{iff } \sigma \models_t \varphi_1 \text{ and } \sigma \models_t \varphi_2$.
- $\sigma \models_t \varphi_1 \cup \varphi_2 \quad \text{iff } \exists t' > t, \sigma \models_{t'} \varphi_2 \text{ and } \forall t'' \text{, } t < t'' < t', \sigma \models_{t''} \varphi_1$.
- $\sigma \models_t \varphi_1 S \varphi_2 \quad \text{iff } \exists t' < t, \sigma \models_{t'} \varphi_2 \text{ and } \forall t'' \text{, } t' < t'' < t, \sigma \models_{t''} \varphi_1$.
- $\sigma \models_t F\varphi_1 U^\tau \varphi_2 \quad \text{iff } \exists t' \in T_{t+\tau}, \sigma \models_{t'} \varphi_2 \text{ and } \forall t'' \text{, } t < t'' < t', \sigma \models_{t''} \varphi_1$. 

166
\[ \sigma \models t \varphi_1 \supset \varphi_2 \quad \text{iff} \quad \exists t' \in T, \varphi \models t' \text{ and } \forall t'' \text{ such that } t'' < t' < t, \sigma \models t'' \varphi_1. \]

\[ \sigma \models t \mathcal{P}_p(\psi) \quad \text{iff} \quad \mu^X (\{v \in \sigma | v \models t \psi\}) \cap p. \]

\[ \sigma \models t \mathcal{P}_p^x \varphi, x \in X_s \cap X_g \quad \text{iff there is a value } a \text{ in } A_s, \text{ such that we have} \]
\[ (\sum_{a \in A_s} \Gamma_x(a) \cdot (\sigma \models t \varphi[a/x])) \cap p, \]
\[ \text{where } \varphi[a/x] \text{ stands for substitution of } x \text{ in } \varphi \text{ by } a. \]

Note that the usual quantifiers (\(\exists x, \forall x\)) are represented by the respective special instances of our probabilistic operator: \(\exists x \equiv \mathcal{P}_0^x\) and \(\forall x \equiv \mathcal{P}_1^x\). Moreover, various logical connectives, temporal and real-time operators can be defined as for PATTL.

As it is usually the case, we will use \(\sigma \models F\) to denote that \(\sigma\) satisfies \(F\) initially, i.e., \(\sigma \models_0 F\). \(F\) is valid over a frame \(F\), iff for any model \(\langle F, \sigma \rangle\), \(\sigma \models F\). \(F\) is valid, iff any frame \(F\), \(F\) is valid over \(F\). \(F\) is satisfiable over a frame \(F\), iff for some model \(\langle F, \sigma \rangle\), \(\sigma \models F\). \(F\) is satisfiable, iff for some frame \(F\), \(F\) is satisfiable over \(F\).

**Example 8.3** Consider the property, as introduced before, of an elevator system to deliver any new passenger to its destination within a certain amount of time \(\tau = 30\), with at least probability \(p = 0.95\): \(\Box(E \rightarrow \mathcal{P}_{0.95}^1(\text{true} \cup 30 D))\). Obviously, the satisfaction of this formula depends on the rate of arrivals of new passenger, on the number of floors of the building being deserved by the elevator and on the velocity of the elevator. Let \(R\) be the domain for the parameter of the probability distribution modeling the rate \(r\) i.e., \(r \in X_g\), and \(r\) is a global variable in the PCN model of the passengers’ arrival. Then the FPATTL formula \(\mathcal{P}_{p}^x (\Box(E \rightarrow \mathcal{P}_{p}^1(\text{true} \cup D)))\) will be true if the measure of all possible parameter values in \(R\) for which the formula is true is greater or equal to the threshold \(p\).

The verification of first-order formulae in FPATTL is closely analogous to that of PATTL formulae with the exception that one has to be careful when defining the prior probability distribution over uncountable domains. The technical issues arising in this case will not be addressed in this dissertation.

Another issue that is important to note is the notion of open state specification. Zhang ([Zha94]) showed that the openness of state formulae (formulae without temporal and real-time operators) was important for requirements specification. For example, consider the safety constraint \(\Box \neg \varphi(x)\). This formula stipulates that the system, in order to satisfy the behavioural constraint requirement, should
never satisfy the predicate $\varphi(x)$, no matter the value of $x$. However, if the predicate $\varphi$ is open, it means that $\neg \varphi$ is closed and hence an undefined value of $x$ would satisfy the property. This can be extremely problematic, especially for critical constraints on the behaviour such as safety. For more details on this topic, the reader is referred to the aforementioned reference.
Chapter 9

Control Synthesis

9.1 Introduction

In this chapter, we will briefly review the concept of control synthesis within the PCN framework. Similarly to the verification problem discussed earlier in this dissertation, the task of control synthesis is extremely complex. In fact, there does not exist a unique algorithm that can be used to synthesize controllers in all classes of systems. Hence, various methods have been developed over the years, each of which are specifically designed to handle a certain type of control problem.

Due to the complexity of the problem, the field of control synthesis is a wide area that includes among others, optimal control, stochastic control, robust control and also planning. We do not intend to cover all those areas here, but rather we wish to introduce the reader to specific techniques that are especially well suited to synthesize controllers when in the presence of a PCN model.

As described earlier, we view a robotic system as a combination of a body, a controller and an environment (Figure 1.1). We defined the behavior of such a robotic system as the set of observable robot/environment traces of the system. Moreover, we introduced the notion of requirements specification as the subset of all possible traces which satisfy a given property. Therefore, within the PCN framework, the problem of control synthesis can be described as follows: given a requirements specification $\mathcal{R}$, the model of the body $BODY$ and the model of the environment $ENVIRONMENT$, synthesize a model of the controller $CONTROLLER$, such that the behaviour of the resulting system satisfies the requirement, i.e.,

$$[X = BODY(U,Y), U = CONTROLLER(X,Y), Y = ENVIRONMENT(X)] \models \mathcal{R}.$$
Historically, planning and control have been studied as different problems. The planning problem [DW91] is defined as using a model to formulate sequences of actions to achieve a certain goal. The control problem [DW91] is considered as finding a policy to achieve a goal or minimizing a functional. Planning is normally restricted to symbolic domains in discrete time; while control is often for numerical domains, particularly $n$-dimensional Euclidean spaces, in either discrete or continuous time, both of which can be handled under the umbrella of the PCN framework. The solution to a planning problem (traditionally) is a trace (sequence) of inputs to a system for approaching a final goal; the solution to a control problem (closed-loop control) is a transduction from the sensor traces to the command traces for minimizing a required functional, such as time, energy, cost for approaching a goal. Common techniques for planning include search algorithms and theorem proving while calculus of variations and optimization have been developed for control. It is to be noted that even though in general the notions of control and planning are often considered as different problems, they can be seen as specific instances of our control synthesis formalism.

Examples of typical planning and control problems include respectively, approaching a final goal and minimizing a global function over time (for example, energy). Both problems can be seen as setting constraints over the possible traces of the system, which clearly relates the notion of control with that of verification.

There are many types of control synthesis problems, depending on the information that is available to the decision maker. These types can be classified in three main categories:

1. **Open-loop**: only the initial state of the system $X_{t_0}$ is known.

2. **Feedback**: both the initial state $X_{t_0}$ and the current state $X_t$ are known.

3. **Closed-loop**: the whole trajectory of the system is known, i.e., $\{X_s\}, t_0 \leq s \leq t$.

It is clear that open-loop controllers form a subset of feedback controllers while feedback controllers are a subset of closed-loop controllers. However, for Markovian systems, feedback control and closed-loop control are in fact equivalent since the state of the system is completely determined by the previous state. In this chapter, we will restrict ourselves to feedback control as most systems of interest possess the Markovian property.

Before formally introducing the notions of stochastic and robust control, let us enumerate some standard definitions of stability in the sense of Lyapunov.
- **Equilibrium point:** A state $x_e$ is an equilibrium point (state) of the system $\dot{x} = f(x, u, t)$ if $x(t) = x_e$ then the system stays in $x_e$ for all time. This essentially means that $f(x_e, u, t_0) = 0, \forall t_0 \geq t$.

- **Stability:** This property denotes a system that remains close to an equilibrium given that the initial condition was reasonably close to the equilibrium in the first place. Formally, the equilibrium point $x_e = 0$ is said to be stable if, for any $\epsilon > 0$, there exists a constant $\delta(t_0, \epsilon) > 0$ such

$$\|x_{t_0}\| < \delta(t_0, \epsilon) \implies \|x(t)\| \leq \epsilon, \forall t \geq t_0.$$

Otherwise the system is deemed unstable. Stability is a very important concept since trying to control an unstable system is useless.

- **Attraction:** The equilibrium point $x_e = 0$ is said to be attractive at time $t_0$ if, for some $\delta > 0$ and each $\epsilon > 0$, there exists a finite time interval $T(t_0, \delta, \epsilon)$ such that

$$\|x_{t_0}\| < \delta \implies \|x(t)\| \leq \epsilon, \forall t \geq t_0 + T(t_0, \delta, \epsilon).$$

- **Asymptotic Stability:** The equilibrium point $x_e = 0$ is asymptotically stable if (1) it is stable at time $t_0$, and (2) it is attractive, or equivalently, there exists $\delta > 0$ such that

$$\|x_{t_0}\| < \delta \implies \|x(t)\| \to x_e \text{ as } t \to \infty.$$

- **Exponential Attraction and Stability:** $x_e = 0$ is exponentially attractive at time $t_0$ if, for some $\delta > 0$, there exist two strictly positive numbers $\alpha(\delta)$ and $\beta$ such that

$$\|x_{t_0}\| < \delta \implies \|x(t)\| \leq \alpha(\delta) e^{-\beta(t-t_0)}.$$ 

The equilibrium point is said to be exponentially stable if, for some $\delta > 0$, there exists constants $\alpha > 0$ and $\beta > 0$ such that

$$\|x_{t_0}\| < \delta \implies \|x(t)\| \leq \alpha \|x_{t_0}\| e^{-\beta(t-t_0)}.$$
Figure 9.1: Comparing a) stability, b) attraction, and c) asymptotic stability.

The above definitions are presented for systems whose equilibrium points are located at the origin. However, they can easily be extended to systems with a known but nonzero equilibrium. The concepts of uniform stability, uniform attraction and uniform asymptotic stability imply that stability and performance properties of many systems are independent of the initial time $t_0$. Figure 9.1 compares the concepts of stability, attraction and asymptotic stability for a two-dimensional system.

9.2 Stochastic and Robust Control

As mentioned above, the field of control is vast and has evolved considerably since its origin which dates all the way back to 300 BC with the preoccupation of the Greeks and Arabs with keeping accurate track of time. For an interesting introduction to the origin of automatic control, the reader should consult §1: Introduction to Modern Control Theory of [Lew92].

Obviously, we cannot cover most of the control theory field in this dissertation. Therefore, we will simply provide a brief introduction to two areas of control theory: stochastic control and robust control, with an emphasis on robust control. We will then illustrate the use of robust control on a simple package delivery robotic system.

In stochastic control the uncertainties in the system are modeled as probability distributions. One desires to combine these distributions to yield the control law. Stochastic control deals essentially with the expected value of control: controlling the system on average, based on the stochastic nature of the uncertainty. However, large deviations can occur and thus move the system away from its (average case) optimum. Hence, this type of control might not be acceptable for embedded con-
control systems that have safety implications: when we want to have probability one that one event will not occur.

In general, stochastic control is interested in minimizing a cost function. The most common stochastic control techniques include Linear Quadratic Gaussian (LQG), Quadratic Stability and Hamilton-Jacobi-Isaacs Equations. An introduction to stochastic control can be found in [Lew86].

Robust control involves a method of measuring the performance changes of a control system in the presence of changing system parameters. The main concern of robust control is uncertainty and how the control system can deal with the presence of uncertainty. Unlike stochastic control, robust control seeks to bound the uncertainty rather than express it in the form of a distribution. Given a bound on the uncertainty, the control can deliver results that meet the control system requirements in all cases. This can be seen as worst-case control. Note that some performance may be lost to guarantee the system meets some requirements. It has been shown that controller arising from robust control can be more conservative than stochastic controllers. However, when one cannot accept the system entering certain states (as in safety critical systems), loss of performance may be a small price to pay.

The field of robust control has yielded many different techniques beyond the scope of this dissertation. Here, we simply attempt to catalog the major ones and briefly describe the basic concepts behind each techniques. The reader is referred to the specific citations for more detail on a particular technique.

- **Adaptive Control**: An adaptive control system sets up an observer for each significant state variable. The system can adjust each observer to account for time varying parameters of the system. The output is to be brought closer to a desired input while at the same time the systems continues to learn about changes in the system parameters. This method may suffer from convergence problems. [K.J96].

- **$H^2$ and $H^\infty$**: Hankel norms are used to measure control system properties. A norm is an abstraction of the concept of length. Both these techniques were originally developed for the frequency domain but have been recently extended for the state space domain [PUS00]. $H^2$ control seeks to bound the power gain of the system while $H^\infty$ control seeks to bound the energy gain of the system. Gains in power or energy in systems indicate operation of the
system near a pole in a transfer function which is an unstable situation. For more on these
techniques, consult [Cha96].

• **Parameter Estimation**: This method establishes boundaries in the frequency domain that
cannot be crossed to maintain stability. These boundaries are evaluated by given uncertainty
vectors. This technique is similar in essence to the root locus method [Eva54] in that it is also
a graphical method which observes the movement of the system. A detailed treatment of the
parameter estimation techniques can be found in [Ack93].

• **Lyapunov**: This is possibly the only universal technique for assessing nonlinear systems.
Lyapunov techniques focus on stability. One constructs Lyapunov functions, which can be
seen as energy functions that model the behavior of the system. Then one evaluates these
functions along the trajectory to ensure that the first derivative is always dissipative in energy.
A gain in energy means that the system is near a pole and will therefore be unstable. Qu de­
scribes these methods in detail [Qu98] and focuses on the Lyapunov first and second methods
for analyzing the stability of systems.

### 9.2.1 Robust Control of Package Delivery Robot

In this section, we present an example illustrating the use of control synthesis on a system, encom­
passing multiple sources of uncertainty, acting on a continuous time structure. Suppose that the
system to be controlled consists of a robot pushing a package to a desired location as shown in
Figure 9.2. The arms of the robotic agent are built on a spring structure. The spring constant $k$ has a
nominal value of $k_0 = 1.25$, but may vary and is considered uncertain: $k \in [0.5, 2.0]$. Furthermore,
there is an external disturbance, $\omega(t)$, affecting the dynamics of the package as it is pushed by the
agent. This disturbance is modeled as a Wiener process with identity covariance. For simplicity,
we assume that both the robot and the package have the same mass $m_r = m_p = 1$. Due to the
uncertain nature of the spring constant along with the stochastic nature of the external disturbance
on the package, we model the system as a *stochastic uncertain system* (see [PUS00] for a thorough
description of stochastic uncertain systems). Then the system can be described by the following set
of equations:
Figure 9.2: Package delivery robotic system.

\[
\dot{x} = (A + F(k(t) - k_0)S)x + B_1u + b_2\omega(t);
\]

\[
z = Cx + Du
\]

where \( z \) is called the uncertainty output of the system, \( x = [x_1 \ x_2 \ \dot{x}_1 \ \dot{x}_2]' \in \mathbb{R}^4 \) is the state space, \( u \) is the control force, and

\[
A = \begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 1 \\
-k_0 & k_0 & 0 & 0 \\
k_0 & -k_0 & 0 & 0 \\
\end{bmatrix}; \quad B_1 = \begin{bmatrix}
0 \\
0 \\
0 \\
1 \\
\end{bmatrix}; \quad B_2 = \begin{bmatrix}
0 \\
0 \\
1 \\
0 \\
\end{bmatrix}; \\
C = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}; \quad D = \begin{bmatrix}
0 \\
0 \\
1 \\
\end{bmatrix};
\]
\[ F = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 1 \end{bmatrix} \quad \text{and} \quad S = \begin{bmatrix} 1 & -1 & 0 & 0 \end{bmatrix} . \]

It has been shown that a controller corresponding to the nominal value \( k = k_0 \) can be found that guarantees an \( H^\infty \) norm bound with \( \gamma = 2.5 \) for a closed-loop nominal transfer function [PUS00]. However, this norm bound holds only for the aforementioned nominal value of \( k \) and is not robust for variations in \( k \).

Suppose instead that a satisfactory controller must guarantee a disturbance attenuation bounds \( \gamma = 2 \) robustly with respect to the variations in the value of \( k \). Define the following: \( \Delta(t) = k(t) - k_0 \). Assume that \( \Delta(t) \) is a Gaussian white noise process with zero mean and variance \( \sigma^2 \).

It is easy to choose the value for the parameter \( \sigma \) such that \( k \) will fall within the bounds \([0.5, 2.0]\) with high probability. For instance, a selection of \( \sigma = 0.25 \) with yield \( P(|k(t) - k_0| \leq 0.75) \geq 0.997 \). Obviously, this probability will increase as \( \sigma^2 \to 0 \). Applying the result of Theorem 3 from [Ugr98], which involves solving algebraic Riccati equations that can be solved by homotopy methods [RHP93b], along with an infinite sequence of Lyapunov equations [Guo01], we obtain the controller \( u = Kx \), with \( K = [0.8547 - 3.7458 - 3.3103 - 3.1986] \). It is possible to show that with this controller, the robotic package delivery system is exponentially stable.

Figure 9.3 shows one realization of the system. In this example, the package needs to be moved by one unit forward. We can observe that the agent overshoot the target location a little before stabilizing around the desired target.

### 9.3 Planning under uncertainty

Coined as planning under uncertainty, decision-theoretic planning (DTP) has recently drawn a considerable amount of attention among the AI community. Basically, the problems which are of interest to decision-theoretic planners are those involving systems whose dynamics can be modeled as stochastic processes and where an agent, acting as a decision maker, can influence the system’s behavior by performing (uncertain) actions. Resulting from the Markov property, the current state of the system and the choice of the action by the agent jointly determine a probability distribution over the possible next states. It is usually assumed that systems evolve in stages, where actions cause
transitions from stage $t$ to $t + 1$. This progression through stages is analogous to the passage in time, with one clock for the whole system, if one assumes that every action takes unit time to complete. In general, it is also assumed that transition probabilities are stationary, although the results can easily be extended to the non-stationary case. The agent is assumed to prefer to be in a certain subset of the system’s states, whose elements are often called the goal states. Therefore, one would like to construct a plan of action which dictates the action that is most likely to bring the agent to these goal states. Such a plan is called a policy and finding it is the goal of DTP. The representation used as an underlying model for such decision-theoretic problems is very often a Markov Decision Process.

The reader should note that it is often the case that the agent does not know the exact state of the world. Hence, when it needs to choose an optimal action, the agent is forced to make decisions based on a probabilistic estimate of the current state of the system. Such problems have been modeled with a Partially Observable Markov Decision Process (POMDP), which can be seen as a fully observable MDP with the addition of an infinite state space consisting of probability distributions over the state,
each distribution representing the agent's belief of where it is at any point in time. For an in depth look at the theory of MDPs or for a survey on decision-theoretic planning (including POMDPs), the reader is referred to [Put94] and [BDH99] respectively.

In the following section, we will briefly introduce the notion of (PO)MDPs. We will carry on to present how (PO)MDPs can be seen as a simple case of a PCN model and how the decision-theoretic algorithms can be used to generate a PCN controller.

9.4 Introduction to Markov Decision Processes

A Markov decision process (or sometimes referred to as a fully observable MDP) is defined by the tuple $(S, A, P, R)$, where $S$ is a finite set of states of the system, and where states are defined as a description (more or less precise) of the system at any point in time. In a MDP, these states can be exactly identified by the agent, i.e., at any given time the agent knows exactly which state it is in. $A$ is a finite set of actions from which the agent can choose; $P$ is the state transition model of the system which is a function mapping from elements of $S \times A$ into discrete probability distributions over $S$; and $R$ is a stationary reward function mapping from $S \times A$ to $\mathbb{R}$. $R(s,a)$ specifies the immediate reward gained by the agent for taking action $a$ in state $s$.

Actions induce stochastic transitions, with $P(s,a,t)$ denoting the probability with which state $t$ is reached when, at the previous time step, action $a$ is performed at state $s$. It is to be noted that the transitions of the model specify the resulting next state using only the state and action at the previous time step. This therefore assumes that the next state is solely determined by the current state and the current action and corresponds to the Markov assumption discussed earlier. It is worth mentioning that not all systems are Markovian in nature. The Markov assumption is merely a property of a particular model of that system, not of the system itself. However, one should note that the Markovian assumption is not too restrictive, since any non-Markovian model of a system can be converted to an equivalent Markov model. In the field of control theory, this conversion is referred to as the conversion to state form [Lue79].

A stationary policy $\pi : S \rightarrow A$ describes a particular, time independent, course of action to be adopted by an agent, with $\pi(s)$ denoting the action to be taken in state $s$. It is often assumed that the agent acts indefinitely (an infinite horizon) but the finite horizon case has also been studied extensively. In the finite-horizon case however, the optimal policy is typically non-stationary: the
agent's choice of action on the last step of his life will generally be very different than when it has a long life ahead of it. We will, in this short presentation of the MDP framework, assume infinite horizon, unless explicitly stated.

A possible way to assess the quality of different policies is to adopt an expected total discounted reward as the optimality criterion wherein future rewards are discounted at a rate $0 \leq \beta < 1$, and the value of a policy is given by the expected total discounted reward accrued. The expected value $V_\pi(s)$ of a policy $\pi$ at a given state $s$ satisfies [Put94]:

$$V_\pi(s) = R(s, \pi(s)) + \beta \sum_{t \in \mathcal{S}} P(s, \pi(s), t) \cdot V_\pi(t)$$  \hspace{1cm} (9.1)

A policy $\pi$ is said optimal if $V_\pi \geq V_{\pi'}$ for all $s \in \mathcal{S}$ and policies $\pi'$. The optimal value function $V^*$ is the value of any optimal policy.

There exists many iterative algorithms for constructing optimal policies; the most popular ones are value iteration and policy iteration. For the purpose of this discussion, we will only briefly present the simplest one: value iteration.

**Value iteration** [Bel57] is a simple iterative approximation algorithm for constructing optimal policies. It proceeds by constructing a series of $n$-stage-to-go value functions $V^n$. Setting $V^0 = R$, we define

$$V^{n+1}(s) = \max_{a \in \mathcal{A}} \left\{ R(s, a) + \beta \sum_{t \in \mathcal{S}} Pr(s, a, t) \cdot V^n(t) \right\}$$  \hspace{1cm} (9.2)

The sequence of value functions $V^n$ produced by value iteration converges linearly to the optimal value function $V^*$. For some finite $n$, the actions that maximize Equation 9.2 form an optimal policy, and $V^n$ approximates its value. A commonly used stopping criterion specifies termination of the iteration procedure when

$$\|V^{n+1} - V^n\| < \frac{\epsilon(1 - \beta)}{2\beta}$$  \hspace{1cm} (9.3)

(where $\|X\| = \max\{|x| : x \in X\}$ denotes the supremum norm). This ensures that the resulting value function $V^{n+1}$ is within $\frac{\epsilon}{2}$ of the optimal function $V^*$ at any state, and that the resulting policy is $\epsilon$-optimal [Put94].
In the MDP setting, the agent behaves according to a policy that is computed off-line, typically by dynamic programming techniques such as those mentioned in Section 9.4. Once this policy is available, the agent simply transitions from one state to another, choosing at each time step the action that is specified by the policy.

This is essentially a control problem where the policy represents a controller and the dynamics of the MDP (transition probabilities) denote the agent’s behavior. In this view, one can see obvious similarities between a PCN model and a MDP. Figure 9.4 graphically represents this similarity. In the PCN framework one describes a robotic system as the coupling of a plant (robot’s body), an environment and a controller. Applying this view to a MDP model, one can see that the MDP dynamics represent both the environment and the plant, while the policy represents the controller. In fact, in an MDP, the robot’s body is included in the general (non-modular) description of the environment. This lack of modularity can significantly complicate the task of system designers. Indeed, modularity and hierarchy are two essential properties of a model for robotic systems, which PCN possesses.

The reason why MDPs are so popular within the AI community is not mainly because of the modeling capabilities of the framework, but rather the algorithms with which one can compute optimal policies. As mentioned earlier, policies can be viewed as controllers; hence computing a policy can be seen as control synthesis. It would be extremely valuable to be able to merge the modeling simplicity and power of the PCN framework with the control synthesis capabilities of MDP. In this section, we will show that for a subclass of PCN models, which we call synchfin-PCN, there exists a one-to-one correspondence between the class of all MDPs and synchfin-PCNs. We
will also show, for the synchfin-PCN class of models, how we can synthesize an optimal controller by taking advantage of the dynamic programming algorithms for MDPs.

Figure 9.5 sketches out the algorithm for performing control synthesis on a synchfin-PCN model. The first step consists of translating the synchfin-PCN model into a MDP. Secondly, one performs off-line value or policy iteration on the resulting MDP to obtain a policy. Finally, the policy is converted into a controller for the initial synchfin-PCN model, thus synthesizing a controller.

Before formally defining the synchfin-PCN class of models and the control synthesis algorithm, let us present an informal discussion on the correspondence between a PCN and a MDP model.

### 9.5.1 MDP to PCN Conversion

Within the MDP framework, one can use an explicit, or sometimes referred to as extensional, representation for the state space. In such a representation, states are enumerated explicitly, hence the designer has to describe the transition probabilities for every pair of states within the system. In this case, the transformation of an MDP into a PCN model is trivial: the state variable and the set of transition probability tables (one per action) can be seen as a single location with a generator, parameterized on the selected action and the current state of the system, which generates the next
Figure 9.6: PCN model of an extensional MDP

(uncertain) state. Figure 9.6 displays a PCN model of a generic extensional MDP.

However, for a problem with a state space \( S \) such that \(|S| = N\) is large, the task of specifying the transition probabilities can be overwhelming \( (O(N^2)) \) and quite impractical. A solution to this problem is to use an intensional (or factored) representation rather than an extensional one. An intensional representation is obtained by specifying a set of features that describes, at an acceptable level of detail, the state of a given dynamical system. Each feature takes a finite number of values (usually quite small) and an assignment of values to every feature of the system completely defines a state. Hence, the state space can be described as the cross product of all the features, and in general, it grows exponentially in the number of features used to describe the system.\(^1\) Another important advantage of adopting a factored representation for the state space within the MDP framework is that it allows one to also use a factored representation of actions, rewards and other components of an MDP. In such a representation, one describes the effects of an action on specific features rather than on the entire state of the system, often leading to a considerably more compact representation.

Bayesian networks\(^2\) (BN) [Pea88] are graphical models used to represent causal and probabilistic processes. In the recent years, BNs have become the tool of choice for probabilistic and statistical modeling. Moreover, they have become the norm for representing probability distributions in factored form as not only do BNs provide convenient means of expressing complex assumptions and relationships between variables but they also greatly facilitate an economical representation of joint

\(^1\)Note that a PCN model is by definition a factored representation as the state space corresponds to the cross product of the domains of the locations of the model.

\(^2\)In AI, Bayesian networks are also referred to as belief nets.
probability functions.

Formally, a Bayesian network is represented by a directed acyclic graph (DAG). Each element in the set of vertices denotes a set of random variables and edges between two vertices represent a direct probabilistic dependency between the two random variables. Obviously the absence of edges between vertices also reflects implicit independencies among the variables. Once the structure of the graph is specified, one must then quantify the network by specifying, for each variable in the graph, a conditional probability table (CPT) where a probability is given for each variable conditioned on all possible values of its immediate parents in the BN. When a variable does not have any parents, an unconditional distribution, also known as a marginal distribution, is specified for the parentless variable. Not only is the structure of the BN displaying the independence assumptions of the variables, but it can be shown that the network defines a unique joint distribution over the variables constituting the global state space of the system. For a thorough introduction to the theory of Bayesian networks, we refer the reader to [Pea88].

Temporal Bayesian networks (also called dynamic Bayesian networks) are special cases of Bayesian networks where the vertices represent features of the systems at different time points and edges denote dependencies over time. For Markovian systems only 2 time points are of importance: $t$ and $t + 1$. Hence, temporal Bayesian networks for such systems are called two-stage temporal Bayes nets (2TBN). In a 2TBN, the set of features (variables) is partitioned into 2 sets: the features at time $t$ and the features at time $t + 1$. We call diachronic arcs the arcs displaying dependencies being variables at time $t$ and $t + 1$ while synchronic arcs denote dependencies between variables at time $t + 1$. For a thorough discussion on dynamics Bayesian networks, the reader is referred to [Mur02].

Similarly to the extensional representation, the conversion of an intensional MDP into a PCN model is rather simple. Given the conditional probability tables from the MDP description, one can build exactly one generator per feature (variable) where the inputs of the generator are the selected action and the parents of the variable, as specified by the two-stage temporal Bayes Net (2TBN). By acting this way, one takes advantage of the factored representation of the conditional independence assumption. It allows one to build generators that are simpler, since the size of their set of input variables is much smaller than the total number of variables in the system. Figure 9.7 shows a general case for a PCN obtained from converting a factored MDP. Note that to avoid algebraic loops
in the PCN model, the use of unit delays is required. This is only a side effect of the fact that the probability distribution of a variable at time $t$ is parameterized on the value of the same variable at time $t-1$. Therefore, it is easy to conclude that unit delays will appear for every diachronic arc in the 2TBN representation of the MDP.

In the event where the action effect on variable $V_i^{t+1}$ is correlated with variable $V_j^{t+1}$, as represented by a synchronic arc in the 2TBN, we can represent this in a PCN by simply making location $V_j$ an input of the probabilistic transduction of variable $V_i$, without the use of a unit delay. This parameterizes the distribution of $V_i$ on not only the values of its set of parents at time $t$ but also on the value of $V_j$ at time $t+1$. The independence of the variables at time $t$ given the values of the parents at time $t-1$ no longer holds when in the presence of synchronic arcs. The conventional dynamic programming techniques, as presented in the previous section, do not work with arbitrary Bayes Net action description (ones with both diachronic and synchronic arcs). Therefore, in order to be able to perform control synthesis, we might have to require that one only specifies actions without correlation. However, this is not desirable as it would seriously diminish the number of systems that can be represented. Indeed, correlated effects are common in most systems so it is essential that we allow them in our models. More specifically, correlated effects will arise very commonly in the
PCN framework, thus the inability to solve MDPs with synchronic arcs would render our control synthesis algorithm impractical as it would only apply to a very small class of PCN models.

Techniques to alleviate this problem have been suggested: one consists of clustering all variables affected by the correlation into a compound variable with size exponential in the number of variables in the cluster (see [BDH99] for more detail). However, this technique not only requires a radical transformation of the problem (thus we would no longer be able to use the original state variables, which could significantly complicate the conversion of the optimal policy into a PCN controller) but can also cause a blowup in the size of the state space. Another technique, proposed in [Lit97], suggests the transformation of the action representations into a STRIPS representation. However, this solution once again necessitates a drastic transformation of the problem which, as previously discussed, is not desirable in the context of control synthesis for PCN. These two methods, while solving the problem of synchronic arcs, are not useful for our purpose. Remember that we are trying to transform a PCN model into a MDP so that we can perform value or policy iteration to obtain an optimal policy. Once this policy is obtained, we wish to convert it into a PCN controller transduction. Therefore, if in order to solve the MDP we have to dramatically transform its state space, we might not be able to convert the policy into a PCN controller relevant to the original state space. It is important that we maintain the same structure so that the conversion of the optimal policy into a PCN controller remains possible.

Fortunately, [Bou97] improved the structured policy iteration algorithm of [BDG95] so that it would not be hindered by synchronic arcs. This generalized structured policy iteration algorithm is important here as it conserves the original problem description when performing decision theoretic regression in the presence of synchronic arcs. Therefore, based on this result, we can assume that synchronic arcs do not represent a limitation in the control synthesis procedure that we are about to describe.

Making an extensional representation intensional

As we have just seen, an extensional representation of a system can be very impractical to deal with, both in terms of the size of the transition matrix and the lack of information provided by that representation. System features in an intensional representation provide the designer with information on the local behavior of the system. Extensional representations only display global behavior.
One could then ask if given an extensional representation, can one automatically decompose it into a potentially simpler intensional representation? That is, are there $n$ independent variables whose joint distribution is the same as the state transition probability distribution?

Using a set of independent variables that represent the state space can reduce considerably the amount of work the designer needs to perform. Since the state space is the cross product of the variables that constitute it, far fewer probabilities need to be specified in the system. Furthermore, an intensional representation of the system can provide some interesting insight on the behavior of the system, thus simplifying the troubleshooting of the system when need be.

Obviously, not every state transition probability distribution can be decomposed into $n$ independent marginal distributions; for example, if the number of states in the state space, $N$, is a large prime number. Then by the prime factorization theorem, one will not be able to represent the state space with independent variables. Indeed, other than by using one single variable with $N$ distinct values, one would have to use a representation that would create more states than in the state space. For example, a state space with 5 states cannot be exactly represented by any combination of independent variables. The closest one can get is to represent the state space with one binary variable and one three-value variable, thus yielding 6 states. Therefore, one extra state would be artificially created, a state which has no equivalent in the state space representation. One could associate a transition probability of 0 to the values of the variables associated with this state, such that the transition probability to this value, from any other value is 0 and the transition from this value to itself is 1. This technique would maintain the existence of a transition probability distribution but would introduce problems when trying to solve the system of equations. Therefore, to solve this problem, one needs to introduce dependence between the variables, and the hope is that the dependency structure will be simple to extract. More work is currently being done to find an algorithmic solution to this problem.

When $N$ is not a prime number, it is possible to exactly represent the state space with a combination of variables; and a simple test which provides information about the existence of marginal distribution of those variables can be performed. The test simply consists of assigning each state to a set of assignments to the $n$ variables for which we desire to generate the marginals. Under the assumption of independence, we obtain a set of nonlinear equations ($2^n$ for binary variables) of the type $P(S_i|S_i) = P(V_1|V_1) \cdots P(V_n|V_n)$ where the set of probabilities $P(S_i|S_i)$ is given in
the extensional representation. Furthermore, we have the following constraints: each row of each marginal distribution must be a probability distribution (sum to 1) and each individual probability must be between 0 and 1. One is thus left with a nonlinear programming problem to resolve. However, it is possible to avoid dealing with nonlinearity by simply using a log-transformation on the equalities and the range constraints. This transformation appears not to change the situation much since the constraints on the rows of the marginals then becomes nonlinear in the log of the probabilities. But we can ignore those constraints, solve the now linear problem, and once a solution is reached, simply normalize the results, hence producing a probability distribution.

We can represent this linear programming problem with a linear system of the form $Ax = B$. With such a system, three possibilities exist for the solution $x$:

1. No solution exists
2. Exactly one solution exists
3. Infinitely many solutions exist

The following table summarizes the solvability of a system of equations in terms of the rank of matrices $A$ and $B$:

<table>
<thead>
<tr>
<th>The system $Ax = b$ has ...</th>
<th>when ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>no solution</td>
<td>$\text{rank}(A) &lt; \text{rank}([A\bar{b}])$</td>
</tr>
<tr>
<td>one solution</td>
<td>$\text{rank}(A) = \text{rank}([A\bar{b}]) = N$</td>
</tr>
<tr>
<td>infinite number of solutions</td>
<td>$\text{rank}(A) = \text{rank}([A\bar{b}]) &lt; N$</td>
</tr>
</tbody>
</table>

If the ranks are such that there are one or more solutions, then the existence of $n$ independent variables is guaranteed and one simply needs to solve to obtain the marginal distribution. When an infinite amount of solutions can be found, one needs to decide on which solution to choose. An algorithm based on the linear interior point solver [FGW02], which is a variant of Mehrotra's predictor-corrector algorithm (a primal-dual interior-point method) [Meh91] can be used to minimize a function defined over the set of all marginals. The sum of all marginals appears to be a suitable heuristic for that function.

Let us present a simple example where we show how one can apply the test:
Example 9.1 (Extensional to intensional representation conversion) Let us define a state space as $S = \{S_1, S_2, S_3, S_4\}$, with its transition probabilities defined as follows:

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>0.18</td>
<td>0.12</td>
<td>0.42</td>
<td>0.28</td>
</tr>
<tr>
<td>$S_2$</td>
<td>0.255</td>
<td>0.045</td>
<td>0.595</td>
<td>0.105</td>
</tr>
<tr>
<td>$S_3$</td>
<td>0.3</td>
<td>0.2</td>
<td>0.3</td>
<td>0.2</td>
</tr>
<tr>
<td>$S_4$</td>
<td>0.425</td>
<td>0.075</td>
<td>0.425</td>
<td>0.075</td>
</tr>
</tbody>
</table>

Let us assume that we want to model the state space with $n = 2$ binary variables $(V_1, V_2)$. Let us represent the state space in the following way:

- $S_1 = V_1 \land V_2$
- $S_2 = V_1 \land \overline{V}_2$
- $S_3 = \overline{V}_1 \land V_2$
- $S_4 = \overline{V}_1 \land \overline{V}_2$

The marginals probability distributions for the variables (e.g. $P(V_1|V_1)$) are unknown: our goal is to find the transition probabilities of each variable. If the variables were to be independent, one could find out their marginals by simply solving, for the $(8 = 2 \times 2^2)$ different marginals $P(V_1|V_1)$, $P(\overline{V}_1|V_2)$, $P(V_1|V_1)$, $P(\overline{V}_2|V_2)$, $P(V_1|\overline{V}_1)$, $P(\overline{V}_2|\overline{V}_2)$, the following systems of equations:

\[
P(S_1|S_1) = P(V_1 \land V_2|V_1 \land V_2) = P(V_1|V_1 \land V_2) \cdot P(V_2|V_1 \land V_2) = P(V_1|V_1) \cdot P(V_2|V_2)
\]

\[
P(S_2|S_1) = P(V_1 \land \overline{V}_2|V_1 \land V_2) = P(V_1|V_1) \cdot P(\overline{V}_2|V_2)
\]

\[\vdots\]

\[
P(S_4|S_4) = P(\overline{V}_1 \land \overline{V}_2|\overline{V}_1 \land \overline{V}_2) = P(\overline{V}_1|\overline{V}_1) \cdot P(\overline{V}_2|\overline{V}_2)
\]

subject to the set of constraints:
By using the log transform, Equations 9.4 become linear and Constraints 9.6 now restrict the log of the marginals to a non-positive number. By ignoring Constraints 9.5, we can then solve this linear system, which has an infinite number of solutions. The solution that minimizes the sum of all marginal probabilities, along with the normalized values of the solution are displayed in the table below. The column Real Values represents the values with which the initial state transition probabilities were computed (reverse engineering). Hence, one can now model the state space system with 2 binary variables, allowing the designer to only specify 8 probabilities instead of 16 (Figure 9.1). Furthermore, depending on the system under study, this could also permit some understanding of the behavior of the system by observing the values taken by each variable.

9.5.2 PCN to MDP conversion

We have shown how one can convert any MDP, whether in extensional or intensional form, into a PCN model, thus proving that MDP models are in fact a special case of PCNs. Although interesting in itself, a more interesting (and useful) problem consists of going in the opposite direction: converting a class of PCN into a MDP. In order to be able to perform control synthesis on a certain PCN class of models, one would like to automatically transform those PCN models into MDPs so that value/policy iteration can be performed.
<table>
<thead>
<tr>
<th></th>
<th>Real Values</th>
<th>Linprog Results</th>
<th>Normalized</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(V_1</td>
<td>V_1)$</td>
<td>0.3</td>
<td>0.2807</td>
</tr>
<tr>
<td>$P(V_2</td>
<td>V_2)$</td>
<td>0.6</td>
<td>0.6413</td>
</tr>
<tr>
<td>$P(V_3</td>
<td>V_2)$</td>
<td>0.4</td>
<td>0.4275</td>
</tr>
<tr>
<td>$P(V_0</td>
<td>V_1)$</td>
<td>0.7</td>
<td>0.6549</td>
</tr>
<tr>
<td>$P(V_2</td>
<td>V_2)$</td>
<td>0.85</td>
<td>0.9085</td>
</tr>
<tr>
<td>$P(V_2</td>
<td>V_2)$</td>
<td>0.15</td>
<td>0.1603</td>
</tr>
<tr>
<td>$P(V_0</td>
<td>V_1)$</td>
<td>0.5</td>
<td>0.4678</td>
</tr>
<tr>
<td>$P(V_0</td>
<td>V_1)$</td>
<td>0.5</td>
<td>0.4678</td>
</tr>
</tbody>
</table>

Figure 9.8: Value for a state space with two boolean variables

To convert a PCN into a MDP one needs to be able to specify the components required in a MDP model: a finite set of states $S$ (or equivalently a finite set of variables $V$ acting as features of the system); a finite set of actions $A$; conditional transition probability distributions parameterized on $A$ and on a parent subset $S_p$ of $S$ (or $V_p$ of $V$ in the intensional case), one per state (or per variable, feature). Furthermore, one is required to specify a reward function, i.e., a function which one wants to optimize over the course of the agent’s life. For example, a popular reward function within the MDP community is the infinite discounted sum of reward obtained. It is to be noted that although the reward function is needed for computing an optimal policy, it need not be part of the PCN model per se.

When wanting to convert a PCN into a MDP, one would in general not have a controller built yet, since as stated above, a reason for converting the PCN model into a MDP would be for control synthesis. Hence, for this discussion, we will assume that the PCN controller is a black box which will be filled once a policy has been computed using dynamic programming methods such as value or policy iteration.

The first restriction, and perhaps the most important, one that we need to impose on our PCN class is that it must only have one discrete-time clock. MDPs model discrete time systems, where each transition follows the same clock. Furthermore, due to the finite state, finite action set requirements of MDPs, we will also need to restrict our PCN class to finite domains (transductions with only finitely many discrete outputs).

Since a one clock, discrete-time PCN model with only one (discrete and finite) location can trivially be converted into a MDP with an extensional representation, we will ignore this case and...
focus our attention on converting general PCN models (from a subclass of all PCN models) into the intensional MDP framework. Of course, our general definition will include this rather trivial case, since the extensional representation is equivalent to having one and only one feature in the system: the state itself.

In a PCN model, one is given a finite set of locations \( L_c \), a finite set of transductions \( T_d \) (including a finite set \( G \) of generators). For a PCN model to be convertible into a MDP, we also need a set of actions \( A \) from which the controller (to be synthesized) will be choosing. We thus need to require that the PCN controller generate a finite number of discrete outputs where the values of this finite domain will constitute the set of actions \( A \). Therefore, without having to specify the internal behavior of the controller module in the PCN model, we still need to define its domain.

As stated above, without needing an explicit PCN representation of the reward function, one still needs to specify a reward function which will serve to determine the optimal policy. The choice of this function can have a drastic effect on the resulting policy, since different reward functions will in general yield completely different policies. Therefore, specifying the reward function is a very important part of the process. Despite its importance, it might not be easy to come up with the right reward function. The designer might know what she wants the system to do, but the task of specifying the associated reward function remains, like in the MDP case, rather complicated.

With this in mind, we now wish to find a correspondence between the components of a PCN and a MDP model. One question that comes to mind right away when considering a PCN model is that, unlike in a MDP, not all transductions are probabilistic; i.e., not all transductions incorporate a generator inducing a probability distribution. However, as mentioned in the introduction of the PCN model, one can see a deterministic transduction as an instance of a probabilistic transduction; one with probability 1 on one value in the domain and 0 for all other values. Hence, regardless of whether the locations are associated with (output of) deterministic or probabilistic transduction, they can still represent variables (features) of the system within the MDP model. However, there is an exception: locations that are outputs of delay transductions. We will call this proper subset \( L_d \subset L_c \). These locations, which are of use only to avoid algebraic loops within the system, do not represent a feature of the system; thus they should not be used in the converted MDP model.

Therefore, we can see each location in \( L = L_c \setminus L_d \) as a potential variable (feature) of a MDP model. Transductions induce a probability distribution on their output location (point probability...
distribution for deterministic transductions), thus acting as the transition probability for this variable. The inputs of each transduction can be seen as the parents of the location (variable) which is associated with that transduction.

One may wonder if all locations in $L$ are needed in the corresponding MDP. That is, does one need to convert every location into a variable, or can one omit some locations which might not be necessary to compute the optimal policy?

One answer to this question is to pay closer attention to the set of locations in the PCN model and use a subset of useful locations as features of the system. Indeed, many deterministic transductions have output location which are not necessarily important features of the system. For example, simple transliterations like addition, subtraction and $\sin$ have output locations which are not very useful when trying to come up with an optimal policy. Hence, one would want to discard those locations and not include them in the set of features of the system. However, this would clearly complicate the conversion of a PCN to a MDP, as it would undoubtedly require human interaction.

A naive approach that would allow the process to remain automatic would be to use only locations whose corresponding variables are present in the reward function. After all, these are the only variables which are of interest when assessing the value of a given state. However, the answer is not as simple as that. When performing Bellman backups in value iteration, new variables (the parents of the variables in the reward function) will be introduced in the value function, thus augmenting the set of variables needed to compute the policy. Example 9.2 which will follow shortly demonstrates this behavior.

Therefore, if one decides to use only variables present in the reward function, an incremental technique is needed when deciding which locations should be converted into variables of the MDP. Starting with all variables whose associated locations are within the reward function, one needs to add all the variables that are parents to those variables. Then, one needs to add the variables that are parents to these newly added variables, and so on until no new variables can be added. This might convert all locations to variables within the MDP to solve. However, for some modular systems, where many locations are independent of others (only local dependence), many locations may be left out, thus yielding a simpler-to-solve decision-theoretic problem.

However, adding a step of human interaction in the algorithm appears to be the sensible thing to do. In general, the system designer should be comfortable with the system and thus would have
a good intuition on which locations are of particular interest for the controller. This is essentially like looking at the PCN at a higher level and creating new modules which incorporate variables that the designer does not want to control. By creating new modules we create a new probability transition for the variables that are now outputs of these modules, and we change the parent set of these variables. This is essentially location elimination, where we remove the intermediary location by incorporating them within a module. This action removes locations that are not of interest for the PCN controller and thus translation to a smaller MDP can be performed. Furthermore, translating the policy back into this higher level PCN would be as simple as before, that is, the controller would have input into those new added modules, thus controlling only the locations selected by the system designer.

Let us now define formally the synchfin-PCN subclass, the class of all PCN models on which control synthesis can be performed via a transformation of the model into a MDP.

**Definition 9.1 (synchfin-PCN class)** A synchfin-PCN is a PCN, with a unique discrete-time clock, denoted by the tuple \((L_c, T_p, T_d, C_n, R, T_{\text{pol}})\), where \(L_c\) is a set of locations, each associated with a finite domain; \(T_p\) is a finite set of labels of probabilistic transductions, each with an output port, a set of input ports (parent set) and associated with a discrete probability distribution, parameterized by their input locations; \(T_d\) is a finite set of labels of deterministic transductions, each deterministic transduction with an output port, a set of input ports and associated with a parameterized point probability distribution; \(C_n\) is a set of connections between locations and transduction (either deterministic or probabilistic) ports.

Additionally, two new components are needed:

- A reward transliteration \(R\), on which some measure of its trace is to be maximized over the life of the system, must be specified. The inputs of this transliteration are the locations which are parameters of the reward function. The output location for this transliteration is named \(R\) with domain \(\mathbb{R}\).

- A controller transduction (to be synthesized) \(T_{\text{pol}}\) is required to complete the PCN model of the system. The designer needs to specify its domain, whose values will represent the action set \(A\) within the MDP model.

Now that we have formally introduced the synchfin-PCN class, let us present, in Figure 9.9...
and 9.10 the algorithms used to convert a PCN model into a MDP and to perform control synthesis respectively. We follow with an example which demonstrates how control synthesis can be performed on a (simple) PCN model.

The version of the algorithm presented is the fully automated one: it considers the case where the designer does not interact with the selection of the locations that will be associated with a variable in the MDP. A semi-automatic method allows the designer to select the locations $L_{vip}$ that are deemed of interest to the controller. Once this selection has been completed, location elimination is performed to remove all locations in $L \setminus L_{vip}$ and we obtain the new transition probability distributions. In this version of the algorithm, the returned MDP is simply $\langle U, A, NP, Rew \rangle$ where $U$ is the set of variable associated with $L_{vip}$; $A$ is the original set of actions, i.e., each action is associated to one element in the domain of the controller; $NP$ is the new set of transition probability distributions as obtained by performing location elimination; and $Rew$ is the original reward function.

Example 9.2 (Control Synthesis) Let us present a simple PCN model, constituted of two locations, $V$ and $A$, which are outputs of generators. Two other locations, $M$ and $P$, are part of the system but are outputs of deterministic transliterations (modulo 2 and addition respectively). The goal of the designer is to synthesize a controller (labeled Policy in the system) that maximizes the value of $P$ at every time step. This requirement can easily be converted into a reward function, one which only depends on $P$ and whose value increases as the value of $P$ increases. In this case, the designer opted to assign a reward of 0, 3, 9 and 20 when the value of $P$ is 0, 1, 2, 3 respectively. Figures 9.11 - 9.12 show the PCN model of this system, which we call VAMP, along with the 2TBN representation. Notice the synchronous arc between variables $P$ and $M$, indicating that there is a correlated effect between $M$ and $P$, at any given time $t$. This can be seen in Figure 9.11 by observing the absence of unit delay between locations $P$ and $M$. However, as explained earlier, synchronic arcs are no problem when solving an MDP.

Let us assume that we wish to perform control synthesis by using the fully automated version of our algorithm, i.e., we let the algorithm choose which locations should be associated with a variable in the equivalent MDP. By observing the above description of the reward function, we can see that we are only interested in maximizing the values of location $P$ over time. However, due to the large set of parents of location $P$, as seen in the 2TBN for this system, we can easily deduce the result of our algorithm: every location shall be associated with a variable in the MDP model.
Let $SF = \langle Lc, G, Td, Cn, R, T_{pol} \rangle$ be the synchfin-PCN model to convert into a MDP where:

$Lc = \{L_1, L_2, \ldots, L_n\}$ is the set of all locations (features)

$G = \{G_1, G_2, \ldots, G_k\}$ is the set of all labels of generators

$Td = \{Td_1, Td_2, \ldots, Td_i\}$ is the set of all labels of deterministic transductions

$Cn = \{Cn_1, Cn_2, \ldots, Cn_m\}$ is the set of all connections

$T_{pol}$ is a policy transduction, yet to be specified but for which the domain is known

$R : Lc \times \text{dom}(T_{pol}) \rightarrow \mathbb{R}$ is the reward function as specified by the system designer

Let $MDPSF = \langle V, A, P, Rew \rangle$ be a MDP model

and let:

$V := \emptyset$ be the (finite) set of all variables (features)

$A := \emptyset$ be the (finite) set of all actions

$P := \emptyset$ is the set of all transition probability distributions

Rew : $V \times A \rightarrow \mathbb{R}$ is the reward function

Let $T := \emptyset$ be the set of all variables associated with the locations in the PCN

1. For Each location $L_i \in L = Lc \setminus Ld$, where $Ld$ is the set of output locations of delay transductions,
   (a) Associate a variable $v_i$
   (b) $T = T \cup \{v_i\}$

2. For Each value $pol_i \in \text{dom}(T_{pol})$
   (a) Associate an action $a_i$
   (b) $A = A \cup \{a_i\}$

3. Set $Rew := R$

4. Set $\Pi \subseteq T$ to the set of variables involved in $Rew$.

5. For Each $\pi \in \Pi$:
   (a) $V = V \cup \{\pi\}$
   (b) Set $\Pi_{\pi}$ to the set of variables associated to the input locations of the transduction whose output location is associated to $\pi$.
   (c) Recursively repeat Step 5 to $\Pi_{\pi}$

6. For Each $v_i \in V$
   (a) Set $P(v_i)$ to the transition probability distribution associated with the transduction whose output location is associated to variable $v_i$

Return $MDPSF = \langle V, A, P, Rew \rangle$

Figure 9.9: pcn2mdp algorithm

We proceeded to dynamic programming using the SPUDD software of [HSAHB99]\textsuperscript{3} which is available for download at www.cs.ubc.ca/spider/staubin/Spudd/index.html

\textsuperscript{3}available for download at www.cs.ubc.ca/spider/staubin/Spudd/index.html

195
Let $MDP_{SF} = (V, A, P, Rew)$ be the MDP model obtained from PCN model $SF$.

Perform Value Iteration on $MDP_{SF}$ and obtain an optimal policy $\Pi(MDP_{SF})$.

Convert $\Pi(MDP_{SF})$ into policy transduction $T_{pol}$ where locations associated to variables in $\Pi(MDP_{SF})$ become input locations of $T_{pol}$.

Add $T_{pol}$ to the synchfin-PCN $SF$ to complete the control synthesis.

Figure 9.10: Control Synthesis algorithm

Figure 9.11: VAMP problem in PCN framework

Appendix A shows the MDP representation of the VAMP system. Notice that even if the reward function at the bottom of the sample code depends solely on variable $P$, the optimal policy depends on all variables in the system, as predicted by our algorithm.$^4$

Now let us consider a different scenario: assume that the designer is only interested in $M$, $V$ and $A$ and let us assume that the designer is interested in maximizing the same quantity (the sum

---

$^4$One should note that for this example, the reward function does not depend on the action selected; although the conclusions obtained from this example would apply to the general definition
Thus, the designer believes that the controller should have effect only on those three locations, which means that location P should be eliminated. However, given the removal of location P, we need to specify a slightly different reward function which turns out to be more complicated (see the MDP description of the VAM system in Appendix A). Figures 9.14 - 9.15 respectively represent the new PCN model of the VAMP example and its 2TBN after the location removal of location P.

When converting this higher level PCN model into a MDP, the algorithm generates an MDP model that only contains three variables, namely V, A and M. The transition distribution for variable M is now different than in the previous system, since the algorithm updated it while removing variable P (see Appendix A to compare the M's transition distribution for both systems). Note that no other transition distribution were updated since none of V and A has P as parents. Furthermore, notice that there are no longer any correlated effects in the system. The removal of location P in the PCN system results in creating a system where there are no longer any synchronic arcs (see Figure 9.15).

We proceed to dynamic programming to generate the optimal policy. Figures 9.16 and C.2 (shown in Appendix A) display the resulting optimal policy and optimal value function respectively.

One can observe that the optimal policy is now much simpler than the one for the vamp problem. This is due to the fact that the designer selected only three out of the four variables for conversion within the MDP model, thus generating a simpler MDP. However, this comes with an obvious price
tag since the designer has to specify a more complicated reward function (once again see Appendix A to compare the two reward functions).

For both examples, it is important to notice the complexity of the optimal policies. Although the systems are rather simple ones, the task of specifying a controller which will maximize the value of \( P \) at any time step is far from trivial (as shown in Figure 9.13 and Figure 9.16). Therefore, this proves that control synthesis is clearly needed, even for simple systems, and the addition of such power to the PCN framework greatly increases its ease of use. Because it combines a flexible and powerful modeling environment with a control synthesis algorithm, the PCN framework is thus an appealing paradigm. \( \square \)
Example 9.3 (A cat guarding a mouse) Here we present a more robotic-inclined (and more complex) system. Let's assume that we have a robotic cat guarding a caged mouse. For the sake of simplicity, we will assume a grid-world as represented in Figure 9.17. The mouse's cage is represented by the three dark squares while the area where the cat can move around is in lighter tone. We model the robotic cat as our agent, which we wish to control and we will consider the mouse as an external agent; that is, we have no direct control over it. However, we will assume a certain probabilistic knowledge of the behavior of the mouse, i.e., depending on the behavior of our robotic cat agent, we know, with some uncertainty, what the mouse will do. We will model our system with a series of variables, which together represent the full behavior of the system. First, we will have variables Cx, Cy, Mx and My, which represent the (x, y) position of the cat and the mouse respectively.
An assignment of values to these variables completely defines the location of the cat and the mouse at any given time. Furthermore, we have a variable Cstate which tells us the current mind-state of our cat agent and finally we have variable hasM which is a binary variable assessing whether or not the cat has captured the mouse and is currently holding it. In Figure 9.17, the state of the system is such that the mouse is currently in middle square of her cage (position (0, 1)) while the cat is on square (2, 2) and is guarding the mouse (Cstate = guard). However, this is a stochastic dynamical
system and (unpredictable) changes can occur. For instance, the mouse can escape from her cage, in which case the cat chases her until capturing her and then can attempt to put her back into the cage. The cat might fail thus losing the mouse who once again will roam freely in the environment.

The cat can adopt different mind-states which affect the behavior of the system. It can guard the mouse, rendering escape attempts more difficult; it can rest and it can also play. In the last two mind-states, the mouse has a greater escape potential since the cat is not paying full attention to her.

The cat also has a choice of several actions that it can perform at any given time: grab the mouse, move:right, left, up, down within the grid, cage the mouse and change mind-states to guard, rest or play.

The system is designed such that the cat gets rewarded with a treat (robotic treat that is) when the mouse is caged, with increasing level of reward when in the mind-state of guarding, resting and playing respectively. The robotic cat is trying to maximize the amount of food it can accumulate over time, where immediate food is more valuable than food received in the future. Hence, the optimal situation for the cat would be to always be playing while the mouse remains in its cage. However, it is exactly when the cat is playing that the mouse is more likely to succeed in her escape attempts. Therefore, the controller must be able to assess the risk of the mouse escaping, and depending on that risk level, tell the robotic cat what to do so that it can accumulate as many treats as possible. Figure 9.18 shows the PCN model of the cat and mouse system, where the controller π remains to be defined.

However, this control problem is far from simple as the system designer needs to consider many cases such as the mouse position, the cat position, its mind-state and the action it attempts. All those influence the behavior of the system in a stochastic fashion, rendering the control synthesis quite complicated. Therefore, it would be very helpful if we could automatically generate a controller for this system.

One can notice that for our system, the reward structure is explicitly represented within the PCN model. The task of converting this system from a PCN to a MDP would not necessitate the creation of a reward function. Notice also that this system is very well suited for our conversion algorithm. Indeed, there is only one discrete-time clock, the states are discrete as well and we are provided with a fully specified reward function: we are in the presence of a synch-fin PCN model. Fig-
Figure 9.18: PCN model for the cat mouse system

Figure 9.19 shows the temporal Bayesian network representing the conditional independence among the variables in the system. The MDP code, resulting from our conversion algorithm is shown in Appendix A.

We once again proceeded to dynamic programming. Figure 9.20 and C.3 (shown in Appendix A) display the resulting optimal policy and optimal value function respectively.

9.6 Introduction to Partially Observable MDPs

As presented above, an MDP assumes full observability: POMDPs are used to model systems for which this assumption has to be lifted. Partial observability clouds the idea of the current state. It
removes the certainty that the agent has about the state of the world and renders the selection of an action much more complicated. Thus, when the state of the system is not completely observable, one must add the notion of observations. The agent, although not being able to fully assess the state in which it is at any given time, can observe its surroundings and infer the state based on those observations. Therefore, a model of observations is added to the existing MDP model to make it a POMDP = (S, A, P, R, O, O) where O is a finite set of observations and O is an observation function, mapping A × S into discrete probability distributions over O. The probability of making observation o when performing action a in state s is thus denoted by O(a, s, o).

Although the underlying dynamics of a POMDP are still Markovian (as for an MDP), the uncertainty surrounding the knowledge of the current state requires that the agent keep track of (possibly) the entire history of the process, making this a non-Markovian process. The history of the process refers to the agent's knowledge of its starting situation, all actions performed and all observations made up to the current time step. Fortunately, for the agent to maintain a probability distribution...
over all of the states (referred to as the belief state from now on) is equivalent to maintaining the complete history of the process. This is due to the fact that the belief state is a sufficient statistic. A proof of this interesting result can be found in Appendix A of [SS73].

We refer to the set of all possible belief states as $B$ and $b(s)$ (where $b(\cdot) \in B$) represents, for each state $s \in S$, the probability that the environment is in state $s$. Therefore, the only overhead to be performed is in updating the distribution, which turns out to be a fairly simple task, involving the transition and observation probabilities along with Bayes' formula.

As with an MDP, there is a policy component to a POMDP, which is the mapping of the current belief state into an action. The optimal policy of a POMDP can in fact be computed very similarly to one in an MDP. One needs to define a continuous-space belief MDP as $\langle B, A, \Phi, \Gamma \rangle$, where $B$, the
set of belief state, is the state space; \( \mathcal{A} \) remains the set of actions; \( \Phi \) is the state transition function from \( \mathcal{B} \times \mathcal{A} \) into probability distributions over \( \mathcal{B} \) defined as

\[
\Phi(b, a, b') = Pr(b'|a, b) = \sum_{o \in \mathcal{O}} Pr(b'|a, b, o) Pr(o|a, b)
\]

with

\[
Pr(b'|b, a, o) = \begin{cases} 
1 & \text{if StateEst}(a, b, o)^5 = b', \\
0 & \text{otherwise}.
\end{cases}
\]

Finally, \( \Gamma \) is the reward function, from \( \mathcal{B} \times \mathcal{A} \) to \( \mathbb{R} \), which is constructed, in the obvious way, from the original reward function:

\[
\Gamma(b, a) = \sum_{s \in \mathcal{S}} b(s) R(s, a).
\]

At first glance, this new reward function might appear as if the agent is rewarded simply by believing it is in a state with high value. However, one must remember that the belief function is not a choice of the agent but rather a function of the observation and the transition model, which are assumed to correctly represent the world surrounding the agent.

This belief MDP is therefore well defined and can be solved to generate an optimal policy. The policy is also optimal for the original POMDP specification, thus solving this continuous-space MDP is equivalent to solving the much more complicated POMDP. Unfortunately, solving continuous-space MDPs is far from trivial as the state space can get really large, even for problems of modest size. Luckily, it has been shown in [SS73, Son78] that this belief MDP has some special properties, namely piecewise linearity and convexity of the value function, which make the problem much simpler to solve.

### 9.7 POMDPs correspondence to PCN

As presented above, a POMDP model can be seen as the coupling of a state estimator and a policy. This representation is very similar to the PCN framework as shown in Fig 9.21.

\(^5\)StateEst is the function that given an action, a belief state and an observation, returns the updated belief state. It will be presented in more detail soon.
Figure 9.21: Similarities between POMDP and PCN frameworks

The state estimator component takes as input the most recent observation obtained from the sensors, the most recent action and the last belief state; and returns the updated belief state. The state estimator component emerges from a straightforward application of Bayes’ rule. Indeed, the output of the state estimator component is a belief state which can be seen as a vector of probabilities. For every state $s \in S$, the component of the belief state, referred to as $StateEst_s(b, a, o)$, can be computed using the previous belief state $b$, the previous action $a$ and the current observation $o$ in the following way:

$$StateEst_s(b, a, o) = \frac{Pr(s|a, o, b)}{Pr(o|a, b)}$$

where $Pr(o|a, b)$ is a normalizing factor defined in the usual way:

$$Pr(o|a, b) = \sum_{s' \in S} O(a, s', o) \sum_{s \in S} P(s, a, s')b(s).$$

As was the case with the MDP framework, the policy component serves as a controller, that is, given the most up-to-date belief state, it selects the best action to be performed by the system.

As with the MDP setting, the conversion of an extensional POMDP model to a PCN model is rather simple, as is shown in Figure 9.22. Each PCN model of a POMDP will have a simple set of locations: $b, a$ and $o$, which represent the belief state, the action and the observation at a given time. Two main transductions will also be part of the PCN, namely the policy transduction (to be synthesized or specified by the system designer) and a state estimator transduction implementing
Equation 9.7. Also, two delay transductions are needed to avoid algebraic loops which arise from the dependencies with values at the previous time step.

Once again, the intensional description of a POMDP is more appealing than the extensional. Indeed, by introducing state and observation features, we get a more compact representation of the POMDP which is often also more intuitive to a designer. By using a factored representation, one would like to take advantage of the structure in the problem and avoid having to specify the $|A|$ different $|S| \times |S|$ transition probability matrices (one per action) and the $|A|$ distinct $|S| \times |O|$ observation probability matrices. Recent work on factored POMDPs [BP96, HF00] using decision trees and Algebraic Decision Diagrams (ADDs) have extended the work of [Mon82]

In Figure 9.23, we show a simple intensional POMDP. Notice that unlike the MDP case, we now need to introduce a set of observation features. This means that the agent can get multiple observations at a given time step and an assignment of value to every one of these observations defines the observation state. This setting appears to be more suitable for robotic domains which very often have multiple sensors such as cameras, ultra sound and laser range finders, constantly generating observations.

Once again, a more interesting problem is the conversion of a PCN into a POMDP so that control synthesis can be performed. Therefore, we would like to specify a class of PCN models for which control synthesis is possible. Combined with the synchfin-PCN class, we would then be equipped with a large class of dynamical stochastic systems for which the task of specifying the controller can be automated. The conversion of a PCN into an POMDP is similar to the case of PCN to MDP. The details are left for future work as the case for the power of expression of the PCN framework has been made extensively already.
Figure 9.23: Intensional POMDP
Chapter 10

Problems, Solutions and Contributions

10.1 Summary of Contributions

In this thesis, we integrated the design and analysis of robotic systems and behaviours by establishing a foundation for modeling, analyzing, specifying, verifying and synthesizing complex uncertain robotic systems that interact with changing environments over time. Our approach extends the Constraint Nets (CN) framework that was proposed by Zhang and Mackworth [ZM95a] for deterministic discrete/continuous hybrid systems. We have developed a semantic model for uncertain hybrid dynamical systems, two languages for requirements specification, a formal method for behaviour verification and investigated potential approaches to control synthesis. In this chapter, we review the goals attained throughout this work, and suggest topics for future work.

As presented in the previous sections, we have decomposed the problem of design and analysis of uncertain robotic systems into four phases: modeling, specification, synthesis and verification. For each individual phase, we have developed formal methods and established the mutual dealings between all the phases.

First, we have developed a semantic model for uncertain hybrid dynamical systems, that we call Probabilistic Constraint Nets (PCN). Based on abstract algebra, topology and measure theory, we have represented both time and domains in abstract forms, and uniformly formalized basic elements of dynamical systems in terms of traces, transductions and probabilistic transductions. Furthermore,
we have also studied both primitive and event-driven transductions which are important elements of dynamical systems, with or without uncertainty.

Since PCN is an extension of the CN framework for deterministic hybrid dynamical systems, it is also an abstraction and generalization of data-flow networks, with the addition that we explicitely handle the uncertain components of the system. Within this framework, the behaviour of a system (the semantics of a PCN model) is formally obtained using both the theory of continuous algebra and stochastic systems. Specifically, a probabilistic constraint net models an uncertain dynamical system as a set of interconnected transductions (deterministic and probabilistic), while the behaviour of the system is the set of input/output traces of the system satisfying all the relationships (constraints on the dynamics) imposed by the transductions. Similarly to CN, PCN models a hybrid system using event-driven transductions, while the events are generated and synchronized within the system.

The motivation for developing the PCN framework is to be able to model hybrid dynamical systems while considering the underlying uncertainty in the system. Uncertainty is inherent in any physical system, hence modeling its effects and considering its impact on system behaviour is essential. We have shown that PCN subsumes most existing computational models handling uncertainty, confirming the expressive power of PCN. Moreover, within PCN, both sequential and analog computations can be modeled effectively.

Second, we have developed two languages, PATTL, a branching time probabilistic logic, and average timed $\forall$-automata, for requirements specification. Average timed $\forall$-automata are non-deterministic finite state automata augmented with local and global average time bounds. In order to handle general systems (discrete time, continuous time, or event-based) both languages are defined on abstract time and domains.

Third, we have developed a formal method, based on model checking and stability analysis, for on average behaviour verification. This verification method is semi-automatic if the time structure is discrete, and is automatic, if, in addition, the domains are finite.

Finally, to address the fourth problem considered in this body of work, we have investigated a systematic approach to control synthesis using stochastic and robust control. Moreover, we have demonstrated that for a certain class of PCN, we can use the very efficient dynamic programming algorithms used in the (PO)MDP field of research.

In summary, we have proposed a theoretical foundation for modeling robotic system exhibiting
uncertainty and for analyzing the behaviours of those systems. We summarize what we believe to be the major contributions of this dissertation as follow:

- **Probabilistic Constraint Nets for the Analysis of Uncertain Hybrid Dynamical Systems**
  PCN possesses the essential properties of a desired model for robotic systems (modified from [LS90] and [Zha94]), namely:
  
  - *Real-Time*: time is explicitly represented, whether it be discrete, continuous or event-based (see hybrid property),
  - *Symmetrical*: the dynamics of environments as well as the dynamics of robotic agents and control can be modeled,
  - *Hybrid*: multiple time and domain structures are uniformly formalized,
  - *Hierarchical*: multiple levels of abstraction are provided,
  - *Formal*: formal syntax and semantics are defined, and formal analysis is facilitated, and
  - *Probabilistic*: formal modeling language which allows to model and analyze systems exhibiting a variety of different types of uncertainty.

- **PATTL and Average-time Timed V-Automata for Requirements Specification**
  PATTL specifies probabilistic discrete/continuous sequential/timed behaviours uniformly; average timed V-automata provide a simple alternative to PATTL, which is illuminating by its graphical representation, and, in some cases, more powerful.

- **A formal Method for On Average and Probabilistic Behaviour Verification**
  These verification methods apply to behaviours of hybrid systems in general under a subclass of uncertainty. The average timed V-automata verification method is semi-automatic for discrete time systems and automatic for discrete time and finite domain systems.

- **Constraint-based Requirements Specification and Control Synthesis**
  This approach proposes a general framework for control synthesis when in the presence of uncertainty. We consider approaches from optimal robust control and stochastic control theory as well as dynamic programming methods used in the MDP framework.
An Integrated Approach to the Design and Analysis of Uncertain Robotic Systems

This dissertation decomposes the problems in the design and analysis of robotic systems and behaviours, and focuses on the relationships among modeling, specification, synthesis and verification, while considering the underlying uncertainties of the systems under study.

10.2 Future Work

In this section we suggest further research directions that this approach could follow. We divide these topics in two main categories: theoretical and practical extensions.

10.2.1 Further Desired Theoretical Development

We have proposed a foundation for the design and analysis of uncertain robotic systems and their behaviours. Obviously, this is a wide area, and a lot more questions remain to be addressed. We believe our work constitutes a step along the way to a powerful and intuitive theory of robotic systems exhibiting uncertainty. Further research that would logically follow from this work includes:

• More Expressive Specification Languages

There are behaviours that are not expressible using PATTL or average-time timed \(V\)-automata, such as long-run behaviours (as investigated in [dA97a]), maximizing global utilities or timed behaviours over intervals. Other specification languages, with more expressive power and formal verification procedures, are yet to be explored. For example, we could extend time bounds on average timed automaton states with upper and lower bounds and verify that the average time falls within these bounds. This could be done without substantially modifying our verification method.

• (Semi-)Automatic Verification for Special Classes of Hybrid System

It has been shown that there are simple hybrid systems that have algorithmic verification [ACHH93]. More work along this line can be done. For example, a stochastic linear continuous system combined with a finite state automaton is a special class of hybrid system that might have simpler verification procedures.
Another desirable extension would be to modify the average-time timed V-automata verification techniques to handle probabilistic requirements (verification with $\alpha < 1$). This would greatly increase the number of applications for which this technique can be used in practice.

- (Semi-)Automatic Synthesis and Analysis of Controllers for Special Classes of Systems

For finite domain systems and certain types of uncertainty, controllers can be synthesized automatically, though with a high degree of complexity. For linear systems, stochastic stability can be analyzed semi-automatically. However, more work along this line can be done to augment the classes of systems that can be handled. For example, combining techniques for optimal stochastic control and robust control could lead to efficient methods of control synthesis for uncertain dynamical systems.

10.2.2 Further Practical Applications

Based on the CN semantic model, a visual programming and simulation environment called ALERT: A Laboratory for Embedded Real-Time systems has been developed [Zha94]. Moreover, CNJ, a JAVA visual programming environment for constraint nets has been developed by Song and Mackworth [SM02]. We have extended ALERT into a Matlab module that follows the PCN semantic model, and that we call PCNmatlab. An extension of the CNJ environment into PCNJ is straightforward and in the making. However, this is only a start since more work is needed along those line and includes:

- a formal programming language with a real-time semantics

PCN is an abstraction of data-flow models for hybrid systems exhibiting uncertainty, with abstract data types and abstract reference time. An instantiation of the data types, the uncertainty sources and the reference time result in a programming language, which can be used for both modeling and programming (control). PCNmatlab, ALERT and CNJ are such a language for modeling. The CES programming language (C for Embedded Systems) of Thrun [Thr98] proposes a new approach for programming robots and other embedded systems and supports computing with uncertain information. However, as the language is an extension of the C programming language, it does not support the temporal aspect of the robotic systems explicity.
• a specification and verification environment based on our methods

Average Timed $\mathcal{V}$-automata have a graphical representation, which can be implemented on a graphical user interface. The formal verification method for discrete time systems can be implemented on an interactive theorem prover based on the verification algorithm shown in the corresponding chapter.

• an integrated design and analysis environment for developing uncertain robotic systems

As both PCN models and average timed $\mathcal{V}$-automata can be implemented on a graphical user interface, one could develop an integrated environment that facilitates both verification and simulation of complex robotic systems exhibiting uncertainty.

• perform experiments on real robotic systems to uncover more design problems and verify the value of the framework

The main goal of this thesis is to establish a theoretical foundation for the problem of design and analysis of robotic systems. Hence, we seek to provide an efficient and formal tool for working on real-life robotic systems. By actually applying the PCN framework to the modeling of real-life robotic systems, we will obtain empirical data that will help us assess the value of the theoretical framework while also allowing us to improve on the current state of the framework. We see these experiments as a bidirectional road that will allow us to move from theory to practice and also from practice to theory.

As of now, the CN framework has already proved useful for several real-life robotic systems, from robot soccer to a situated robot that repeatedly finds, tracks, chases and kicks a ball, called Ainia [SMBK95, Zha98, MM03, MOM04]. We are confident that adding uncertainty models to such systems, by modeling them with PCN, will prove to be an improvement on the actual implementations.
Appendix A

Proofs of Theorems

A.1 Measure-Theoretical and Topological Structure of Dynamics

**Proposition 2.1** (1) A subset is closed iff it includes all its limit points. (2) A topology is trivial iff every point $x$ is a limit point of any subset with elements distinct from $x$. A topology is discrete iff no point is a limit point of any subset.

**Proof:** (1) If a subset $S$ of $X$ is closed, $X - S$ is open and there is no point in $X - S$ that is a limit point of $S$. If there is no point in $X - S$ that is a limit point of $S$, $S$ is closed, since if $S$ is not closed, $X - S$ is not open. If $X - S$ is not open, there is at least one point in $X - S$ that is a limit point of $S$, otherwise every point in $X - S$ has a neighborhood in $X - S$, thus $X - S$ is open.

(2) If a topology is trivial, any point has only one neighborhood, the total set. If every point $x$ is a limit point of any subset with elements distinct from $x$, the topology is trivial since otherwise there is an open set $S \subset X$ and no point in $S$ is a limit point of $X - S$, contradiction. If a topology is discrete, any point is a neighborhood of itself, thus cannot be a limit point of any subset. If no point is a limit point of any subset, the topology is discrete since otherwise there is a point that is not open, which is a limit point of the total set, contradiction. ■

**Proposition 2.2** A topological space is connected iff the only sets that are both open and closed are the empty set and the total set.

**Proof:** If there is $\emptyset \subset X' \subset X$ that is both open and closed, both $X'$ and $X - X'$ are non-empty open sets. Therefore, $X$ is separated. ■

**Proposition 2.3** (1) Continuous functions are closed under functional composition. (2) A function $f_\omega : X \to X'$ is pathwise continuous, iff $x \in X$ is a limit point of $S \subset X$ implies that $f_\omega(x)$ is a
point or a limit point of \( f(S) = \{ f(x) | x \in S \} \).

**Proof:** The first property is deduced directly from the definition of continuous functions. The second property is deduced from an equivalent definition of pathwise continuous functions, i.e., a function is pathwise continuous iff the inverse image of any closed subset is closed, and from the property that a closed subset includes all its limit points. ■

**Proposition 2.4** Let \( \langle X, \tau \rangle \) be a topological space, \( X' \subseteq X \) and \( \tau' = \{ W | W = X' \cap U, U \in \tau \} \).

The collection \( \tau' \) is a topology on \( X' \). We call \( \tau' \) the **subspace topology** on \( X' \), and \( \langle X', \tau' \rangle \) a subspace of \( \langle X, \tau \rangle \).

**Proof:** This result is obtained directly from the definition of topology. ■

**Proposition 2.5** Let \( \{ X_i \}_{i \in I} \) be a family of topological spaces and \( J \) be an arbitrary index set. Then \( (\times_i X_i)' = \times_i X_i' \).

**Proof:** \( \times_J (\times_i X_i) \) and \( \times_I (\times_j X_j) \) are isomorphic. ■

**Proposition 2.6** A flat partial order is a cpo.

**Proof:** \( \bot_A \) is the least element and every directed subset is a linearly ordered subset of \( A \) with a greatest element. ■

**Proposition 2.7** The product of cpos is a cpo. Let \( \{ A_i \}_{i \in I} \) be a set of cpos and \( \mathcal{A} = \times_i A_i \). The least element of \( \mathcal{A} \) is \( \bot_{\mathcal{A}} \) with \( (\bot_{\mathcal{A}})_i = \bot_{A_i}, \forall i \in I \). Let \( D \) be a directed subset of \( \mathcal{A} \). The least upper bound of \( D \) is \( \bigvee_A D \) with \( \bigvee_A D_i = \bigvee_{A_i} D_i, \forall i \in I \), where \( D_i \) is the projection of \( D \) onto its \( i \)th component, i.e., \( D_i = \Pi_i D \).

**Proof:** According to the definition of least elements and least upper bounds. ■

**Proposition 2.8** The partial order topology of a non-trivial partial order is non-Hausdorff.

**Proof:** For any \( a <_A a' \), every neighborhood of \( a \) includes \( a' \). ■

**Proposition 2.9** Any continuous (or pathwise continuous) function is monotonic, i.e., if \( f : \Omega \times A \to A' (f_\omega : A \to A') \) is continuous (pathwise continuous), then \( (\omega_1, a_1) \leq_{\Omega \times A} (\omega_2, a_2) (a_1 \leq_A a_2) \) implies \( f(\omega_1, a_1) \leq_A f(\omega_2, a_2) (f_\omega(a_1) \leq_{A'} f_\omega(a_2)) \).

**Proof:** We prove this result for pathwise continuous functions. The result extends easily to continuous functions. Suppose \( f_\omega(a_1) <_{A'} f_\omega(a_2) \), then according to the definition of partial order topology, there is an open set \( S \subseteq A' \) including \( f_\omega(a_1) \) but not \( f_\omega(a_2) \). Therefore, \( f_\omega^{-1}(S) \subseteq A \) is an open set including \( a_1 \) but not \( a_2 \). So \( a_1 \not<_A a_2 \). ■

**Proposition 2.10** Let \( A \) and \( A' \) be two cpos. Then \( f : \Omega \times A \to A' \) is continuous iff for every...
directed subset \( D \subseteq (\Omega \times A) \),

1. \( f(D) = \{ f(d) | d \in D \} \) is directed and

2. \( f(\bigvee_{\Omega \times A} D) = \bigvee_{A'} f(D) \).

The same result applies to pathwise continuous functions.

**Proof:**

If \( (\rightarrow) \): From Proposition 2.9, we know that if \( f \) is continuous, \( f \) is monotonic. Therefore, if \( d \) is an upper bound of \( d_1 \) and \( d_2 \), \( f(d) \) is an upper bound of \( f(d_1) \) and \( f(d_2) \). Therefore, if \( D \) is directed, then \( f(D) \) is directed and \( f(\bigvee_{A} D) \geq_{A'} \bigvee_{A'} f(D) \). We now prove that \( f(\bigvee_{A} D) \leq_{A'} \bigvee_{A'} f(D) \).

If \( f(\bigvee_{A} D) \not\leq_{A'} \bigvee_{A'} f(D) \), there is an open set \( S \subseteq A' \) including \( f(\bigvee_{A} D) \) but not \( \bigvee_{A'} f(D) \). Therefore, \( f^{-1}(S) \subseteq A \) is an open set including \( \bigvee_{A} D \) but not any \( d \in D \), contradicting to the definition of open sets in partial order topologies.

Only if \( (\leftarrow) \): If conditions (1) and (2) are satisfied, \( f \) is monotonic. Therefore, for any upward closed set \( S \), \( f^{-1}(S) \) is also upward closed. Since \( f(\bigvee_{A} D) = \bigvee_{A'} f(D) \), if \( S \) is inaccessible from any directed subset \( f(D) \), then \( f^{-1}(S) \) is inaccessible from any directed subset \( D \). Therefore, \( f \) is continuous since for any open set \( S \), \( f^{-1}(S) \) is open.

**Proposition 2.11** Metric topologies are Hausdorff.

**Proof:** Given any two elements \( x, x' \) with \( l = d(x, x') \), \( N^{1/2}(x) \cap N^{1/2}(x') = \emptyset \).

**Proposition 2.12** Let \( (X, \mathcal{B}) \) be a measurable space. Every continuous mapping of \( X \) is Borel measurable.

**Proof:** Let \( f : X \to Y \) be a continuous mapping of \( X \), where \( Y \) is any topological space. Then by definitions of continuity, \( f^{-1}(V) \in \mathcal{B} \) for every open set \( V \) in \( Y \).

**Proposition 2.13** If \( X \) is of a Hausdorff topology and \( v : \Omega \times L \to X \) is a stochastic linear set of values, then \( F_0 \to F_{v_1} \) and \( F_0 \to F_{v_2} \) imply \( F_{v_1} = F_{v_2} \).

**Proof:** Since \( X \) is of Hausdorff topology, we know that for two disjoint points \( v_1 \in X \) and \( v_2 \in X \), there exist neighborhoods \( N(v_1) \) and \( N(v_2) \) such that \( N(v_1) \cap N(v_2) = \emptyset \).

Fix \( \omega \in \Omega \). Since a limiting distribution is assumed to exist for \( v : \Omega \times L \to X \), we also have that the linear set of values \( v_\omega : L \to X \) converges to a value \( v^* \), i.e., \( v \to v^* \). Since \( F_{v_1} \neq F_{v_2} \), then for any \( \omega \in \Omega \), there exists \( v_1^* \) and \( v_2^* \), 2 disjoint limits such that \( v_\omega \to v_1^* \) and \( v_\omega \to v_2^* \). Hence there is \( l_0 \), such that for all \( l \geq L \), \( v(l) \in N(v_1^*) \cap N(v_2^*) \), which constitute a contradiction on the
assumption that $v^*_1$ and $v^*_2$ are disjoint and thus on the assumption that $F_{v^*_1} \neq F_{v^*_2}$. Therefore, we obtain that if $F_v \rightarrow F_{v^*_1}$ and $F_v \rightarrow F_{v^*_2}$, then $F_{v^*_1} = F_{v^*_2}$. ■

**Proposition 2.14**

1. For any time structure $(T, d, \mu)$, if $T \subseteq T$ has an upper bound in $T$, $T$ has a least upper bound in $T$.

2. The following properties for a time structure are equivalent:

   (a) $(T, d, \mu)$ is discrete.

   (b) Let $(t_1, t_2) = \{t | t_1 < t < t_2\}$. For all $t$, if $t$ is not the least element of $T$, then $\exists t' < t$, denoted $\text{pre}(t)$, such that $(t', t) = \emptyset$, and for all $t$, if $t$ is not the greatest element of $T$, then $\exists t' > t$, denoted $\text{suc}(t)$, such that $(t, t') = \emptyset$.

   (c) $(T, d, \mu)$ is well-founded, i.e., $\forall t \in T, [0, t)$ is finite.

3. The following properties for a time structure are equivalent:

   (a) $(T, d, \mu)$ is continuous.

   (b) $(T, d, \mu)$ is dense, i.e., for all $t_1 < t_2$, there exists $t_0$ such that $t_1 < t_0 < t_2$.

**Proof:** (1) For any $T \subseteq T$ with an upper bound $t \in T$, let $\tau = \inf \{m(t) | t \text{ is an upper bound of } T\}$. Since $T$ is a time structure, $\{t | m(t) \leq \tau\}$ has a greatest element $t_0$. Since $T \subseteq \{t | m(t) \leq \tau\}$, $t_0$ is the least upper bound of $T$.

   (2) (a) $\rightarrow$ (b): For any $t$, $t$ is not the least element of $T$, let $\tau = \sup \{m(t') | t' < t\}$. Since $T$ is a time structure, $\{t' | m(t') \leq \tau\}$ has a greatest element, denoted $t_0$. Since $T$ is discrete, $t_0 < t$. However, $(t_0, t) = \emptyset$. For any $t$, $t$ is not the greatest element of $T$, let $\tau = \inf \{m(t') | t' > t\}$. Since $T$ is a time structure, $\{t' | m(t') \geq \tau\}$ has a least element, denoted $t_0$. Since $T$ is discrete, $t_0 > t$. However, $(t, t_0) = \emptyset$.

   (b) $\rightarrow$ (a): Every point has a neighborhood including no other points but itself. So every point is an open (or closed) set. Therefore, $T$ is of discrete metric topology.

   (b) $\rightarrow$ (c): If $T$ is not well-founded, there is $t \in T, [0, t)$ is infinite. Therefore, $T = \{\text{suc}^n(0) | n \in N\} \subseteq [0, t) \subseteq T$. According to (1), $t_0 = \bigvee T \in T$. However there is no $t < t_0$ such that $(t, t_0) = \emptyset$, contradiction.
(c) → (b): For any \( t > 0 \), there exists \( t' < t \), \((t', t) = \emptyset \) since \([0, t)\) is finite. For any \( t, t' \) is not the greatest element, there exists \( t' > t \), \((t, t') = \emptyset \) since otherwise for any \( t' > t \), \([0, t')\) is infinite.

(3) (a) → (b) (Not Dense → Not Continuous): If \( T \) is not dense, there exist \( t_1 \) and \( t_2 \) such that \((t_1, t_2) = \emptyset \). Then \( T \) is separated (or not continuous) since \( T \) is the union of two disjoint, non-empty open sets \( \{t|m(t) < m(t_1) + d(t_1, t_2)/2\} \) and \( \{t|m(t) > m(t_2) - d(t_1, t_2)/2\} \). (b) → (a) (Not Continuous → Not Dense): If \( T \) is not continuous, \( T \) is the union of two disjoint, non-empty open (or closed) sets \( T_1 \) and \( T_2 \). Let \( \tau_1 = \sup\{m(t)|t \in T_1\} \) and \( \tau_2 = \inf\{m(t)|t \in T_2\} \). Since \( T \) is a time structure, \( \{t|m(t) \leq \tau_1\} \) has a greatest element \( t_1 \) and \( \{t|m(t) \geq \tau_2\} \) has a least element \( t_2 \). Since \( T_1 \) and \( T_2 \) are closed, \( t_1 \in T_1 \) and \( t_2 \in T_2 \). Therefore, \((t_1, t_2) = \emptyset \).

**Proposition 2.15** \( \perp_A \) is not \( \tau \)-open. The only neighborhood of \( \perp_A \) is \( A \).

**Proof:** According to the definition of topology. ■

**Proposition 2.16** For any domain, its partial order topology is finer than its derived metric topology, and both are non-Hausdorff.

**Proof:** Trivial. ■

**Proposition 2.17** (1) Function \( f : \Omega \times \overline{A} \rightarrow \overline{A'} \) is continuous in the partial order topology iff \( f \) is strict or constant. (2) If \( f : \Omega \times \overline{A} \rightarrow \overline{A'} \) is continuous in the derived metric topology, then \( f \) is continuous in the partial order topology. (3) Function \( f : \Omega \times \overline{A} \rightarrow \overline{A'} \) is continuous in the derived metric topology iff \( f \) is continuous in the partial order topology and the restriction of \( f \) on \( \Omega \times A \) and \( A' \) is continuous in the metric topology, namely, for any open subset \( S \) of \( A' \), \( f^{-1}(S) \cap (\Omega \times A) \) is open.

**Proof:** (1) If \( f \) is strict or a constant, \( f \) is continuous. If \( f \) is continuous and \( f \) is not strict, then \( f \) is constant since \( \perp_A \leq a \) implies that \( f(\omega, \perp_A) = f(\omega, a) \) for any \( a \) and any \( \omega \in \Omega \) if \( f(\omega, \perp_A) \neq \perp_{A'} \).

(2) If \( f \) is continuous in the derived metric topology, \( f \) is strict or constant, since \( \perp_A \) is a limit point of any \( \{a\} \) and \( f(\cdot, \perp_A) \) is a point or a limit point of \( \{f(\cdot, a)\} \).

(3) If \( f \) is strict or constant, and the restriction of \( f \) on \( A \) and \( A' \) is continuous in the metric topology, then \( f \) is continuous in the derived metric topology, since for any open set \( S \) of \( \overline{A'} \), \( f^{-1}(S) \) is open. If \( f \) is continuous in the derived metric topology, \( f \) is strict or constant, since in either case, the restriction of \( f \) on \( A \) and \( A' \) must also be continuous. ■
Proposition 2.18 Let $I$ be a finite index set. (1) Function $f : \Omega \times_I A_i \to A$ is continuous in the partial order topology iff $f$ is continuous w.r.t. all $i \in I$. (2) If $f : \Omega \times_I \overline{A_i} \to \overline{A}$ is continuous in the derived metric topology, then $f$ is continuous in the partial order topology. (3) Function $f : \Omega \times_I \overline{A_i} \to \overline{A}$ is continuous in the derived metric topology iff $f$ is continuous in the partial order topology and the restriction of $f$ on $\Omega \times_I A_i$ and $A$ is continuous in the product metric topology, namely, for any open subset $S$ of $A$, $f^{-1}(S) \cap (\Omega \times_I A_i)$ is open.

Proof: (1) For sake of simplicity, let $I = \{1, 2\}$. Note however that $I$ can be extended to any finite index set. If a function $f : \Omega \times A_1 \times A_2 \to A$ is continuous, it is right continuous since $V_A f(\omega, a, D) = f(\omega, a, \bigvee A_2 D)$. Similarly, if $f$ is left continuous. On the other hand, if $f$ is both left and right continuous, $f(\omega, \bigvee A_1 \times A_2 D) = f(\omega, \bigvee A_1 D_1, \bigvee A_2 D_2) = V_A f(\omega, D_1, \bigvee A_2 D_2) = V_A f(\omega, D_1, D_2) = V f(\omega, D)$ ([Hen88]).

(2) If $f : \Omega \times_I \overline{A_i} \to \overline{A}$ is continuous in the derived metric topology, $f$ is continuous in the derived metric space w.r.t. any argument $i \in I$, $f$ is continuous in the partial order w.r.t. any argument $i \in I$ (Proposition 2.17 (2)), $f$ is continuous in the partial order (Proposition 2.18 (1)).

(3) If $f$ is strict or constant, and the restriction of $f$ on $\times_I A_i$ and $A$ is continuous in the metric topology, $f$ is continuous in the derived metric topology, since for any open set $S$ of $\overline{A}$, $f^{-1}(S)$ is open. If $f$ is continuous in the derived metric topology, $f$ is strict or constant w.r.t. argument $i$ for all $i \in I$. In either case, the restriction of $f$ on $\times_I A_i$ and $A$ must also be continuous, since for any open set $S$ of $A$, either $f^{-1}(S) \subseteq \Omega \times_I A_i$ or the projection onto the $i$-th argument is $\overline{A_i}$ for any $i$. Therefore, $f^{-1}(S) \cap \Omega \times_I A_i$ is open. ■

Proposition 2.19 Let $\nu : \Omega \times L \to \overline{A}$ be a stochastic linear set of values. Then

(1) $F_{\nu} \to F_{\perp A}$, and

(2) $F_{\nu} \to F_{\nu_1}$ and $F_{\nu} \to F_{\nu_2}$ imply that either $F_{\nu_1} = F_{\nu_2}$ or one of $F_{\nu_1}$ and $F_{\nu_2}$ is $F_{\perp A}$.

Proof: (1) The only neighborhood of $\perp A$ is $\overline{A}$. Therefore, $u_{\omega}(l) \in N(\perp A)$ for all $l$ and $\omega \in \Omega$. (2) If $F_{\nu_1} \neq F_{\nu_2}$, then one of them must be $F_{\perp A}$, since the metric topology is Hausdorff with unique limits in distribution (Proposition 2.11, Proposition 2.13). ■

Proposition 2.20 For any time structure $T$ and any event trace $e_\omega$, $\langle T_{\omega}, d_\omega, \mu_\omega \rangle$ is a discrete sample time structure of $T$. 220
Proof: For any \( t_e \in T_e \) and \( 0 \leq \tau < \sup m(T_e) \), let \( T_e = \{ t'_e | m(t'_e) \leq \tau, t'_e \in T_e \} \) and \( T = \bigcup_{t'_e \in T_e} \{ t | t \leq t'_e \}. \) If \( T_e \) has no greatest element, \( T \) has no greatest element. Furthermore, \( e(T) \) is not defined, otherwise \( \exists t_0 \in T, e \) is constant on \( \{ t | t \geq t_0, t \in T \} \) and \( t_0 \) would be an upper bound of \( T_e \) in \( T_e \). However, if \( e(T) \) is not defined, there will be no \( t_e \in T_e \) with \( m(t_e) > \tau \), since \( e \) is non-intermittent. Therefore, for any \( t_e \in T_e \) and \( 0 \leq \tau < \sup m(T_e) \), \( T_e = \{ t'_e | m(t'_e) \leq \tau, t'_e \in T_e \} \) has a greatest element.

For any \( t_e \in T_e \) and \( 0 \leq \tau < \sup m(T_e) \), let \( T_e = \{ t'_e | m(t'_e) \geq \tau, t'_e \in T_e \} \) and \( T = \bigcup_{t'_e \in T_e} \{ t | t \geq t'_e \}. \) Let \( \tau' = \inf \{ m(t) | t \in T \} \) and \( t_0 \) be the least element of \( \{ t | m(t) \geq \tau' \}. \) If \( T_e \) has no least element, \( e(t_0) \) is not defined since \( e \) is right-continuous. However, since \( e \) is also non-intermittent, \( e(t) \) is not defined \( \forall t > t_0 \), contradiction. Therefore, for any \( t_e \in T_e \) and \( 0 \leq \tau < \sup m(T_e) \), \( T_e = \{ t'_e | m(t'_e) \geq \tau, t'_e \in T_e \} \) has a least element.

Therefore, \( T_e \) is a time structure.

For any \( t_e \in T_e, t_e > 0 \), let \( \text{pre}(t_e) = \{ t'_e | t'_e < t_e, t'_e \in T_e \} \) and \( T = \bigcup_{t'_e \in \text{pre}(t_e)} \{ t | t \leq t'_e \}. \) If \( \text{pre}(t_e) \) has no greatest element, \( T \) has no greatest element. Furthermore, \( e(T) \) is not defined, otherwise \( \exists t_0 \in T, e \) is constant on \( \{ t | t \geq t_0, t \in T \} \) and \( t_0 \) would be an upper bound of \( \text{pre}(t_e) \) in \( T_e \). However, if \( e(T) \) is not defined, \( e(t_e) \) will not be defined since \( e \) is non-intermittent. Therefore, \( \text{pre}(t_e) \) has a greatest element.

For any \( t_e \in T_e, t_e \) is not the greatest element of \( T_e \), let \( \text{suc}(t_e) = \{ t'_e | t'_e > t_e, t'_e \in T_e \} \) and \( T = \bigcup_{t'_e \in \text{suc}(t_e)} \{ t | t \geq t'_e \}. \) Let \( \tau = \inf \{ m(t) | t \in T \} \) and \( t_0 \) be the least element of \( \{ t | m(t) \geq \tau \}. \) If \( \text{suc}(t_e) \) has no least element, \( e(t_0) \) is not defined since \( e \) is right-continuous. However, since \( e \) is also non-intermittent, \( e(t) \) is not defined \( \forall t > t_0 \), contradiction. Therefore, \( \text{suc}(t_e) \) has a least element.

Therefore \( T_e \) is discrete. \( \blacksquare \)

**Proposition 2.21** Sampling and extending are continuous transductions

**Proof:** Let \( T \) be a time structure and \( T_r \) be a reference time structure of \( T \) with a reference time mapping \( h. \) Sampling is a transduction \( S_{\Omega,T,T_r} : A^{\Omega \times T_r} \rightarrow A^{\Omega \times T}. \) We prove that it is continuous.

Let \( D \subseteq A^{\Omega \times T_r} \) be directed and \( v^* \) be the least upper bound of \( D. \) Let \( y \) be \( S_{\Omega,T,T_r}(v). \)

\[
v^*(\omega, t) = v^*(\omega, h(t)) = \bigvee_{A} \{ y(\omega, h(t)) | v \in D \} = \bigvee_{A} \{ y(\omega, t) | v \in D \} = (\bigvee_{A^T} \{ y | v \in D \})(t).
\]

Therefore, \( \bigvee_{A^{\Omega \times T_r}} D = \bigvee_{A^{\Omega \times T}} D. \)
Similarly, extending is continuous since \( h^{-1}(t_r) = \{ t \mid m(t) \leq m_r(t_r) \} \) has a greatest element if \( \exists t \in T, \mu_r([0_r, t_r]) \leq \mu([0, t]) \) or \( \mu_r([0_r, t_r]) < \mu(T) \).

**Proposition 2.22** The partial order of a domain is a cpo.

**Proof:** A flat partial order is a cpo. The product partial order of cpos is a cpo. ■

**Proposition 2.23** The partial order of a stochastic trace space is a cpo.

**Proof:** The product partial order of cpos is a cpo. ■

**Proposition 2.24** The partial order of a stochastic event space is a cpo.

**Proof:** We first prove that the sub-partial order with the set of non-intermittent and right-continuous traces of a trace space is a cpo.

Let \( \mathcal{V} \subseteq \mathcal{A}^{T} \) be the set of non-intermittent and right-continuous traces on a simple domain. The least element in \( \mathcal{V} \) is \( \lambda t. \bot \) for \( \mathcal{A} \). The least upper bound of a directed subset \( D \) of \( \mathcal{V} \) is \( \bigvee_{D} D(t) \), which is also in \( \mathcal{V} \) for the following reasons: First, according to Proposition 3.4.4, \( (\bigvee_{D})(T) \geq \bigvee_{A} \bigvee_{T} D(T) \), if \( (\bigvee_{D})(T) \) is \( \perp \) for all \( d \in D \). Second, for any \( t \in T \), if \( (\bigvee_{D})(t) = \perp \), \( (\bigvee_{V} D) \) is right-continuous at \( t \); if \( (\bigvee_{V} D)(t) = a \in A \), there is \( d \in D \) such that \( d(t) = a \). Since \( d \) is right-continuous at \( t \), \( (\bigvee_{V} D) \) is right-continuous at \( t \).

Because of the composite properties of non-intermittent traces and limits, non-intermittent and right-continuous traces are closed under least upper bounds for traces on composite domains as well.

Therefore, the partial order of an event space is a cpo. ■

**Proposition 2.25** A transliteration \( f_T : \mathcal{A}^{\Omega \times T} \rightarrow \mathcal{A'}^{\Omega \times T} \) on any sample space \( \Omega \) and time structure \( T \) is continuous if \( f : \mathcal{A} \rightarrow \mathcal{A'} \) is continuous.

**Proof:** Let \( D \subseteq \mathcal{A}^{\Omega \times T} \) be directed, and \( v^* \) be the least upper bound of \( D \). We will prove that \( f_T(\bigvee_{\mathcal{A}^{\Omega \times T}} D) = \bigvee_{\mathcal{A}^{\Omega \times T}} f_T(D) \), i.e., for any \( \omega \) and \( t \), \( f_T(v^*)(\omega, t) = (\bigvee_{\mathcal{A'}^{T}} f_T(D))(\omega, t) \).

\[
f_T(v^*)(\omega, t) = f(v^*(\omega, t)) = f(\bigvee_{\mathcal{A}} \{ v(\omega, t) \mid v \in D \})
\]

\[
= \bigvee_{\mathcal{A'}} \{ f(v(\omega, t)) \mid v \in D \} \quad \text{since } f \text{ is continuous}
\]

\[
= \bigvee_{\mathcal{A'}^{T}} \{ f_T(v)(\omega, t) \mid v \in D \} = \bigvee_{\mathcal{A'}^{T}} f_T(D)(\omega, t). \]

Proposition 2.26 A unit delay on any discrete time structure is continuous.
Proof: Let \( D \subseteq A^{\Omega \times T} \) be directed and \( v^* \) be the least upper bound of \( D \). Since \( T \) is discrete, \( \text{pre}(t) \) has a greatest element, which is denoted by \( \text{pre}(t) \).

\[
\delta_T^A(\omega, v_0)(v^*)(t) = \begin{cases} v_0 & \text{if } t = 0 \\ v^*(\omega, \text{pre}(t)) = \bigvee_A \{v(\omega, \text{pre}(t)) | v \in D\} & \text{otherwise} \\
\end{cases}
\]

Proposition 2.27 A transport delay is continuous.

Proof: Similar to the proof of Proposition 3.6.5. Since \( T \) is a time structure, for any \( \tau > 0 \), \( t - \tau \) has a greatest element when \( m(t) \geq \tau \). ■

Proposition 2.28 An event-driven transduction \( F^o \) is continuous if its primitive transduction \( F \) on any discrete time structure is continuous.

Proof: The proof is divided into two steps. First, \( F^o \) is continuous w.r.t. the second argument if \( F \) is continuous on discrete time structures, since any event-based time is discrete, both sampling and extending are continuous (Proposition 2.21), and continuity is closed under functional composition. Second, if we can show that \( F^o \) is continuous w.r.t. the first argument, then, according to Proposition 2.18 (1), we will have showed that \( F^o \) is continuous. Therefore, let us now prove that \( F^o \) is continuous w.r.t. the first argument. Let \( T \) be any time structure and \( v \in A^{\Omega \times T} \) be fixed. For any directed subset \( D \) of \( \mathcal{E}^T \), \( D \) is a chain. According to the definition, \( F^o_T(D, v) \) is a chain too, i.e., a directed subset. Furthermore, for any \( t \) if \( (\bigvee_{\mathcal{E}^T} D)(t) \neq \bot_B \), there is \( d \in D \) such that for all \( t' \leq t \), \( d(t') = (\bigvee_{\mathcal{E}^T} D)(t') \), i.e., \( \bigvee_{A^T} F^o_T(D, v) \geq F^o_T(\bigvee_{\mathcal{E}^T} D, v) \). On the other hand, \( F^o_T \) is monotonic w.r.t. the first argument, i.e., \( \bigvee_{A^T} F^o_T(D, v) \leq F^o_T(\bigvee_{\mathcal{E}^T} D, v) \). Therefore, \( \bigvee_{A^T} F^o_T(D, v) = F^o_T(\bigvee_{\mathcal{E}^T} D, v) \), it is continuous w.r.t. the first argument. ■

Theorem 2.1 Let \( A \) be a \( \Sigma \)-domain structure and \( T \) a time structure. The \( \Sigma \)-dynamics structure \( D(T, A) = (\mathcal{V}, \mathcal{F}) \) satisfies (1) \( \mathcal{V} \) is a multi-sorted set of \( cpos \) and (2) transliterations, transport delays and event-driven transductions in \( \mathcal{F} \) are continuous in the partial order topology. If, in addition, \( T \) is discrete, all transductions in \( \mathcal{F} \) are continuous in the partial order topology.

Proof: Follows from Propositions 2.22 – 2.28. ■

Proposition 2.29 A transliteration \( f_T \) is well-defined iff function \( f \) is well-defined; \( f_T \) is strict w.r.t. an argument iff \( f \) is strict w.r.t. the argument.
Proof: According to the definitions of well-definedness and strictness. □

Proposition 2.30 Any delay is not strict. A unit delay on any discrete time structure is well-defined. A transport delay is well-defined.

Proof: According to the definitions of well-definedness and strictness. ■

Proposition 2.31 An event-driven transduction $F^\circ$ is well-defined iff $F$ on any discrete time structure is well-defined; $F^\circ$ is strict w.r.t. its event input, and $F^\circ$ is strict w.r.t. one of the other input arguments iff $F$ is strict w.r.t. the argument.

Proof: Event-based time is discrete, and sampling and extending are well-defined. ■

Proposition 2.32 A transliteration $f_T$ is right-continuous if $f$ is continuous in the derived metric topology; $f_T$ with $f : \times I A_i \to \overline{A}$ is non-intermittent if $f$ is strict, well-defined and continuous in the derived metric topology.

Proof: For any neighborhood $N(f(v(t))))$, there is a neighborhood $N(v(t)))$, such that $x \in N(v(t)))$ implies $f(x) \in N(f(v(t))))$. For any neighborhood $N(v(t)))$, there is $T = (t, t'), t'' \in T$ implies $v(t'') \in N(v(t)))$. Therefore, for neighborhood $N(f(v(t)))$, there is $T = (t, t'), t'' \in T$ implies $f(v(t'')) \in N(f(v(t)))$.

If $f$ is strict and well-defined, $v(t)$ is well-defined implies that $f_T(v)(t)$ is well-defined, $v(t)$ is not well-defined implies that for all $t' \geq t$ $f_T(v)(t')$ is undefined. If, in addition, $f$ is continuous in the derived metric topology, $\lim v_T$ is well-defined implies that $\lim f(v)_{|T}$ is well-defined, $\lim v_T$ is not well-defined implies that $\lim f(v)_{|T}$ is undefined. ■

Proposition 2.33 A delay is non-intermittent. A transport delay is right-continuous. Proof: The output of a delay is non-intermittent if its input is non-intermittent. The output of a transport delay is right-continuous if its input is right-continuous. ■

Proposition 2.34 An event-driven transduction is right-continuous. An event-driven transduction $F^\circ$ is non-intermittent if $F$ is non-intermittent.

Proof: Any trace on a discrete time structure is right-continuous. Any extension of a discrete-time trace is right-continuous. Both sampling and extending are non-intermittent and non-intermittent transductions are closed under functional composition. ■
A.2 Probabilistic Constraint Nets

**Theorem 3.1** Let $A$ be a cpo and assume that either $A$ is also a total order or that the set of distributions over $A$ is a cpo and the function over distributions is continuous. Then, every continuous function $f : \Omega \times A \rightarrow A$ or pathwise continuous function $f_\omega : A \rightarrow A$ (for a fixed $\omega \in \Omega$) has a least fixpoint in distribution.

**Proof:**

When $A$ is a total order,

To prove this results, we will use the classic Tarski’s fixpoint theorem (Lattice-theoretical fixpoint theorem) [Tar55]. Let us first introduce the theorem and then show how to use the result to prove our Fixpoint Theorem.

**Theorem A.1 (Tarski’s Fixpoint Theorem)** Let

1. $\mathcal{A} = (A, \leq)$ be a complete lattice,
2. $f$ be a monotonically increasing function on $A$ to $A$,
3. $P$ be the set of all fixpoints of $f$.

Then the set $P$ is not empty and the system $(P, \leq)$ is a complete lattice.

**Proof:** (Theorem A.1) For the proof of this well-known result the reader is referred to the original work from Tarski [Tar55].

In order to be able to use Tarski’s results, we need to show that the set of distributions and its partial order define a complete lattice. Moreover, we also need to show that the function $f$ on the set of distribution is monotonically increasing.

First, denote the set of all distributions on $A$ by $\mathcal{D}$. We formally define a partial order on the set of distributions $\mathcal{D}$. The binary relation $\leq_D$ on $\mathcal{D}$ is defined as follow. Let $F_{X_1}$ and $F_{X_2}$ be distributions of two random variables, namely $X_1$ and $X_2$. We write $F_{X_1} \leq_D F_{X_2}$, if $\forall a \in A$, $Pr(X_1 \leq a) \geq Pr(X_2 \leq a)$. It is easy to show that $\leq_D$ induces a partial order on $\mathcal{D}$.

Second, we need to show that for any two distributions $F_1, F_2 \in \mathcal{D}$, there exists a least upper bound and a greatest lower bound. To prove this, let us look at the cumulative distribution function (cdf) of each distributions. Here we reproduce Theorem 1.5.1 of [CB90] and refer to this reference for the proof.
Theorem A.2 (Theorem 1.5.1 of Casela and Berger)

The function \( F(x) \) is a cdf if and only if the following three conditions hold:

1. \( \lim_{x \to -\infty} F(x) = 0 \) and \( \lim_{x \to \infty} F(x) = 1 \).

2. \( F(x) \) is a nondecreasing function of \( x \).

3. \( F(x) \) is right-continuous. That is, for every number \( x_0 \), \( \lim_{x \uparrow x_0} F(x) = F(x_0) \).

Proof: (Theorem A.2) See p. 30 of §1.5 from [CB90].

Since \( A \) is a total order, for each \( F \in \mathcal{G} \), we have a well defined cdf. Hence every \( F \in \mathcal{G} \) possess the three properties of a formal cdf. Based on these properties, it is easy to show that the upper envelope of any set of cdf is a least upper bound (LUB) while the lower envelope of the cdfs is the greatest lower bound (GLB). Moreover, both the LUB and GLB can be showed to be cumulative distribution functions since they are nondecreasing, right-continuous and converge to 0 and 1 as \( x \downarrow -\infty \) and \( x \uparrow \infty \) respectively. This demonstrate that we have a complete lattice.

Now let us show that \( f \) applied recursively generates a sequence of monotonically increasing distributions. First assume, without loss of generality, that \( f \) is Markovian. Moreover, let us assume that at each transition, the events \( \{\omega_1, \cdots, \omega_n\} \) are independent and chosen from the sample space \( \Omega \). Order the events \( \{\omega_1, \cdots, \omega_n\} \) such that \( f_{\omega_1}(a) \leq_A f_{\omega_2}(a) \leq_A \cdots \leq_A f_{\omega_n}(a) \) for any \( a \in A \). Let \( \perp_A \) denote the least element of \( A \) and let \( F_X \) denote the distribution of the random variable \( X \).

Now we want to prove that \( F_{f^n(\perp_A)} \leq_D F_{f^{n+1}(\perp_A)} \).

Proof by Induction on \( n \):

- For \( n = 0 \): We have that \( F_{f^0(\perp_A)} = \perp_A \).

- For \( n = 1 \): From Proposition 2.9, since \( f_\omega \) is continuous, we have that \( f_\omega \) is also monotonic, \( \forall \omega \in \Omega \). Hence we have \( f_\omega(\perp_A) \geq_A \perp_A, \forall \omega \in \Omega \). Therefore, it is trivial to prove that \( F_{f^0(\perp_A)} = F_{\perp_A} \leq_D F_{f^1(\perp_A)} \).

- Induction Hypothesis: Assume that for an arbitrary chosen \( n \in \mathbb{N} \), \( F_{f^n(\perp_A)} \) exists and is well-defined. We now need to show that \( F_{f^n(\perp_A)} \leq_D F_{f^{n+1}(\perp_A)} \).

Let \( M_1 = \max \{f^n(\perp_A)\} = f_{\omega_1} \circ \cdots \circ f_{\omega_n}(\perp_A) \) Based on this definition, we have \( P(f^n(\perp_A \text{ n times}) \leq M_1) = 1 \).
It is easy to show that \( f_{\omega_n} \circ \cdots \circ f_{\omega_0} \circ f_{\omega_i} \geq M_1 \) since \( f_{\omega_n} \circ \cdots \circ f_{\omega_n} \) is monotonic. Hence, we get the following result:

\[
P(f^{n+1}(\perp_A) \leq M_1) \leq 1 - \sum_{i=1}^{n} P(\text{event } \omega_n \cdots \omega_0(\omega_i)) \leq 1 - \frac{P(\text{event } \omega_n \cdots \omega_0(\omega_i))}{\text{finite and } > 0} \leq 1 - P^n(\omega_n) \leq 1.
\]

Let \( M_2 = \max\{\{f^n(\perp_A)\} - M_1\} \) be the second highest value after \( n \) iterations.

Say that \( M_2 \) arose from \( f_{\omega^*}(\perp_A) \) where \( \omega^* \in \omega_1^* \omega_2^* \cdots \omega_n^* \) with \( \omega_i \in \Omega \). Then,

\[
P(f^n(\perp_A) \leq M_2) \leq 1 - P(\omega_n \cdots \omega_n) = 1 - P^n(\omega_n)
\]

(A.2)

Based on the same reasoning as above, we know that \( f_{\omega^*} \circ f_{\omega_i}(\perp_A) \geq f_{\omega^*}(\perp_A) \). Therefore, we have

\[
P(f^{n+1}(\perp_A) \leq M_2) \leq 1 - \frac{P^n(\omega_n)}{> 0} - \frac{P(\omega^*)}{> 0} \leq 1 - P^n(\omega_n)
\]

(A.3)

By applying this reasoning until \( M_n = \min\{f^n(\perp_A)\} = f_{\omega_1} \circ \cdots \circ f_{\omega_1}(\perp_A) \), we get

\[
P(f^n(\perp_A) \leq M_n) = P^n(\omega_1)
\]

We know that \( f_{\omega_1} \circ \cdots \circ f_{\omega_1} \circ f_{\omega_1}(\perp_A) \geq f_{\omega_1} \circ \cdots \circ f_{\omega_1}(\perp_A) = M_n \), which means that

\[
P(f^{n+1}(\perp_A) \leq M_n) = 0.
\]

We have proven that \( F_{f^n(\perp_A)} \leq F_{f^{n+1}(\perp_A)} \) for any value of \( n \in \mathbb{N} \). Hence, the transformational process on the distributions is a monotonically increasing one.

By applying Tarski’s theorem, we have that the set of fixpoints of the distributions is non-empty and is a complete lattice. Therefore, there exist a least fixpoint in distribution and it concludes the proof under the assumption that \( A \) is a total order. ■
When the set of distributions over $A, \mathcal{D}$, is a cpo and the function over $\mathcal{D}$ is continuous.

To prove this result, we can simply claim the following fixpoint theorem on cpos:

**Theorem A.3** Let $(A, \leq, \bot_A)$ be a cpo with least element $\bot_A$. Let $f : (A, \leq, \bot_A) \rightarrow (A, \leq, \bot_A)$ be a continuous function and let $\mu.f$ be the least upper bound of the chain $\{f^n(\bot_A) | n \in \mathbb{N}\}$. Then $\mu.f$ is the least fixpoint of $f$.

**Proof:** (Theorem A.3) The proof can be found in any elementary algebraic theory textbooks such as [Hen88].

This concludes the proof of the Fixpoint Theorem under the assumption that the set of distributions over $A, \mathcal{D}$, is a cpo and the function over $\mathcal{D}$ is continuous since we have satisfied all the necessary assumptions of the fixpoint theorem on a cpo.

**Theorem 3.2** Let $A$ and $A'$ be two cpos and assume that either $A, A'$ are also total orders or that the set of distributions over $A'$ is a cpo and the function over distributions is continuous. If $f : \Omega \times A \times A' \rightarrow A'$ is a continuous function, then there exists a unique continuous function $\mu.f : \Omega \times A \rightarrow A'$, such that for all $a \in A$, the distribution of $(\mu.f)(a)$ is the least fixpoint in distribution of $\lambda \omega, x.f_\omega(a, x)$.

**Proof:** Let $F^0(a) = f(\omega, a, \bot_{A'})$ and $F^{k+1}(a) = f(\omega, a, F^k(a))$. Since $f$ is continuous, it is continuous w.r.t. the third argument. Moreover, a continuous function in any partial order is also monotonic. Therefore, for every $a$,

$$F^0(a) \leq_A F^1(a) \leq_A \cdots \leq_A F^k(a) \leq \cdots$$

The proof of the existence of the least fixpoint $\mu.f$ is left to the reader as it is very similar to the proof of Theorem 3.1.

Next, we prove that $\mu.f$ is continuous.

Clearly for every $k$, $F^k$ is continuous since $f$ is continuous and continuity is closed under
functional composition. Therefore, for any directed subset $D$ of $A$, 

$$
\mu_f(\bigvee_D) = \bigvee_{A'} \{ F^k(\bigvee_D) | k \geq 0 \}
$$

$$
= \bigvee_{A'} \bigvee_{A'} \{ F^k(D) | k \geq 0 \}
$$

$$
= \bigvee_{A'} \bigvee_{A'} \{ F^k(a) | k \geq 0, a \in D \}
$$

$$
= \bigvee_{A'} \mu_f(D).
$$

**Proposition 3.1** Let $I \subseteq J$ be an index set. If $f : \Omega \times (\times_i A_i) \to A$ is a continuous or pathwise continuous function, then the extension of $f$, $f' : \Omega \times (\times_i A_i) \to A$ satisfying $f'(\omega, a) = f(\omega, a_{|I})$, is a continuous or pathwise continuous function.

**Proof:** According to the definitions of continuous functions and product topologies. ■

**Proposition 3.2** Here are some properties that can be inferred from modules operators. The semantics of a composite module can be derived from the semantics of the components with which it was constructed:

- **Cascade connection:** If $PCN(I, O) = PCN_2(I_2, O_2) \circ PCN_1(I_1, O_1)$, then

  $$[PCN(I, O)] = \{ F_2 \circ F_1 | F_1 \in [PCN_1(I_1, O_1)], F_2 \in [PCN_2(I_2, O_2)] \}.$$

- **Parallel connection:** If $PCN(I, O) = PCN_1(I_1, O_1) + PCN_2(I_2, O_2)$, then

  $$[PCN(I, O)] = \{ (F_1, F_2) | F_1 \in [PCN_1(I_1, O_1)], F_2 \in [PCN_2(I_2, O_2)] \}$$

where $(F_1, F_2)_{|O_1}(i) = F_1(i_{|I_1})$ and $(F_1, F_2)_{|O_2}(i) = F_2(i_{|I_2}).$

- **Feedback connection:** If $PCN'(I', O') = F(PCN(I, O))$, then

  $$[PCN'(I', O')] = \{ \mu.F | F \in [PCN(I, O)] \}$$

where $\mu.F$ is the the least fixpoint of $F$.

- **Union:** If $PCN(I, O) = PCN_1(I_1, O_1) \parallel PCN_2(I_2, O_2)$, then

  $$[PCN(I, O)] = [PCN_1(I_1, O_1)] \times [PCN_2(I_2, O_2)].$$

229
Proof: Based on the definition of the semantics of modules.

Proposition 3.3 If \(CN_1(I_1, O_1)\) and \(CN_2(I_2, O_2)\) are well-defined modules, then the following resulting modules:

- **Union Connection:** \(PCN_1(I_1, O_1) \parallel PCN_2(I_2, O_2)\),
- **Cascade Connection:** \(PCN_1(I_1, O_1) \circ PCN_2(I_2, O_2)\),
- **Parallel Connection:** \(PCN_1(I_1, O_1) + PCN_2(I_2, O_2)\),

are well-defined modules.

Proof: According to the definition of the well-definedness of modules.

Proposition 3.4 Let \(A\) and \(A'\) be two cpos and assume that either \(A\) and \(A'\) are also total orders or that the set of distributions over \(A'\) is a cpo and the function over distributions is continuous. If \(f : \Omega \times A \times A' \rightarrow A'\) is a strict continuous function w.r.t. its third argument \((a' \in A')\), then the least fixpoint of \(f\), or the least solution of the equation \(o = f(o, i, o)\), is undefined.

Proof: \(\mu.f = \lambda \omega, x. \bot_{A'}\)

Proposition 3.5 A module \(PCN(I, O)\) is not well-defined if there is an output location \(l \in O\) such that \(PCN\) has an algebraic loop on \(l\).

Proof: If \(l \rightarrow l\) then \(l\) results in an undefined stochastic trace.

### A.3 Modeling in PCN

**Theorem 4.1** Let \(\rho\) and \(\tau\) be two \(\mathcal{F}_t\)-adapted stopping times with \(t_0 \leq \rho \leq \tau \leq T < \infty\) (a.s.). Assume \(V : [0, T] \times \mathbb{R}^d \rightarrow \mathbb{R}^k\). Take any hierarchical set \(Q \in \mathcal{M}_{0,m}\). Then, each Itô SDE with coefficients \(f, g^j\) possesses a Itô-Taylor expansion of Equation 4.5 with respect to the hierarchical set \(Q\), provided that all derivatives of \(V, f, g^j\) (related to \(Q\)) exist.

Proof: The reader is referred to [KP99] for the proof using the Itô formula and induction on the maximum length \(\sup_{\alpha \in Q^l(\alpha)} \in \mathbb{N}\). Note that a similar expansion holds for Stratonovich SDEs as well.

**Proposition 4.1** Given a probabilistic constraint net made of the stochastic differential equations \(\dot{x}_k = f_k(\bar{x}) + g_k(\bar{x})N_t, k = 1, \ldots, n\) with \(x_k(t_0) \in \mathbb{R}\) and \(f_k, g_k : \mathbb{R}^n \rightarrow \mathbb{R}\) as partial or total functions, and given that all \(f_k\) and \(g_k\) are sufficiently smooth at \(\bar{x}(t_0)\), the limiting semantics of
the probabilistic constraint net, based on the Euler-Maruyama method, is well-defined over $T = [t_0, t_1]$ for some $t_1 > t_0$. In particular, the results of Theorem 4.1 apply for the smooth function $V(t, x) = x$. The reader should note that if $f_k$ and $g_k$ are polynomial functions, then both $f_k$ and $g_k$ are smooth over $\mathbb{R}^n$ and hence this results holds.

Proof: If all $f_k$ and $g_k$ are smooth enough, $x^{(n)}(t_0)$ exists, the semantics results in a Itô-Taylor expansion.

A.4 Behavioural Verification with Average Timed $\forall$-Automaton

Proposition 7.1 Assume a finite state space $S$, and assume that, for all the bad states $B$, there is a positive probability of moving toward an accepting state $R$ or $S$, i.e., the set of bad states is irreducible. Define $\zeta_s^{s'}$ as the time needed to reach state $s'$ from state $s$. Then, $E(\zeta_s^{s'} | b \in B, s \in R \cup S) < \infty$.

Proof: Let $p > 0$ be the smallest probability with which a bad state $b \in B$ moves toward an accepting state. Due to the irreducibility of $B$, we are guaranteed that $p > 0$ exists. We can show that the measure of paths starting in a bad state and never reaching the set of accepting states is $\lim_{t \to \infty} (1 - p)^t = 0$. Hence, any path which never reaches the set of accepting states has measure 0. Let us now prove that $E(\zeta_s^{s'} | s \in B, s' \in R \cup S) < \infty$. Assume that state $s$ is located at $n$ transitions away from $s'$, the closest $R \cup S$-state. Hence,

$$E(\zeta_s^{s'}) = \sum_{i=n}^{\infty} ip^n(1-p)^{i-n}$$

$$= \frac{p^n}{(1-p)^{n-1}} \sum_{i=n}^{\infty} i(1-p)^{i-1}$$

$$= \frac{p^n}{(1-p)^{n-1}} \frac{\partial}{\partial p} \left( \sum_{i=0}^{\infty} (1-p)^i \right)$$

$$\leq \frac{p^n}{(1-p)^{n-1}} \frac{\partial}{\partial p} \left( \frac{1}{1-(1-p)} \right)$$

$$= \frac{p^n}{(1-p)^{n-1}} \frac{1}{(1-(1-p))^2}$$

$$< \infty$$

Proposition 7.2 Observe a PCN model at the finest level of details, i.e., looking at the network created by the connections between locations and basic transduction (transliterations, delays and generators). A PCN model induces a Markovian behaviour of order $n$ iff all the delays, for which
the output location is included in the state space, are of finite length of at most $n$ or if the sum of the strictly consecutive delays with output location not included in the state space is at most $n$.

**Proof:** By looking at the PCN model at the finest level of details, we obtain a network composed only of locations and basic transductions. If all the output locations of the delays are included in the state space, then for each of these locations, the system must remember an history of up to $n$ elements, hence being Markovian of order $n$.

For $m$ delay transductions connected in series, with respective delay $\delta_i = n_i, i = 1, \ldots, m$, for which the intermediate locations are not part of the state space, the value at time $t$ of the final location in the series is the value of the input location at time $t - \sum_{i=1}^{m} \delta_i$. Hence for a system to be Markovian of order $n$, we must have $\sum_{i=1}^{m} \delta_i = \sum_{i=1}^{m} n_i \leq n$.

**Proposition 7.3** Let $\{\alpha_q\}_{q \in Q}$ be invariants for $B$ and $A$. If $r$ is a run of $A$ over a trace $v \in B$, then $\forall t \in T, v(t) \models \alpha_r(t)$.

**Proof:** For any trace $v$, $v(0)$ is an initial state, therefore, $v(0) \models \Theta$. Furthermore, $r$ is a run, $v(0) \models e(r(0))$. Therefore, $v(0) \models e(r(0)) \land \Theta$. Since $e(r(0)) \land \Theta \rightarrow \alpha_r(0)$, we have $v(0) \models \alpha_r(0)$.

Assume that $v(pre(t)) \models \alpha_{r(pre(t))}$. Therefore, $v(t) \models c(r(pre(t)), r(t)) \rightarrow \alpha_r(t)$ since $v(pre(t)) \sim v(t)$. In addition, $v(t) \models c(r(pre(t)), r(t))$. Therefore, $v(t) \models \alpha_r(t)$.

A set equipped with a well-founded relation is said to be a well-founded set. A well-founded set is a partially ordered set which contains no infinite descending chains, or equivalently, a partially ordered set in which every non-empty subset has a minimal element. If the order is a total order then the set is called a well-ordered set. Since by definition, time is a linearly ordered set with $0$ as a least element, $T$ is then a well-founded set. Using the technique of mathematical induction for well-founded sets, we can conclude that $v(t) \models \alpha_r(t)$ for all $t$.

**Proposition 7.4** Let $\{\alpha_q\}_{q \in Q}$ be a set of invariants for $B$ and $A$. Let $r$ be a run of $A$ over a trace $v \in B$. Also, let $\mathcal{V}_B = \{r \mid r$ is a run of $A$ over $v \in B\}$ be the set of runs induced by $B$. If $\{\rho_q\}_{q \in Q}$ is a set of Lyapunov functions for $B$ and $A$, then

- $\forall t \in T, \exists r', u^*(\rho_{r'}(v^*(t))) \leq \rho_{r(pre(t))}(v(pre(t)))$, where $r^*$ and $v^*$ denote all $r' \in \mathcal{V}_B$ and $v' \in B$ such that $v(pre(t)) \sim v'(t)$ and $c(r(pre(t)), r'(t))$, when $r(pre(t)) \in S$;

- $\exists \epsilon > 0, \forall t \in T, \exists v' \in B, \exists r' \in \mathcal{V}_B, ([\rho_{r'}(v'(t)) - \rho_{r(pre(t))}(v(pre(t))) \leq -\epsilon \land (v(pre(t)) \sim}$
\( v'(t) \land c(r(pre(t)), r'(t)) \) when \( r(pre(t)) \in B \).

**Proof:** Obtained from the conditions of the Lyapunov functions. ■

**Theorem 7.1** Let \( \{\alpha_q\}_{q \in Q} \) be a set of invariants for \( B \) and \( A \). Let \( r \) be a run of \( A \) over a trace \( v \in B \). If \( \{\rho_q\}_{q \in Q} \) is a set of Lyapunov functions for \( B \) and \( A \), then

- if \( BS \) is the set of segments of consecutive \( B \)-states and \( S \)-states in \( r \), then \( \forall q^* \in BS, q^* \) has a finite number of \( B \)-states;

**Proof:** From the conditions of the Lyapunov functions, we are assured that a transition from any \( B \)-states will eventually lead to a state with smaller Lyapunov value. Moreover, from these conditions and due to the fact that the Lyapunov functions are defined on a well-founded set, we are also assured that any segments containing only \( B \)-states and \( S \)-states, the system will eventually finish in an absorbing \( S \) or will leave to an \( R \)-state, both cases being satisfactory, hence completing the proof. ■

**Proposition 7.5** Let \( \{\alpha_q\}_{q \in Q} \) be a set of invariants for \( B \) and \( A \), and \( r \) be a run of \( A \) over a trace \( v \in A \). If there exist local timing functions, \( \{\gamma_q\}_{q \in T} \), and global timing functions, \( \{\eta_q\}_{q \in Q} \), for \( B \) and \( A TA \), then

1. if \( Sg(q) \) is the set of segments of consecutive \( q \)'s in \( r \), then \( \forall q \in T, q^* \in Sg(q), E(\mu(q^*)) \leq \tau(q) \), and
2. if \( BS \) is the set of segments of consecutive \( B \) and \( S \)-states in \( r \), then \( \forall q^* \in BS, E(\mu_B(q^*)) \leq \tau(bad) \).

**Proof:**

1. Let \( \{s_i\}_{i=1}^n \) be a sequence of \( q \)-states.

   From condition \( ATL1 \), we know that \( \forall q \in T, \alpha_q \rightarrow \lambda \leq \gamma_q \leq \tau(q) \). Therefore we can also deduce that:

   (1) \( E(\lambda(s_n)) \leq E(\gamma_q(s_n)) \) and
   (2) \( E(\gamma_q(s_n)) \leq E(\tau(q)) = \tau(q) \).
From condition (ETL2), we get:

\[ E(\gamma_q(s_2) - \gamma_q(s_1)) \leq -E(\lambda(s_1)) \]
\[ E(\gamma_q(s_3) - \gamma_q(s_2)) \leq -E(\lambda(s_2)) \]

... \[ E(\gamma_q(s_n) - \gamma_q(s_{n-1})) \leq -E(\lambda(s_{n-1})) \]

adding these inequalities, we have

\[ E(\gamma_q(s_n)) - E(\gamma_q(s_1)) \leq -\Sigma_{i=1}^{n-1} E(\lambda(s_i)). \]

Using (1) and (2) above we get:

\[ \Sigma_{i=1}^{n} E(\lambda(s_i)) \leq \tau(q), \]
\[ E(\Sigma_{i=1}^{n} \lambda(s_i)) \leq \tau(q), \]
\[ E(\mu(q^*)) \leq \tau(q). \]

2. Let \( \{s_i\}_{i=1}^{n} \) be a sub-sequence of B-states in a BS segment. Similarly to the proof above (using (ATG4) this time), we get:

\[ E(\eta_{q_1}'(s_1') - \eta_{q_1}(s_1)) \leq -E(\lambda(s_1)) \]
\[ E(\eta_{q_2}'(s_2') - \eta_{q_2}(s_2)) \leq -E(\lambda(s_2)) \]

... \[ E(\eta_{q_n}'(s_n') - \eta_{q_n}(s_n)) \leq -E(\lambda(s_n)) \]

and

\[ E(\eta_{q_1}'(s_1')) \geq E(\eta_{q_{i+1}}(s_{i+1})) \]

we have

\[ E(\eta_{q_n}'(s_n')) - E(\eta_{q_1}(s_1)) \leq -\Sigma_{i=1}^{n} E(\lambda(s_i)). \]
Finally, from (ATG2) we get that $E(\eta_{q_1}(s_1)) \leq \tau(bad)$ and $E(\eta_{q_n}(s'_n)) \geq 0$. Therefore,

$$\begin{align*}
\Sigma_{i=1}^n E(\lambda(s_i)) &\leq \tau(bad) \\
E(\Sigma_{i=1}^n \lambda(s_i)) &\leq \tau(bad) \\
E(\Sigma_{i=1}^n \mu(q^*)) &\leq \tau(bad)
\end{align*}$$

Theorem 7.2 For any state-based and time-invariant behaviour $B$ with an infinite time structure and a complete average-timed $\forall$-automaton $ATA$, the verification rules are sound and complete, i.e., $B \models ATA$ iff there exist a set of invariants, Lyapunov functions and timing functions.

Proof: (Verification Rules)

**Soundness ($\Leftarrow$)**

The construction of these rules guarantees the soundness of the verification method. For any trace $v$, there is a run because $ATA$ is complete. For any run $r$ over $v$, if any automaton state in $R$ appears infinitely many times in $r$, $r$ is accepting. Otherwise, there is a time point $t_0 \in T$, the sub-sequence $r$ on $I = \{t \in T | t \geq t_0\}$, denoted $q^*$, has only bad and stable automaton states. From the results of Proposition 7.4, if there exist a set of invariants and a set of Lyapunov functions, $q^*$ has only a finite number of $B$-states. Since time is infinite, all the automaton states appearing infinitely many times in $r$ belong to $S$; so $r$ is accepting too. Therefore, every trace is accepting for the automaton. If there exists a set of local and global timing functions, every trace satisfies the timing constraints on average.

**Completeness ($\Rightarrow$)**

On the other hand, if $ATA$ is valid over $B$, then to prove completeness, we need to show that there exist a set of invariants, a set of Lyapunov functions, and a set of local and global timing functions that satisfy the requirements. We will use a constructive proof which in turn will be used later when introducing the verification algorithm.

For any state $s \in S$ and state proposition $\alpha$, we write $\alpha(s)$ iff $s \models \alpha$. It is possible to construct the invariants by choosing them as the fixpoint of the set of equations:

$$\alpha_{q'}(s') = (\exists q, s, \alpha_q(s) \land (s \leadsto s') \land c(q, q')(s')) \lor (\theta(s') \land e(q')(s')).$$

235
We can verify that \( \{ \alpha_q \}_{q \in Q} \) is a set of propositions over \( S_B \) and satisfies the requirements of initiality and consecution. The reader is referred to [AP86, SDG84] for a formal argument showing that the invariants can be obtained via the fixpoints of the above equations. Furthermore, \( s \models \alpha_q \iff \langle q, s \rangle \) is a reachable pair for \( A \mathcal{T} A \) and \( B \).

Given the constructed invariants \( \{ \alpha_q \}_{q \in Q} \), a set of Lyapunov functions \( \{ \rho_q \}_{q \in Q} \) and a set of global timing functions \( \{ \eta_q \}_{q \in Q} \) can be constructed as follows:

- \( \forall q \in R, s \models \alpha_q \), let \( \rho_q(s) = 0 \) and \( \eta_q(s) = 0. \)

- \( \forall q \not\in R, s \models \alpha_q, \rho_q(s) \) and \( \eta_q(s) \) are defined as follows. Construct a weighted directed graph \( G = \langle V, E, W \rangle \), where \( W \) is the set of weights corresponding to the transition probabilities, such that \( \langle q, s \rangle \in V \iff q \not\in R, s \models \alpha_q, \) and \( \langle q, s \rangle \leadsto \langle q', s' \rangle \) in \( E \iff s \leadsto s' \land c(q, q')(s'). \)

For any \( \langle q, s \rangle \in V \), let \( \mathbb{E}(P_B) \) be the average number of \( B \)-states in the set of paths \( P \) starting from \( \langle q, s \rangle \) and \( \mathbb{E}(B(P)) \) be the average measure of \( B \)-states in \( P \). Let \( \rho_q(s) = \mathbb{E}(P_B) \) and \( \eta_q(s) = \mathbb{E}(B(P)) \).

We can verify that \( \{ \rho_q \}_{q \in Q} \) is a set of Lyapunov functions, and that \( \{ \eta_q \}_{q \in Q} \) is a set of global timing functions.

Similarly, a set of local timing functions \( \{ \gamma_q \}_{q \in T} \) can be constructed as follows. For all \( q \in T \), construct a weighted directed graph \( G = \langle V, E, W \rangle \), such that \( s \in V \iff s \models \alpha_q, \) and \( s \leadsto s' \) in \( E \iff s \leadsto s' \land c(q, q')(s'). \)

For paths \( P \) starting at \( s \), let \( \mathbb{E}(P) \) be the average measure of the path. Let \( \gamma_q(s) = \mathbb{E}(P) \). We can verify that \( \{ \gamma_q \}_{q \in T} \) is a set of local timing functions.

**Proposition 7.6** Let \( \{ \alpha_q \}_{q \in Q} \) be invariants for \( B \) and \( A \). If \( r \) is a run of \( A \) over \( v \in B, \forall t \in T, v(t) \models \alpha_r(t) \).

**Proof:** In order to prove this proposition, we shall introduce a variation of the method of continuous induction [Khi61]. A property \( \Gamma \) is *inductive* on a time structure \( T \) iff for all \( t_0 \in T, \Gamma \) is satisfied at all \( t < t_0 \) implies that \( \Gamma \) is satisfied at \( t_0 \). \( \Gamma \) is *continuous* iff \( \Gamma \) is satisfied at a non-greatest element \( t \in T \) implies that \( \exists t' > t, \forall t < t' < t', \Gamma \) is satisfied at \( t' \). Note that when \( T \) is discrete, any property is continuous. The theorem of continuous induction [Khi61] says:

**Theorem A.4 (Continuous Induction)** If the property \( \Gamma \) is inductive and continuous on a time structure \( T \) and \( \Gamma \) is satisfied at 0, \( \Gamma \) is satisfied at all \( t \in T \).
Proof: (Theorem A.4: Continuous Induction) We call a time point \( t \in T \) regular iff \( T \) is satisfied at all \( t' \), \( 0 < t' \leq t \). Let \( T \) denote the set of all regular time points. \( T \) is not empty since \( T \) is satisfied at \( 0 \). We prove the theorem by contradiction, i.e., assume that \( T \) is not satisfied at all \( t \in T \). Therefore, \( T \subset T \) is bounded above; let \( t_0 = \bigvee T \in T \) be the least upper bound of \( T \) (\( t_0 \) exists according to Proposition 3.2.1). Since \( t_0 \) is the least upper bound, it follows that \( T \) is satisfied at all \( t, 0 \leq t < t_0 \). Since \( T \) is inductive, it is satisfied at time \( t_0 \). Therefore, \( t_0 \in T \).

Since \( T \subset T \), \( t_0 \) is not the greatest element in \( T \). Let \( T' = \{ t \mid t \geq t_0 \} \). There are two cases: (1) if \( T' \) has a least element \( t' \), since \( T \) is inductive, \( t' \in T \) is a regular time point. (2) otherwise, for any \( t' \in T', \{ t \mid t_0 < t < t' \} \neq \emptyset \). Since \( T \) is also continuous, we can find a \( t' \in T' \) such that \( T'' = \{ t \mid t_0 < t < t' \} \). Therefore, \( t \) is a regular time point \( \forall t \in T'' \). Both cases contradict the fact that \( t_0 \) is the least upper bound of the set \( T \). 

We prove that the property \( v(t) \models \alpha_{r(t)} \) is satisfied at \( 0 \) and is both inductive and continuous on any time structure \( T \).

- **Initiality:** Since \( v(0) \models \Theta \) and \( v(0) \models e(r(0)) \), we have \( v(0) \models \Theta \land e(r(0)) \). According to the Initiality condition of invariants, we have \( v(0) \models \alpha_{r(0)} \).

- **Inductivity:** Suppose \( v(t) \models \alpha_{r(t)} \) is satisfied at \( 0 \leq t < t_0 \). Since \( r \) is a run over \( v \), \( \exists q \in Q \) and \( t'_1 < t_0 \), \( \forall t, t'_1 \leq t < t_0, r(t) = q \) and \( v(t_0) \models c(q, r(t_0)) \). According to the Consecution condition of the invariants, \( \exists t'_2 < t_0, \forall t, t'_2 \leq t < t_0, v(t) \models \alpha_q \) implies \( v(t_0) \models c(q, r(t_0)) \rightarrow \alpha_{r(t_0)} \). Therefore, \( \forall t, \max(t'_1, t'_2) \leq t < t_0, r(t) = q, v(t) \models \alpha_q \) (assumption), \( v(t_0) \models c(q, r(t_0)) \rightarrow \alpha_{r(t_0)} \) and \( v(t_0) \models c(q, r(t_0)) \). Thus, \( v(t_0) \models \alpha_{r(t_0)} \).

- **Continuity:** Suppose \( v(t_0) \models \alpha_{r(t_0)} \). Since \( r \) is a run over \( v \), \( \exists q \in Q \) and \( t'_1 > t_0 \), \( \forall t, t > t_0 \) and \( r(t) = q \) and \( v(t) \models c(r(t_0), q) \). According to the Consecution condition of the invariants, \( \exists t'_2 > t_0, \forall t, t < t'_2, v(t_0) \models \alpha_{r(t_0)} \) implies \( v(t) \models c(r(t_0), q) \rightarrow \alpha_q \). Therefore, \( \forall t, t_0 < t < \min(t'_1, t'_2) \), \( r(t) = q, v(t_0) \models \alpha_{r(t_0)} \) (assumption), \( v(t) \models c(r(t_0), q) \rightarrow \alpha_q \) and \( v(t) \models c(r(t_0), q) \). Thus, \( \forall t, t_0 < t < \min(t'_1, t'_2), v(t) \models \alpha_{r(t)} \).

**Proposition 7.7** Let \( \{ \alpha_q \}_{q \in Q} \) be invariants for \( B \) and \( A \) and \( r \) be a run of \( A \) over a trace \( v \in B \). If \( \{ \rho_q \}_{q \in Q} \) is a set of Lyapunov functions for \( B \) and \( A \), then

- \( E(\rho_{r(t_2)}(v(t_2))) \leq \rho_{r(t_1)}(v(t_1)) \) when \( \forall t_1 \leq t \leq t_2, r(t) \in B \cup S \),
• \( \frac{E(\rho_{(t_2)}(v(t_2)) - \rho_{(t_1)}(v(t_1))}{\mu(t_1, t_2)} \leq -\epsilon \) when \( t_1 < t_2 \) and \( \forall t_1 \leq t \leq t_2, r(t) \in B \), and

• if \( BS \) is the set of segments of consecutive \( B \) and \( S \)-states in \( r \), then \( \forall q^* \in BS, \mu_B(q^*) \) is finite.

Proof: For any run \( r \) over \( v \) and for any segments \( q^* \) of \( r \) containing only bad states and stables states, \( E(\rho) \) on \( q^* \) is non increasing, i.e., let \( I \) be the time interval of \( q^* \), for any \( t_1 < t_2 \in I \),

\[ \rho_{r(t_1)}(v(t_1)) \geq E(\rho_{r(t_2)}(v(t_2)), \) and the decreasing speed at the bad states is no less than \( \epsilon \). Let \( m \) be the upper bound on \( \{\rho_{r(t)}(v(t)) \mid t \in I\} \). Since \( \rho_q \geq 0, \mu_B(q^*) \leq m/\epsilon < \infty \).

Proposition 7.8 Let \( \{\alpha_q\}_{q\in Q} \) be invariants for \( B \) and \( A \) and \( r \) be a run of \( A \) over a trace \( v \in B \). If there exist local and global timing functions for \( B \) and \( ATA \), then

• if \( Sg(q) \) is the set of segments of consecutive \( q \)'s in \( r \), then \( \forall q \in T, q^* \in Sg(q), E(\mu(q^*)) \leq \tau(q), \) and

• if \( BS \) is the set of segments of consecutive \( B \) and \( S \)-states in \( r \), then \( \forall q^* \in BS, E(\mu_B(q^*)) \leq \tau(bad) \).

Proof: Similar to the proofs of Proposition 7.5 and Proposition 7.7.

Theorem 7.3 The verification rules (I), (S) and (AT) are sound if the following conditions on \( B \) and \( ATA \) are satisfied:

• \( T \) is an infinite time structure.

• All traces in \( B \) are specifiable by \( ATA \).

The verification rules are complete if the following conditions on \( B \) and \( ATA \) are satisfied:

• \( \{<v, r> \mid v \in B, r \text{ is a run over } v\} \) is time-invariant.

• All transitions from \( R \) to non-\( R \)-states are left-closed, i.e., if \( r \) is a run, and there is a transition from a \( R \)-state to a \( B \)-state or a \( S \)-state at \( t \), then \( r(t) \in B \cup S \).

Proof: Soundness is derived from Propositions 7.6, 7.7, 7.8. For any trace \( v \), there is a run since \( v \) is specifiable by \( TA \). For any run \( r \) over \( v \), if any automaton-state in \( R \) appears infinitely many times in \( r \), \( r \) is accepting. Otherwise there is a time point \( t_0 \), the sub-sequence \( r \) on \( I = \{t \in T \mid t \geq t_0\} \), denoted \( q^* \), has only bad and stable automaton states. If there exist a set of invariants and a set
of Lyapunov functions, $\mu_B(q^*)$ is finite. Since time is infinite, all the automaton states appearing infinitely many times in $r$ belong to $S$; $r$ is accepting too. If there exists a set of local and global timing functions, $r$ satisfies the time constraints; $r$ is accepting for $ATA$.

On the other hand, if $ATA$ is valid over $B$, there exist a set of invariants, a set of Lyapunov functions, and a set of local and global timing functions that satisfy the requirements.

The set of invariants can be constructed as follows: $\forall s \forall q, s \models \alpha_q$ iff the pair $(q, s)$ is reachable, i.e., $\exists r, v, t, r(t) = q \land v(t) = s$. We shall prove that $\{\alpha_q\}_{q \in Q}$ is a set of invariants.

- **Initiality:** if $\Theta(s) \land e(q)(s), \exists r, v, r(0) = q$ and $v(0) = s$. Therefore, $s \models \alpha_q$.

- **Inductivity:** $\forall v, t, \text{if } \exists t' < t, \forall t' < t'' < t, \exists r, r(t'') = q \left( v(t'') \models \alpha_q \right)$, then $\exists r, \exists t'_0 < t, \forall t'_0 < t'' < t, r(t'') = q$. If $v(t) \models c(q, q')$, then $r(t) = q'$, i.e., $v(t) \models \alpha_{q'}$. Therefore, $v(t) \models c(q, q') \rightarrow \alpha_{q'}$.

- **Continuity:** $\forall v, t, \text{if } \exists r, r(t) = q \left( v(t) \models \alpha_q \right)$, and $\exists t' > t \exists q', \forall t' < t'' < t, v(t'') \models c(q, q')$, $\forall t' < t'' < t, r(t'') = q'$. Therefore, $\exists t' > t, \forall t' < t'', r(t'') = q'$. Given the above constructed invariants, a set of Lyapunov functions can be constructed as follows:

  - $\forall q \in R$ and $s \models \alpha_q$, let $\rho_q(s) = 0$.

  - $\forall q \not\in R$ and $s \models \alpha_q$, the Lyapunov function is defined as follows. For any $r, v, t$ with $r(t) = q$ and $v(t) = s$, let $q^*$ be a segment of $r$ with only bad and stable states starting at $q$, and $\mu_B(q^*)$ be the measure of $B$-states in $q^*$. Let $\rho_q(s)$ be the average measure for all $r, v, t$ with $r(t) = q$ and $v(t) = s$, i.e., $\rho_q(s) = \mathbb{E}(\mu_B(q^*))$.

We shall prove that $\{\rho_q\}_{q \in Q}$ is a set of Lyapunov functions and global timing functions. For $q, q' \not\in R$, let $(q, s) \prec (q', s')$ iff $\exists r, v, t, t' < t'', t'' < t', r(t'') \not\in R, r(t) = q, v(t) = s$ and $r(t') = q'$ and $v(t') = s'$. Since $\{(v, r)\}$ is time-invariant, $\prec$ is transitive. Therefore, $(q, s) \prec (q', s')$ implies $\rho_q(s) \geq \mathbb{E}(\rho_{q'}(s'))$.

- **Definedness:** $\forall q \in Q, s \models \alpha_q, \rho_q$ is defined at $s$.

- **Non-increase:** $\forall v \in B, \forall q \in S, q' \in R$,

\[
\{\alpha_q \land \rho_q = w\} v^{-1}c(q, q') \rightarrow \mathbb{E}(\rho_q') \leq w
\]
is trivially satisfied. \( \forall q \in S, q' \in B \cup S, \)

\[
\{ \alpha_q \land \rho_q = w \} v^{-} \{ c(q, q') \rightarrow E(\rho_{q'}) \leq w \}
\]

is satisfied since \( (q, s) \prec (q', s') \).

\( \forall v \in B, \forall q \in B \cup S, q' \in S, \)

\[
\{ \alpha_q \land \rho_q = w \} v^{+} \{ c(q, q') \rightarrow E(\rho_{q'}) \leq w \}
\]

is satisfied since \( (q, s) \prec (q', s') \). \( \forall q \in R, q' \in S, c(q, q') \) is false since all transitions from \( R \) to non-\( R \)-states are left-closed.

- Decrease: \( \forall v \in B, \forall q \in B, q' \in Q, \)

\[
\{ \alpha_q \land \rho_q = w \land E(\lambda) = t \} v^{-} \{ c(q, q') \rightarrow \frac{E(\rho_{q'}) - w}{\mu([0, E(\lambda)])} \leq -1 \}
\]

\( \forall q \in R, q' \in B, \)

\[
\{ \alpha_q \land \rho_q = w \land E(\lambda) = t \} v^{+} \{ c(q, q') \rightarrow \frac{E(\rho_{q'}) - w}{\mu([0, E(\lambda)])} \leq -1 \}
\]

is trivially satisfied since \( c(q, q') \) is false. \( \forall q \in B \cup S, q' \in B, \)

\[
\{ \alpha_q \land \rho_q = w \land E(\lambda) = t \} v^{+} \{ c(q, q') \rightarrow \frac{E(\rho_{q'}) - w}{\mu([0, E(\lambda)])} \leq -1 \}
\]

The local timing functions can be defined similarly. ■

**Proposition 7.9** All transitions from \( R \) to non-\( R \)-states are left-closed, if the following conditions are satisfied:

- \( \mathcal{AT} \mathcal{A} \) is open and complete.

- \( \forall q \in R, q_1 \not\in R \) and \( q_2 \in R, c(q, q_1) \land c(q, q_2) \) is not satisfiable.

- All traces in \( B \) are right-continuous.

**Proof:** Since \( \mathcal{AT} \mathcal{A} \) is open, \( \forall q \in Q, q' \in R, c(q, q') \) is open. Therefore, \( \forall q \in Q, \bigvee_{q' \in R} c(q, q') \) is open. Since \( \forall q \in R, q_1 \not\in R \) and \( q_2 \in R, c(q, q_1) \land c(q, q_2) \) is not satisfiable, \( (\bigvee_{q' \in R} c(q, q')) \land (\bigvee_{q'' \in B \cup S} c(q, q'')) \) is not satisfiable. Since \( \mathcal{AT} \mathcal{A} \) is complete, \( \bigvee_{q' \in R} c(q, q') \) and \( \bigvee_{q'' \in B \cup S} c(q, q'') \) are complementary. Therefore, we obtain \( \bigvee_{q' \in R} c(q, q') \) is closed. Since all traces in \( B \) are right-continuous, for all \( v, t \), if \( t \) is a limit point to the right time points \( T \), \( v(t) \) is a point or a limit point of \( v(T) \). If \( \exists t' > t, \forall t < t'' < t', v(t'') \in \bigvee_{q' \in R} c(q, q''), v(t) \in \bigvee_{q' \in R} c(q, q') \). Therefore, all transitions from \( R \) to non-\( R \)-states are left-closed. ■

240
### A.5 Behavioural Verification with a Probabilistic Temporal Logic

**Proposition 8.1** Assume a well-defined continuous time $T$ and discrete domain $\times_I A_i$ PCN model (with positive transport delays to avoid algebraic loops) where all probabilistic transductions $P_i$ are event-based with exponential distribution $\text{exp}(\lambda_i)$. For any initial value $v_0 \in \times_I A_i$, the probability measure of the set of infinite traces $B : v_0, v_1, v_2, \ldots$ for which time is convergent, i.e., $\sum_{i \geq 0} t_i < \infty$ where $t_i$ denote the time of events $i \geq 0$, converges to 0.

**Proof:** Let us assume that $\mathcal{X} = \langle L, T, P, C \rangle$ is PCN model and that $\lambda_m = \max \{\lambda_i | \text{event } P_i \sim \text{exp}(\lambda_i)\}$. $\lambda_m$ represents the maximum rate at which the probabilistic events (transitions) can occur. For the purpose of this proof, let us extend the notion of trace into a timed trace. A timed trace simply corresponds to a trace augmented with the information of the length of time that the system spends at a given state value. That is, a timed trace is a sequence of pairs $(v_i, t_i)$ where $v_i$ is the value of the system’s state ($v_i \in \times_I A_i$) and $t_i$ is the time spent at $v_i$ after the probabilistic event that lead to $v_i$ occurred. The $t_i$ values can easily be obtained by subtracting the time instances at which the events occur.

First, let us denote the set of all non-time-divergent timed traces that start at value $v_0$ by $NTD(v_0)$ and the set of all timed traces starting in $v_0$ and which stay at most for 1 time unit at each value of the trace by $D(v_0)$. Formally, $D(v_0)$ consists of all timed traces $v : (v_0, t_0), (v_1, t_1), \ldots$ such that the sequence $v_0, v_1, \ldots$ is a well-defined trace and $t_i \leq 1, \forall i$. In order to prove the result of the proposition, we first show that the probability measure of $D(v_0)$ is 0. Let us consider the cylinder set $CS_n(v_0) = C(v_0, [0,1], v_1, [0,1], \ldots, [0,1], v_n)$. Clearly, $CS_n(v_0)$ is the superset of the set that contains exactly the timed traces $v : (v_i, t_i)$, where $t_i \leq 1$ and $i < n$. Moreover, one can easily see that $D(v_0) = \bigcap_{n \geq 0} CS_n(v_0)$. By assumption, each time delay between change in state is distributed with respect to an exponential distribution. Hence we get

$$\mu_{v_0}^{\mathcal{X}}(CS_n(v_0)) = \prod_{0 \leq i \leq n} P^{\mathcal{X}}(v_i, v_{i+1}) \cdot (1 - e^{-\lambda_i}) \leq (1 - e^{-\lambda_m})^n.$$  

Furthermore, since $0 < (1 - e^{-\lambda_m}) < 1$, we get that

$$\mu_{v_0}^{\mathcal{X}}(D(v_0)) = \lim_{n \to \infty} \mu_{v_0}^{\mathcal{X}}(CS_n(v_0)) = 0.$$
Now let us conclude the proof by showing that the probability measure of the set of non-time-divergent traces is 0. Obviously, for any non-time-divergent trace, the sum of time delays between transitions converges, i.e., \( \sum_{i=0}^{\infty} t_i < \infty \). A necessary condition for such a convergence is that the sequence \( (t_i)_{i \geq 1} \) converges to 0. Thus, there exists a natural number \( \eta \geq 1 \) for which \( t_i < 1/\eta \), \( \forall i > \eta \). Hence this implies that the suffix timed trace \( (v_\eta, t_\eta), (v_{\eta+1}, t_{\eta+1}), \ldots \) belongs to the set \( D(v_\eta) \). We then obtain that the set \( NTD(v_0) \) is a subset of

\[
\bigcup_{n \geq 1} \{ v \in B^\infty_{v_0} \mid (\exists v' \in D(v_\eta)) \land (\exists t_0, t_1, \ldots, t_{n-1} \in \mathbb{R}), v : (w_0, t_0), \ldots, (w_{n-1}, t_{n-1}), v' \}.
\]

With this, we obtain that

\[
\mu(NTD(v_0)) \leq \sum_{n=0}^{\infty} \sum_{v_0, \ldots, v_n \in \times_i A_i} \mu(CS_n(v_0)) = 0.
\]

Note: this proposition can be extended to the case where the distribution of the delays between probabilistic transduction is not necessarily exponential. For all non-time-divergent traces, as long as the the probability of the time delay for each probabilistic transduction being between \([0, 1]\) is less than one, then the result will hold.

**Proposition 8.2** Assume a PCN trace \( v : T \to A \) which satisfies a PATTL formula \( \varphi \). Let \( T_r \) be any reference time of \( T \) with a reference time mapping \( h : T \to T_r \). Then the extension trace \( \overline{v} : T_r \to A \) of \( v \) onto \( T_r \) also satisfies \( \varphi \).

**Proof:** Without loss of generality, assume that \( \varphi = \varphi_1 \cup^r \varphi_2 \). A similar proof can be constructed for \( \varphi = \varphi_1 \setminus^r \varphi_2 \).

Since \( v \) satisfies \( \varphi_1 \cup^r \varphi_2 \), we have that \( \exists t' \in T_{t+\tau}, v \models \varphi_2 \) and \( \forall t'' \in T_{t+\tau}, t < t'' < t', v \models \varphi_1 \). From the definition of extension trace and the fact that a reference time of \( T \) is as dense as \( T \), we also have that \( \exists t' \in T_{t+\tau}, \overline{v} \models \varphi_2 \).

Let \( t_1 \) and \( t_2 \) be two consecutive time points in the time structure \( T \). Then for every such \( t_1, t_2 \in T \), we have \( \overline{v}(t) = v(t_1), \forall t \in T_r, t_1 \leq t < t_2 \). Since \( \forall t'', t < t'' < t', v \models \varphi_1 \), this implies that \( \forall t'' T_{t+\tau}, t < t'' < t', v \models \varphi_1 \). Hence, for any arbitrary reference time structure \( T_r \), we have that \( \overline{v} \models \varphi_1 \cup^r \varphi_2 \).

**Proposition 8.3** For any PCN with multiple clocks, the number of different induced probability distribution is finite and at most \( 2^n - 1 \), where \( n \) is the size of the location set \( L \).
Proof: Assume a PCN with \( n \) different locations. From the definition of the reference time structure \( T_r \) (the coarsest time structure), at each transition from \( t_i \) to \( t_{i+1} \), there is at least one location that is not extended. On the other hand, there is at most \( n \) locations that can transition for each time point. This coincides to a time point which is a multiple of every single clock of the system. Hence, we can easily conclude that at each time point, we can have between 1 and \( n \) locations that are not extended, which in turn gives us that at most \( 2^n - 1 \) different transition distributions must exist, where \( n \) is finite. 

**Proposition 8.4** For \( \tau \geq 0 \), \( \mathcal{B}(\tau, v_t, t) \) is obtained via the following recurrence relation:

\[
\mathcal{B}(\tau, v_t, t) = \begin{cases} 
1 & \text{if } \varphi_2 \in \text{valid}(v_t) \\
0 & \text{elseif } \varphi_1 \notin \text{valid}(v_t) \\
\sum_{v' \in A} \mathcal{P}^{\mathcal{X}}_r(v_t, v', t + \Delta_r) \times \mathcal{B}(\tau - \Delta_r, v', t + \Delta_r) & \text{otherwise}
\end{cases}
\]

Note that since \( \Delta_r \) is positive, we are guaranteed that the recurrence relation will be well-defined and always terminate (after \( k > 0 \) steps when \( \tau - k\Delta_r \) becomes negative).

**Proof:** This proof is based on the proof of Proposition 1 from [HJ94]

Assume a PCN \( \mathcal{X} \). For states of \( \mathcal{X} \) denoted by \( v \in A \) and a nonnegative real time threshold \( \tau \), let \( \Psi(\tau, v_t, t) \) be the set of finite traces, starting at time \( t \), of the form \( v_t, v_{t+\Delta_r}, \ldots, v_{t+k\Delta_r} \) such that \( k\Delta_r \leq \tau, v_{t+k\Delta_r} \models \varphi_2 \), and for all \( 0 \leq i < k \) we have \( v_{t+i\Delta_r} \models \varphi_1 \) and \( v_{t+i\Delta_r} \not\models \varphi_2 \). As defined above, \( \mathcal{B}(\tau, v_t, t) \) represents the measure for the set of traces \( w \in B_{v_t}^\mathcal{X} \) for which the PATTL until formula \( \varphi_1 \U \varphi_2 \) is satisfied. From the definition of \( \mathcal{B}(\tau, v_t, t) \) we get:

\[
\mathcal{B}(\tau, v_t, t) = \sum_{v_t, v_{t+\Delta_r}, \ldots, v_{t+k\Delta_r} \in \Psi(\tau, v_t, t)} \mathcal{P}^{\mathcal{X}}_r(v_t, v_{t+\Delta_r}, t + \Delta_r) \times \cdots \times \mathcal{P}^{\mathcal{X}}_r(v_{t+(k-1)\Delta_r}, v_{t+k\Delta_r}, t + k\Delta_r) \quad (A.5)
\]

Based on Equation A.5, we consider three distinct cases, associated with the three conditions of the recurrent loop:

1. \( v_t \models \varphi_2 \): For any trace \( w \in B_{v_t}^\mathcal{X} \) with initial state \( v_t \) satisfying \( \varphi_2 \), we have, from the semantics of the until formula, that \( w \models \varphi_1 \U \varphi_2 \). Therefore, we get \( \mathcal{B}(\tau, v_t, t) = 1 \).

2. \( v_t \not\models \varphi_1 \land v_t \not\models \varphi_2 \): By definition, for any trace \( w \in B_{v_t}^\mathcal{X} \), where the initial state \( v_t \) does not satisfy either \( \varphi_1 \) or \( \varphi_2 \), we have \( w \not\models \varphi_1 \U \varphi_2 \) which yields \( \mathcal{B}(\tau, v_t, t) = 0 \).
3. \( v_t \models \varphi_1 \land v_t \not\models \varphi_2 \): For this situation, we need to consider two sub cases.

(a) \( \tau = 0 \): From the previous section, we know that a state \( v_t \) satisfies \( \varphi_1 \cup \varphi_2 \) iff \( v \models \varphi_2 \).

Hence for \( \tau = 0 \), we have \( B(\tau, v_t, t) = 0 \).

(b) \( \tau > 0 \): From the definition of the recurrence relation, we have:

\[
B(\tau, vt, t) = \sum_{\psi(\tau, vt, t)} p_\tau^X(v_t, v_{t+\Delta_{r'}}, t + \Delta_r) \cdots p_\tau^X(v_t+(k-1)\Delta_{r'}, v_{t+k\Delta_{r'}}, t + k\Delta_r)
\]

\[
= \sum_{v_t+\Delta_r} p_\tau^X(v_t, v_{t+\Delta_r}, t + \Delta_r) \times \sum_{v_{t+\Delta_r}, \ldots, vt+k\Delta_r} p_\tau^X(v_{t+\Delta_r}, v_{t+2\Delta_r}, t + 2\Delta_r) \cdots p_\tau^X(v_{t+(k-1)\Delta_r}, v_{t+k\Delta_r}, t + k\Delta_r)
\]

\[
= \sum_{v_t+\Delta_r} p_\tau^X(v_t, v_{t+\Delta_r}, t + \Delta_r) B(\tau - \Delta_r, v_{t+\Delta_r}, t + \Delta_r)
\]

Proposition 8.5 Let \( Cl = \{c_1, c_2, \ldots, c_k\} \) be the set of all clocks of a PCN \( X \). If for every clock in \( Cl \) the time increments are equally spaced, then there exist a finite cycle in the order of the transition distributions \( p_i^X \).

Proof: Let \( \Delta_i > 0, i = 1, \ldots, k \) be the time increments of each of the \( k \) clocks of the system on the reference time structure \( T_r \). Since \( T_r \) is the coarsest time structure for the system, it is easy to show that each \( \Delta_i \) is an integer. Further assume that all clocks are initiated at the same time point \( t_0 \). Let us show that there exist a point in the future, call it \( t_c \), hence \( t_c > t_0 \), for which all the clocks will be clicking all at once. The value \( t_c - t_0 \) represents the period of the cycle, period during which the finite set of induced transition probability distributions will be used.

The least common multiple of two numbers \( \Delta_1 \) and \( \Delta_2 \), denoted by \( \text{LCM}(\Delta_1, \Delta_2) \), is the smallest number \( T \) for which there exist positive integers \( n_1 \) and \( n_2 \) such that

\[
n_1\Delta_1 = n_2\Delta_2 = T.
\]

It is easy to show that the least common multiple of \( \Delta_i > 0, i = 1, \ldots, k \) is the value of the period of the cycle.

The least common multiple of two numbers \( \Delta_1 \) and \( \Delta_2 \) can be obtained by finding the prime factorization of each

244
\[ \Delta_1 = p_1^{e_1} \cdots p_n^{e_n} \]
\[ \Delta_2 = p_1^{e_1} \cdots p_n^{e_2n} \]

where the \( p_i \)'s are all prime factors of \( \Delta_1 \) and \( \Delta_2 \), and if \( p_i \) does not occur in one factorization, then the corresponding exponent is taken as 0. The least common multiple is then given by

\[ \text{LCM}(\Delta_1, \Delta_2) = \prod_{i=1}^{n} p_i^{\max(e_{i1}, e_{i2})} \]

The least common multiple \( \text{LCM}(\Delta_1, \Delta_2, \Delta_3, \ldots, \Delta_k) \) of more than two numbers is similarly defined. Hence, from the uniqueness property of the prime factorization and from the finiteness of the set of clocks, we have shown that a cycle of finite period always exists. \( \blacksquare \)

**Corollary 8.1** \( \forall p \in (0, 1), p \in \mathbb{R}, \) and any \( k \in \mathbb{N} \), there exist a subset \( I \subseteq \{1, \ldots, k\} \) such that the following inequality holds:

\[ p - \frac{1}{2k} \leq \sum_{i \in I} \frac{1}{2^i} \leq p \]

(A.6)

**Proof:**

We will proceed to show this result by induction on \( k \).

**Base Case:** \( k = 1 \)

At \( k = 1 \), there are only 2 possible choices for \( I \): either \( I = \{1\} \) or \( I = \emptyset \).

- \( p \geq 0.5 \): Let \( I = \{1\} \). Then we have \( p - 1/2 \leq 1/2 \leq p \). \( \checkmark \)
- \( p < 0.5 \): Let \( I = \emptyset \). Then we have \( p - 1/2 \leq 0 \leq p \). \( \checkmark \)

**Inductive Step:** Assume Equation 8.5 is true for some \( k > 1 \). We need to show that Equation 8.5 holds for \( k + 1 \).

From our induction hypothesis, for \( I_k \subseteq \{1, \ldots, k\} \) we have

\[ p - \frac{1}{2^k} \leq \sum_{i \in I_k} \frac{1}{2^i} \leq p \]
For $k + q$, there are 2 possible choices for $I_{k+1}$. Either $I_{k+1} = I_k$ or $I_{k+1} = I_k \cup \{k+1\}$. We want to show:

$$p - \frac{1}{2^{k+1}} \leq \sum_{i \in I_{k+1}} \frac{1}{2^i} \leq p.$$

Consider now the two cases:

1. if $\sum_{i \in I_k \cup \{k+1\}} \frac{1}{2^i} > p$ then choose $I_{k+1} = I_k$.

   From the induction hypothesis, we already know that $\sum_{i \in I_k} \frac{1}{2^i} \leq p$. Hence we only need to show the left inequality, that is, $p - \frac{1}{2^{k+1}} \leq \sum_{i \in I_k} \frac{1}{2^i}$. This is trivial since $\sum_{i \in I_k \cup \{k+1\}} \frac{1}{2^i} > p$, which is equivalent to $p - \frac{1}{2^{k+1}} < \sum_{i \in I_k} \frac{1}{2^i}$. √

2. if $\sum_{i \in I_k \cup \{k+1\}} \frac{1}{2^i} \leq p$ then choose $I_{k+1} = I_k \cup \{k+1\}$.

   Once again, we need only show the leftmost inequality: $p - \frac{1}{2^{k+1}} \leq \sum_{i \in I_k \cup \{k+1\}} \frac{1}{2^i}$. We know, from the induction hypothesis, that $p - \frac{1}{2^k} \leq \sum_{i \in I_k} \frac{1}{2^i}$. Hence, all we need to show is that $1/2^k - 1/2^{k+1} \leq 1/2^{k+1}$, which is easily shown. √

We have shown that $\forall p \in (0, 1), p \in \mathbb{R}$, and any $k \in \mathbb{N}$, there exist a subset $I \subseteq \{1, \ldots, k\}$ such that the set of inequalities 8.5 holds.

Theorem 8.1 There does not exist a finite iterative procedure to solve the set of linear Equations 8.4 for any order on the induced probability distributions.

Proof: Let us consider the two transition probability distributions $P_1$ and $P_2$ as displayed in Figure A.1. We can show that there exists an order of selecting the two distributions in such a way that we can generate any stationary distribution for which the system will end up in state $A$ and $B$, with probability $p$ and $q = 1 - p$ respectively. This is proved by a direct application of Corollary 8.1 where the set $I$ is the infinite set which indicates at which time point the probability distribution $P_1$ should be used. We have that for any number $p \in (0, 1)$, the corollary holds. In particular, let $p = \pi/4$. Since $\pi/4$ is a transcendental number, we know that it is not the solution of any polynomial equation of the form $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0 = 0$ where $n \geq 1$ and the coefficients $a_i$ are rationals, not all 0. Therefore, the distributions of Figure A.1, there exists no polynomial equations which can solve for the stationary distribution showed on the right.
Hence, there does not exist a finite algorithm that can compute the solution of the system of linear equations 8.4 for any order of applying the induced probability distributions.

**Proposition 8.6** For any \( \tau > 0 \), we have

\[
\mathcal{B}(\tau, t) = \mathcal{M}_t \times \mathcal{M}_{t+\Delta_r} \times \cdots \times \mathcal{M}_{t+r\Delta_r} \times \mathcal{B}(0, t) = \widehat{\mathcal{M}} \times \mathcal{B}(0, t)
\]  

\[(A.7)\]

**Proof:** See result of Proposition 8.4 [HJ94] □

**Proposition 8.7**

Let \( M \) be the number of matrix multiplication such that we have \( \mathcal{B}(\infty, t) = \mathcal{M}_t \times \mathcal{M}_{t+\Delta_r} \times \cdots \times \mathcal{M}_{t+M\Delta_r} \times \mathcal{B}(0, t) = \widehat{\mathcal{M}} \times \mathcal{B}(0, t) \). Furthermore, let \( \delta_i^M = \sum_{j \in \mathcal{A}_e^\mathcal{K}} \widehat{\mathcal{M}}[i, j] \), with \( i \in \{k \mid v_k \in \mathcal{A}_e^\mathcal{K} \} \).

Then, for every equivocal state \( v_i \), the true measure \( \mathcal{B}(\infty, v_i, t) \) of the paths satisfying \( \varphi \) lies between the following bounds:

\[
\mathcal{B}(\infty, t)_i \leq \mathcal{B}(\infty, v_i, t) \leq \mathcal{B}(\infty, t)_i + \delta_i^M
\]  

\[(A.8)\]

**Proof:** Assume we proceeded to \( M \) matrix multiplication to obtain \( \widehat{\mathcal{M}} \). Since we are multiplying probability distributions, it is easy to show that each row of \( \widehat{\mathcal{M}} \) will sum to 1. After \( M \) multiplication, for every \( v_i \in \mathcal{A}_e^\mathcal{K} \), the values in \( \widehat{\mathcal{M}} \) associated to the columns corresponding to states in

![Diagram](image-url)
\( A_v^\mathcal{X} \) and \( A_f^\mathcal{X} \) represents the current estimate of the measure of paths which respectively satisfy \( \varphi \) and \( \neg \varphi \). Let us call these measure estimates \( \mu_v M v(i) \) and \( \mu_f M (i) \). From the definition of \( \mathcal{M} \), after the \( M + 1 \)th multiplication, it is easy to show that we are guaranteed to get \( \mu_v^{M+1} v(i) \geq \mu_v^M v(i) \) and \( \mu_f^{M+1} (i) \geq \mu_f^M v(i) \), hence proving the lower bound of Equation 8.8.

To prove the upper bound value, we just need to notice that from our construction of the matrices \( \mathcal{M} \), based on the partitions of the state space, no state in \( A_v^\mathcal{X} \) is absorbing. Hence, we get that \( \lim_{M \to \infty} \delta_t^M = 0 \). \( \delta_t \) represents the measure of paths which so far only visited equivocal states. In the long run, we are guaranteed that these paths will either end up in \( A_v^\mathcal{X} \) or \( A_f^\mathcal{X} \) or remain there forever. Hence, the value of the total measure attributed to these paths will eventually be distributed to the absorbing states. Hence, the most weight a single absorbing state \( v_i \) can get from this point on is exactly the value of \( \delta_t^M \), which confirms the value of the upper bound and conclude this proof. 

**Proposition 8.8**

Let us consider two separates case for the value of the threshold \( \tau \).

- For \( \tau < \Delta r \) or \( t = 0 \), \( \mathcal{D}(\tau, v_i) \) is obtained via the following equation:

\[
\mathcal{D}(\tau, v_i) = n \varphi^2
\]

(A.9)

- For \( \tau \geq \Delta r \), the paths measure is obtained with

\[
\mathcal{D}(\tau, v_i) = \left[ \mathcal{P}_0^\mathcal{X} \times (\mathcal{P}_r^\mathcal{X}) M_1 \times \left[ \sum_{k=1}^{M_2} (\mathcal{P}_r^\mathcal{X})^{k-1} M_{\varphi_2-\varphi_1} M_{\varphi_1-\varphi_1} \right] \right] ' + n \varphi^2 (i) \quad (A.10)
\]

and where \( \mathcal{P}_0^\mathcal{X} \) is the vector of the system’s initial state’s distribution, \( M_1 = \left\lfloor \frac{\max(0, t - \tau)}{\Delta r} \right\rfloor \), \( M_2 = \left\lfloor \frac{\min(t, r)}{\Delta r} \right\rfloor \), and the matrices \( M_{\varphi_k-\varphi_l} \) are defined as follows:

\[
M_{\varphi_k-\varphi_l}(v_i, v_j) = \begin{cases} 
\mathcal{P}_r^\mathcal{X}(v_i, v_j) & \text{if } v_i \models \varphi_k \text{ and } v_j \models \varphi_l \\
0 & \text{otherwise}
\end{cases} \quad (A.11)
\]

**Proof:**

Let us consider the two cases specified in the Proposition 8.8.
• For \( \tau < \Delta_r \) and for \( t = 0 \), it is easy to see, from the semantics of the since formula in Equation 8.2, that the only time value \( t' \) included in the set \( T_{t-\tau} \) is \( t \) itself. Therefore, at state \( v \), the satisfaction of \( \varphi_1 S^r \varphi_2 \) is equivalent to \( v \models \varphi_2 \), which is obtained from Equation 8.11.

• For \( \tau \geq \Delta_r \), we are looking for the measure of all paths for which \( \exists t' \in T_{t-\tau}, v \models \varphi_2 \) and \( \forall t'', t' < t'' \leq t, v \models \varphi_1 \). Hence, we are concerned with the sub paths ending at time \( t \) which once they reach a \( \varphi_2 \) state within the interval \( [t - \tau, t] \), either do no leave \( \varphi_2 \) states or move to \( \varphi_1 \) states continuously until reaching time \( t \). The inner parenthesis in Equation 8.12 computes the measure of all sub paths between \( [t - \tau, t] \) which follows this property. The measure is then multiplied with the probability of reaching the intermediate start state at time \( t - \tau \). To this measure, we finally add the value of \( \varphi_1 S^r \varphi_2 \) with probability one. ■
Appendix B

Modeling and Verifying an Elevator System with Uncertainty

To demonstrate the scope of our approach, we will analyze an elevator system exhibiting uncertainty. Elevator systems constitute a useful testbed for hybrid systems, as confirmed by the fact that they have been used as a benchmark for various approaches to real-time systems [ZM99, Bar85, DC95, San89]. Nevertheless, most previous approaches focus on discrete deterministic dynamics. Combining continuous Newtonian dynamics and discrete control from users' requests, we extend the benchmark example in [ZM99] to account for the different types of uncertainty that can arise in a real physical elevator system. We model the different levels of the system (continuous motion, discrete controller) and provide a methodology for verifying the behaviour of the system with respect to a behavioural constraint of real-time response. We refer the reader to [ZM99] for a complete description and analysis of the deterministic hybrid version of this example. For another application of our framework, the reader should consult [SAM04] where we modeled and analyzed a robotic museum surveillance system encompassing uncertainty, on which were imposed constraints on the quality of service.

In Figure B.1, we show the PCN model of the elevator body as represented by a second order stochastic differential equation, based on Newton’s second law:

\[ \ddot{h} = k(t)\dot{h} + F + w(t) \]  

(B.1)

where \( F \) is the motor force (control input), \( k(t) \) is the coefficient of friction and \( h \) is the height of
the elevator. Moreover, the model is augmented two types of uncertainty: (1) uncertainty in the dynamics through variation of the friction coefficient \( k(t) \), and (2) a time-varying external disturbance force \( w(t) \) acting on the elevator. We assume that \( k(t) \) has a nominal value of \( k_0 = 1.05 \) and that it can vary with \( k(t) \in [0.70, 1.40] \), modeled as a Gaussian White Noise process with zero mean and standard deviation \( \sigma = 0.15 \). With these parameters, we have \( Pr(|k(t) - k_0| < 0.35) > 0.9804 \); that is, the value of the friction coefficient may exceed the presumed bounds on the uncertainty, but with a small non-zero probability. This phenomenon indicates a "soft" norm constraint on the uncertainty.

We will later augment the system with probabilistic passenger arrivals, and will then show that the system satisfies the constraint of serving passengers within fixed bounded time on average.

### B.1 Augmented Model

We augment the elevator system introduced in Section B with probabilistic passenger arrivals, which are modeled as a PCN transduction of a Poisson process. Passengers can arrive at any floor to request the use of the elevator. These requests will be granted when they conform to the elevator serving state. Obviously, passenger arrivals have an effect on the current passengers by increasing the time needed for the elevator to service their request.

As an example, we verify the non-trivial behavioural constraint that a passenger request will be serviced on average within \( \tau = 40 \) time units, regardless of the incoming requests that can occur during that passenger’s travel. Furthermore, we would like to obtain a probability bound on the time that a request could take to be satisfied. Before we proceed to behavioural constraint verification,
let us describe the elevator model in more detail.

**B.1.1 Continuous model**

Let us assume that floors are separated by $H$ units. Using the continuous model of the dynamics presented earlier we calculate the current floor number with:

$$f = \lfloor h/H \rfloor + 1$$  \hspace{1cm} (B.2)

where $\lfloor x \rfloor$ denotes the integer value closest to $x$. Using this relationship, we can get the distance to the nearest floor from: $d_s = h - (f - 1)H$. We also say that the elevator is at "home" position, for some $\epsilon > 0$, if:

$$e_h : |d_s| \leq \epsilon.$$  \hspace{1cm} (B.3)

In Figure B.2 we present the PCN module of the continuous component of the system. In this diagram, $Com$ is a high level command that can take values 1, -1 and 0, respectively denoting **up**, **down** and **stop**. $CONTROL_0$ is an analog controller which determines the force that drives the elevator body $BODY$ (see Figure B.1). Since the dynamics of the elevator are uncertain, $CONTROL_0$ needs to be optimal in some stochastic sense. Finally, the components $FLOOR$ and $HOME$ are represented by Equations B.2 and B.3 respectively.
Figure B.3: The hybrid model: combining continuous and discrete components of the elevator system.

**B.1.2 Discrete model**

An important discrete component of the system is the set of push buttons used by the users to issue requests. Each push button takes value 1 if pushed, and 0 otherwise. In our model, we will consider three different types of buttons: $U_b$, $D_b$ and $F_b$, which respectively denote up, down, and floor buttons. For an elevator consisting of $n$ floors, we have $U_b, D_b, F_b \in \{0, 1\}^n$ with $U_b(n) = D_b(1) = 0$. The state of a push button $b_s$ is determined by the user's input $b_i$ and the reset signal $b_r$ issued when a request has been served. A floor button will be on until the elevator stops at that floor while a direction button will be on until the elevator stops at the floor and is heading in the corresponding direction. The next state of a push button can be represented as: $b_s' = b_i \lor (\neg b_r \land b_s)$.

**B.1.3 Hybrid model**

Equipped with a model for the continuous dynamics of the elevator and a model for the user's input, we now need to combine the two to form an hybrid model of the elevator system. A discrete event-driven controller $CONTROL_1$ takes as input the current floor $f$ and button states $b_s$, and outputs the command $Com$ and a serving state as displayed in Figure B.3.

The events driving the controller are the results of the union of three event spaces: (1) there is a
user request when the elevator is idle at floor 1; (2) the elevator reached a home position \(|d_s| \leq \epsilon\); and (3) a request has been served for a given amount of time. Therefore, when any of these events occur, \(\text{CONTROL}_1\) proceeds to an update using the current values of its inputs.

### B.1.4 Control Design

In the previous sections, we referred to \(\text{CONTROL}_0\), an analog controller generating the force to drive the elevator’s body, and to \(\text{CONTROL}_1\), a discrete controller generating the high level command sent to the elevator. However, we did not define the controllers completely, instead using them as black boxes assumed to perform optimally in some sense. In general, the task of designing optimal controllers is complex and no automatic method exists. Furthermore, remember that we are dealing with uncertainty in the dynamics, which renders the design task harder.

**\(H^\infty\) control design**

Assume that we are interested in finding a simple linear proportional and derivative controller of the form:

\[
F = \begin{cases} 
F_0 & \text{if } \text{Com} = 1 \\
-F_0 & \text{if } \text{Com} = -1 \\
-K_p d_s - K_v d_s & \text{if } \text{Com} = 0
\end{cases}
\]  

where \(F_0 > 0\) is a constant force, \(K_p\) is a proportional gain and \(K_v\) is the derivative gain.

For a controller to be acceptable for our system, it needs to possess continuous (and exponential) stability. Moreover, we require hybrid consistency. That is, the analog controller must interface with the discrete control in a consistent fashion: if a stop command is issued by \(\text{CONTROL}_1\), then the elevator should continuously maintain \(|d_s| \leq \epsilon\), for all time values.

Let us now show that we can design a stabilizing controller. To design the controller for the elevator’s body, we chose to apply a robust control design using an \(H^\infty\) method [Zam81, PUS00]. We can rewrite Equation B.1 in a mathematically sound fashion by replacing the Gaussian white noise term with Brownian motion and by using the Itô stochastic differential equation:

\[
\begin{align*}
\,d x &= (Ax + B_1 u + B_2 w)dt + H x dW(t) \\
\,d z &= C x + D u
\end{align*}
\]  

(B.5)
where $x = [h \; \dot{h}]' \in \mathbb{R}^2$, $z$ is called the uncertainty output [PUS00] and $W(t)$ is a scalar Brownian motion process with identity covariance. Furthermore from our elevator model we get

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -1.15 \end{bmatrix}; \quad B_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}; \quad B_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix};$$

$$C = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}; \quad D = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}; \quad F = \begin{bmatrix} 0 \\ -1 \end{bmatrix}; \quad S = \begin{bmatrix} 0 & 1 \end{bmatrix}. $$

We also define $H = \sigma FS$, where $\sigma = 0.15$ as specified in Section 3.

We now apply Theorem 8.2.2 from [PUS00] to obtain a stabilizing controller which solves the $H^\infty$ problem associated with our stochastic dynamical system. The process of designing a controller based on Theorem 8.2.2 involves solving a set of Riccati equations which can be solved by homotopy method [RHP93a] or by a version of Newton’s method introduced in [Ugr98, DH01, Won68, Guo01]. We used the latter for this particular example. We obtained a stabilizing controller of the form (B.4) with $F_0 = 0.31$, $K_p = 1.1547$ and $K_v = 1.0691$.

To demonstrate that this controller guarantees hybrid consistency, we needed to show that, given the values of $F_0$, $K_p$ and $K_v$ obtained above, we have $\max_t |d_s(t)| \leq \epsilon$ for every possible value of $k(t) \in [0.70, 1.40]$. From the relationship between the variables $h$ and $d_s$, we can see that $\dot{h} = d_s$. Therefore, for $Com=0$, and by combining Equation B.1 with Equation B.4, we have

$$\ddot{d}_s + (k(t) + K_v)d_s + K_p d_s = 0 \quad (B.6)$$

It is easy to deduce that the maximum distance to a floor $D$, once $Com=0$, is attained when $\dot{d}_s = 0$. At this point, it is important to notice that Equation B.6 can be critically damped, underdamped or overdamped given that $k(t)$ takes values $1.08$, $[0.70, 1.08)$ and $(1.08, 1.40]$ respectively. Therefore, we need to analyze the solution of Equation B.6 for those three cases separately. Nevertheless, we showed that for each of these three cases, we obtain hybrid consistency.

**Discrete control design**

At the discrete level, we adopt a control strategy that forces the elevator to move persistently in one direction until there are no more requests in that direction. This ensures that we avoid the presence
of dead locks or live locks in the system. We define \textit{CONTROL}_0 as a controller which accepts the current request from the push buttons \( b_s \) along with the current floor \( f \) and state values \( s \) and determines: (1) the next command to issue to the elevator module; (2) the updated value of the serving state \( s \). For our system, we assume three different serving states: \textbf{up}, \textbf{down} and \textbf{idle}. The elevator is only idle at the first floor. We consider also three distinct types of binary requests that can be sent to the elevator: \textbf{UpRequest}, \textbf{DownRequest} and \textbf{StopRequest}, which we define as follows:

\[
Ur = \text{UpRequest} = Ub(f) \bigvee_{n \geq k > f} (Ub(k) \lor Db(k) \lor Fb(k))
\]

\[
Dr = \text{DownRequest} = Db(f) \bigvee_{1 \leq k < f} (Ub(k) \lor Db(k) \lor Fb(k))
\]

\[
Sr = \text{StopRequest} = \begin{cases} 
(Db(f) \lor Fb(f)) & \text{if } s = \text{down} \\
(Ub(f) \lor Fb(f)) & \text{otherwise}
\end{cases}
\]

Given these components, we can define the logical expressions for the transition functions for the serving state and the command to the elevator:

\[
s' = \text{ServingState}(f, s, Ur, Dr) = \begin{cases} 
\text{up} & \text{if } Ur \land (s \neq \text{down} \lor \neg Dr) \\
\text{down} & \text{if } (-Ur \land (f > 1)) \lor (Dr \land s = \text{down}) \\
\text{idle} & \text{otherwise}
\end{cases}
\]

\[
Com = \text{Command}(Sr, s) = \begin{cases} 
0 & \text{if } Sr \lor (s = \text{idle}) \\
1 & \text{if } \neg Sr \land (s = \text{up}) \\
-1 & \text{otherwise}
\end{cases}
\]

We show in Figure B.4 the PCN model of the discrete controller of the elevator, obtained by combining the logical expressions above.

\textbf{B.1.5 Example of Behavioural Constraint Verification}

Given a simple 3-floor elevator modeled as in Figure B.2(b), we can verify whether or not a request to go up from floor 1 to floor 3 will be served within 40 units of the elevator's motion time. This constraint can be represented as in Figure 7.1(d).

256
Even though the dynamics of the elevator are continuous, the specification of the behavioural constraints are such that combined with our system’s model, we can associate it to the stochastic transition system of Figure B.5. The corresponding state space $S$ is of the form $(f, s, Com, N1, N2, N3)$ where $f$, $s$ and $Com$ are the current floor, serving state and command of the elevator. $N1$, $N2$ and $N3$ denote the number of passengers currently in the elevator wanting to go to floors 1, 2 and 3, respectively. The initial state $\Theta$ is set to $(1, up, 0; 0, 0, 1)$. Furthermore, to keep it simple, we assume: (1) only one arrival can occur at any floor; (2) the average time required for someone to get in or out of the elevator is 5 units; and (3) the time to close the elevator doors is 3 units. By simple analysis of the passenger arrival probabilities, we obtain that the probability for the occurrence of a new request for the elevator is $p = 0.15$. Indeed, since we assume that the arrivals follow a Poisson process, and given the fact that the events triggering the discrete controller only happens once the elevator has reached a floor (there is no events possible while the elevator is traveling between floor), we can assume that the arrivals are concentrated at the time when the elevator reaches a new floor. Therefore, we can summarize the probability of arrivals with a single probability, $p$, obtained from solving $p = Pr(X(t_{max}) = 1)$, where $X(t)$ is the Poisson process modeling the arrivals and $t_{max}$ is the maximum traversal time of the elevator from one floor to another. To calculate $t_{max}$,
we perform a worst case analysis, for all possible values of \( k(t) \), on \( h = (F_0/k(t))(1 - e^{-k(t)t}) \) and \( h(0) = 0 \). we obtain that \( h = (F_0/k(t))(t + (1/k(t))e^{-k(t)t}) - F_0/K^2 \).

Assume that the
distance between two floors if \( H \). Then the time to traverse one level from stationary state will be
\[
t < \frac{Hk(t)}{F} + \frac{1}{K}.
\]

If we assume \( H = 2 \), we get \( t_{\text{max}} = 9.75 \) time units. The evolution kernel
\( \mathcal{P} \) is represented by the values at the head of the arrows in Figure B.5.

Figure B.5: Stochastic State Transition System of the elevator behaviour

To apply the verification rules, we need to:

I: Find a set of invariants for the average-timed \( \forall \)-automata in Figure 7.1(d) that satisfies the
invariance rules. Let us define \( I_B : Fb(3) = 1 \land \text{Com} \neq \text{down}, I_S : \text{Com} = \text{down}, \) and
\( I_R : Fb(3) = 0 \). It is easy to see that \( I_B, I_S \) and \( I_R \) are invariants for states \( B, S \) and \( R \),
respectively. Note that \( R = S \) in our example. In Figure B.5, bad states \( B \) are denoted by
empty nodes while recurrent/stable states \( R \) and \( S \) are denoted by filled nodes.

S: Find a set of Lyapunov functions. We omit the details but, in conjunction with the invariants
above, choosing a function \( \rho \), whose value is the average number of transitions for reaching
state \( R \), satisfies the Stability rules.

AT: We can show that choosing a global timing function \( \eta \) which corresponds to the average time
to reach $R$ satisfies the average-timeliness rules. For values of $\eta$ at each state, see the number in the state nodes in Figure B.5. It is easy to see that these values satisfy the rules for average-timeliness. Indeed, every value is less than the global bound on bad states $\tau(bad)$ and each transition from a state $s$ to another state $s'$ leads to a decrease in value for the global timing function.

We have showed that the average time for a passenger located at floor 1 to be taken to floor 3 is $29.6 < 40$ time units. Hence we have shown that the constraint on the elevator behaviour is satisfied. Note that this is not an absolute bound on the value. The completion time of an instance of the request may exceed 40 time units. With our method, we can automatically obtain probability bounds on the possible time of service. For the elevator example discussed here, we can show that $Pr(\text{time of service} > 90 \text{ time units}) \leq 0.12$. 
Appendix C

Code and output for the MDP systems

C.1 VAMP system

(variables (V one zero) (A one zero) (M one zero) (P zero one two three))

action a

V

(V (one (A (one (0.7 0.3)) (zero (0.51 0.49)))

(zero (A (one (0.6 0.4)) (zero (0.2 0.8)))))

A

(A (one (M (one (0.9 0.1)) (zero (0.3 0.7))))

(zero (M (one (0.37 0.63)) (zero (0.53 0.47)))))

M

(P (zero (0 1))

(one (1 0))

(two (0 1))

(three (1 0)))

P

(V (one (A (one (M (one (0 0 0 1)) (zero (0 0 1 0))))

(zero (M (one (0 0 1 0)) (zero (0 1 0)))))

(zero (A (one (M (one (0.0 1 0)) (zero (0 1 0)))))

(zero (M (one (0 1 0)) (zero (1 0 0)))))

260
endaction

action b

<table>
<thead>
<tr>
<th></th>
<th>V</th>
<th>A</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(V)</td>
<td>(one)</td>
<td>(one)</td>
</tr>
<tr>
<td></td>
<td>(zero)</td>
<td>(zero)</td>
<td>(zero)</td>
</tr>
<tr>
<td></td>
<td>(one)</td>
<td>(one)</td>
<td>(one)</td>
</tr>
<tr>
<td></td>
<td>(zero)</td>
<td>(zero)</td>
<td>(zero)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(P)</th>
<th>(P)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(P)</td>
<td>(P)</td>
</tr>
<tr>
<td></td>
<td>(P)</td>
<td>(P)</td>
</tr>
</tbody>
</table>

endaction

reward (P)

<table>
<thead>
<tr>
<th></th>
<th>(zero)</th>
<th>(zero)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(one)</td>
<td>(one)</td>
</tr>
<tr>
<td></td>
<td>(two)</td>
<td>(three)</td>
</tr>
</tbody>
</table>

discount 0.9000000

tolerance 0.01

C.2 VAM system

(variables (V one zero) (A one zero) (M one zero))

action a

<table>
<thead>
<tr>
<th></th>
<th>V</th>
<th>A</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(V)</td>
<td>(one)</td>
<td>(one)</td>
</tr>
<tr>
<td></td>
<td>(zero)</td>
<td>(zero)</td>
<td>(zero)</td>
</tr>
<tr>
<td></td>
<td>(one)</td>
<td>(one)</td>
<td>(one)</td>
</tr>
<tr>
<td></td>
<td>(zero)</td>
<td>(zero)</td>
<td>(zero)</td>
</tr>
</tbody>
</table>

261
Figure C.1: Value for the vamp PCN

\[
\begin{align*}
M & ((zero \ (0.3 \ 0.7))) \\
   & (M \ (one \ (0.37 \ 0.63))) \\
   & (zero \ (0.53 \ 0.47)))
\end{align*}
\]
C.3  Cat Mouse system

:variables (hasM yes no) (Mx zero one two three) (My zero one two)
Figure C.2: Value for the vam PCN

(Cx one two three) (Cy zero one two) (Cstate guarding resting playing))

action down
  hasM (hasM (yes (0.97 0.03))
  (no (0.0 1.0)))

264
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>(zero)</th>
<th>(one)</th>
<th>(two)</th>
<th>(three)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mx</td>
<td>(Mx)</td>
<td>Cstate</td>
<td>guarding</td>
<td>(0.9 0.1 0.0 0.0)</td>
<td>(0.9 0.1 0.0 0.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(resting)</td>
<td>(0.7 0.3 0.0 0.0)</td>
<td>(0.7 0.3 0.0 0.0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(playing)</td>
<td>(0.3 0.3 0.2 0.2)</td>
<td>(0.3 0.3 0.2 0.2)</td>
<td></td>
</tr>
<tr>
<td>My</td>
<td>(My)</td>
<td>(zero)</td>
<td>(0.4 0.4 0.2)</td>
<td>(0.4 0.4 0.2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(one)</td>
<td>(0.33 0.34 0.33)</td>
<td>(0.33 0.34 0.33)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(two)</td>
<td>(0.2 0.4 0.4)</td>
<td>(0.2 0.4 0.4)</td>
<td></td>
</tr>
<tr>
<td>Cx</td>
<td>(Cx)</td>
<td>(one)</td>
<td>(0.8 0.2 0.0)</td>
<td>(0.8 0.2 0.0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(two)</td>
<td>(0.15 0.7 0.15)</td>
<td>(0.15 0.7 0.15)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(three)</td>
<td>(0.0 0.2 0.8)</td>
<td>(0.0 0.2 0.8)</td>
<td></td>
</tr>
<tr>
<td>Cy</td>
<td>(Cy)</td>
<td>(zero)</td>
<td>(0.95 0.04 0.01)</td>
<td>(0.95 0.04 0.01)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(one)</td>
<td>(0.9 0.09 0.01)</td>
<td>(0.9 0.09 0.01)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(two)</td>
<td>(0.01 0.9 0.09)</td>
<td>(0.01 0.9 0.09)</td>
<td></td>
</tr>
<tr>
<td>Cstate</td>
<td>(Cstate)</td>
<td>(guarding)</td>
<td>(1.0 0.0 0.0)</td>
<td>(1.0 0.0 0.0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(resting)</td>
<td>(0.0 1.0 0.0)</td>
<td>(0.0 1.0 0.0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(playing)</td>
<td>(0.0 0.0 1.0)</td>
<td>(0.0 0.0 1.0)</td>
<td></td>
</tr>
</tbody>
</table>

endaction

action up

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>(yes)</th>
<th>(no)</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasM</td>
<td>(hasM)</td>
<td>(0.97 0.03)</td>
<td>(0.0 1.0)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>(zero)</th>
<th>(one)</th>
<th>(two)</th>
<th>(three)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mx</td>
<td>(Mx)</td>
<td>Cstate</td>
<td>guarding</td>
<td>(0.9 0.1 0.0 0.0)</td>
<td>(0.9 0.1 0.0 0.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(resting)</td>
<td>(0.7 0.3 0.0 0.0)</td>
<td>(0.7 0.3 0.0 0.0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(playing)</td>
<td>(0.3 0.3 0.2 0.2)</td>
<td>(0.3 0.3 0.2 0.2)</td>
<td></td>
</tr>
<tr>
<td>My</td>
<td>(My)</td>
<td>(zero)</td>
<td>(0.4 0.4 0.2)</td>
<td>(0.4 0.4 0.2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(one)</td>
<td>(0.33 0.34 0.33)</td>
<td>(0.33 0.34 0.33)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(two)</td>
<td>(0.2 0.4 0.4)</td>
<td>(0.2 0.4 0.4)</td>
<td></td>
</tr>
<tr>
<td>Cx</td>
<td>(Cx)</td>
<td>(one)</td>
<td>(0.8 0.2 0.0)</td>
<td>(0.8 0.2 0.0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(two)</td>
<td>(0.15 0.7 0.15)</td>
<td>(0.15 0.7 0.15)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(three)</td>
<td>(0.0 0.2 0.8)</td>
<td>(0.0 0.2 0.8)</td>
<td></td>
</tr>
</tbody>
</table>

265
| Cy | (Cy | (zero | (0.09 0.9 0.01)) |
|    |    | (one  | (0.01 0.09 0.9)) |
|    |    | (two  | (0.01 0.04 0.95)) |

| Cstate | (Cstate | (guarding | (1.0 0.0 0.0)) |
|        |        | (resting | (0.0 1.0 0.0)) |
|        |        | (playing | (0.0 0.0 1.0)) |

endaction

action left

| hasM | (hasM | (yes  | (0.98 0.02)) |
|      |      | (no   | (0.0 1.0))) |

| Mx | (Mx | (zero | (Cstate | (guarding | (0.9 0.1 0.0 0.0)) |
|    |    | (one  |        | (resting | (0.8 0.2 0.0 0.0)) |
|    |    | (two  |        | (playing | (0.4 0.3 0.2 0.1)))) |

| My | (My | (zero | (0.4 0.4 0.2)) |
|    |    | (one  | (0.33 0.34 0.33)) |
|    |    | (two  | (0.2 0.4 0.4))) |

| Cx | (Cx | (one  | (0.95 0.04 0.01)) |
|    |    | (two  | (0.9 0.05 0.05)) |
|    |    | (three | (0.15 0.8 0.05)) |

| Cy | (Cy | (zero | (0.95 0.04 0.01)) |
|    |    | (one  | (0.05 0.9 0.05)) |
|    |    | (two  | (0.01 0.04 0.95)) |

| Cstate | (Cstate | (guarding | (1.0 0.0 0.0)) |
|        |        | (resting | (0.0 1.0 0.0)) |
|        |        | (playing | (0.0 0.0 1.0)) |

endaction

action right

| hasM | (hasM | (yes  | (0.95 0.05)) |
|      |      | (no   | (0.0 1.0))) |
Mx
(zero (Cstate (guarding (0.7 0.2 0.1 0.0))
(resting (0.5 0.3 0.1 0.1))
(playing (0.2 0.3 0.3 0.2))))
(one (0.0 0.33 0.33 0.34))
(two (0.0 0.33 0.33 0.34))
(three (0.0 0.33 0.33 0.34))

My
(zero (0.4 0.4 0.2))
(one (0.33 0.34 0.33))
(two (0.2 0.4 0.4))

Cx
(one (0.05 0.8 0.15))
(two (0.05 0.05 0.9))
(three (0.01 0.04 0.95))

Cy
(zero (0.95 0.04 0.01))
(one (0.05 0.9 0.05))
(two (0.01 0.04 0.95))

Cstate
(guarding (1.0 0.0 0.0))
(resting (0.0 1.0 0.0))
(playing (0.0 0.0 1.0))

endaction

action grab

hasM
(yes (Mx
(zero (0.05 0.95))
(one (0.9 0.1))
(two (0.8 0.2))
(three (0.7 0.3))))

(no (Mx
(zero (0.0 1.0))
(one (Cx
(zero (Cstate (guarding (0.9 0.1))
(resting (0.7 0.3))
(playing (0.5 0.5))))
(two (Cstate (guarding (0.7 0.3))
(resting (0.4 0.6))
(playing (0.2 0.8))))
(three (Cstate (guarding (0.45 0.55))
(resting (0.25 0.75))
(playing (0.05 0.95))))
(two (Cx
(zero (Cstate (guarding (0.7 0.3))
(resting (0.4 0.6))
(playing (0.2 0.8))))
(two (Cstate (guarding (0.9 0.1))

267
Mx (Mx (zero (0.4 0.3 0.2 0.1))
(one (0.0 1.0 0.0 0.0))
(two (0.0 0.0 1.0 0.0))
(three (0.0 0.0 0.0 1.0)))

My (My (zero (1.0 0.0 0.0 ))
(one (0.0 1.0 0.0 ))
(two (0.0 0.0 1.0 )))

Cx (Cx (one (1.0 0.0 0.0))
(two (0.0 1.0 0.0))
(three (0.0 0.0 1.0)))

Cy (Cy (zero (1.0 0.0 0.0 ))
(one (0.0 1.0 0.0 ))
(two (0.0 0.0 1.0 )))

cage

Cstate (Cstate (guarding (1.0 0.0 0.0))
(resting (0.0 1.0 0.0))
(playing (0.0 0.0 1.0)))

endaction

action hasM

(hasM (yes (0.0 1.0))
(no (0.0 1.0))

Mx (hasM (yes (Mx (zero (0.5 0.3 0.2 0.0))
(one (0.7 0.2 0.1 0.0))

268
My (hasM
(two (0.5 0.3 0.2 0.0))
(three (0.4 0.3 0.2 0.1))))
(no (Mx (zero (0.0 0.33 0.33 0.34))
(one (0.0 1.0 0.0 0.0))
(two (0.0 0.0 1.0 0.0))
(three (0.0 0.0 0.0 1.0))))

Cy (Cy
(zero (0.3 0.3 0.3 0.3))
(one (0.3 0.3 0.3 0.3))
(two (0.3 0.3 0.3 0.3)))

Cx (Cx
(one (1.0 0.0 0.0))
(two (1.0 0.0 0.0))
(three (1.0 0.0 0.0)))

endaction

action rest

hasM (hasM
(yes (0.55 0.45))
(no (0.0 1.0)))

Mx (Mx (zero (Cstate (guarding (0.7 0.3 0.0 0.0))
(resting (0.6 0.3 0.05 0.05))
(playing (0.3 0.3 0.2 0.2))))
(one (0.0 0.33 0.33 0.34))
(two (0.0 0.33 0.33 0.34))
(three (0.0 0.33 0.33 0.34)))

My (0.33 0.33 0.34)

Cx (CX
(one (1.0 0.0 0.0))
(two (0.0 1.0 0.0))
(three (0.0 0.0 1.0)))

Cy (Cy
(zero (1.0 0.0 0.0))

Cstate (0.0 1.0 0.0)
endaction

action guard

hasM  (hasM  (yes  (0.7 0.3))
(no  (0.0 1.0))

Mx  (Mx  (zero  (Cstate  (guarding  (0.9 0.1 0.0 0.0))
(resting  (0.7 0.3 0.0 0.0))
(playing  (0.5 0.3 0.2 0.0))
(one  (0.0 0.33 0.33 0.34))
(two  (0.0 0.33 0.33 0.34))
(three  (0.0 0.33 0.33 0.34))

My  (0.33 0.33 0.34)

Cx  (Cx  (one  (1.0 0.0 0.0))
(two  (0.0 1.0 0.0))
(three  (0.0 0.0 1.0))

cy  (Cy  (zero  (1.0 0.0 0.0))
(one  (0.0 1.0 0.0))
(two  (0.0 0.0 1.0))

Cstate  (1.0 0.0 0.0)
endaction

action play

hasM  (hasM  (yes  (0.0 1.0))
(no  (0.0 1.0))

Mx  (Mx  (zero  (Cstate  (guarding  (0.7 0.3 0.0 0.0))
(resting  (0.5 0.4 0.05 0.05))
(playing  (0.2 0.3 0.3 0.2))
(one  (0.0 0.33 0.33 0.34))
(two  (0.0 0.33 0.33 0.34))
(three  (0.0 0.33 0.33 0.34))

My  (0.33 0.33 0.34)

270
Cx (Cx (one (1.0 0.0 0.0))
  (two (0.0 1.0 0.0))
  (three (0.0 0.0 1.0)))

Cy (Cy (zero (1.0 0.0 0.0))
  (one (0.0 1.0 0.0))
  (two (0.0 0.0 1.0)))

Cstate (0.0 0.0 1.0)
endaction
reward (hasM (yes (Mx (zero (O)))
  (one (Cstate (guarding (3))
    (resting (6))
    (playing (-1))))
  (two (Cstate (guarding (2))
    (resting (4))
    (playing (-2))))
  (three (Cstate (guarding (1))
    (resting (2))
    (playing (-3)))))
(no (Mx (zero (Cstate (guarding (5))
    (resting (20))
    (playing (50))))
  (one (Cstate (guarding (-5))
    (resting (-10))
    (playing (-40))))
  (two (Cstate (guarding (-5))
    (resting (-10))
    (playing (-40))))
  (three (Cstate (guarding (-5))
    (resting (-10))
    (playing (-40)))))

discount 0.9000000
tolerance 0.01

271
Figure C.3: Value function for the cat mouse PCN


275


276


280


283


Index

$T_2$ space, see Hausdorff space

$\forall$-automaton
  Accepting run, 120
  Complete, 120
  Run, 120
  Semantics, 121
  Syntax, 120

Algebra, 35
  $\sigma$-algebra, 35

Algebraic loop, 79

Attraction, 175
  Exponential, 175

Average-timed $\forall$-automaton
  Accepting run, 125
  Average-timing function, 125
  Global time constraint, 125
  Local time constraint, 125
  Run, 125
  Semantics, 125
  Syntax, 125

Behaviour, 126
  Complexity, 129
  Equivalent, 126
  Markovian, 127
  Probability measure on, 128
  Time-invariant, 127

Brownian motion, 42

Causality, 54

Control Synthesis Problem
  Closed-loop, 174
  Feedback, 174
  Open-loop, 174

Convergence
  In distribution, 40
  In law, 40

Cpo, see Complete partial order

CTMC, 107

Delay
  Transport, 57
  Unit, 57

Distance function, see Metric

Domain
  Composite, 47
  Simple, 46

Domain structure, 48

DTMC, 99

DTMP, 106

DTP, 180

Dynamics structure, 59

Elementary outcome, 28

Equilibrium point, 175

Event generator, 88

Event synchronizer, 90

Event-driven transduction, 59

Extension stochastic trace, 58
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filtration</td>
<td>35</td>
</tr>
<tr>
<td>Fixpoint</td>
<td></td>
</tr>
<tr>
<td>In distribution</td>
<td>74</td>
</tr>
<tr>
<td>Least</td>
<td>74</td>
</tr>
<tr>
<td>FPATTL</td>
<td></td>
</tr>
<tr>
<td>Frame</td>
<td>170</td>
</tr>
<tr>
<td>Model</td>
<td>170</td>
</tr>
<tr>
<td>Semantics</td>
<td>171</td>
</tr>
<tr>
<td>Syntax</td>
<td>170</td>
</tr>
<tr>
<td>Terms</td>
<td>169</td>
</tr>
<tr>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>Additive</td>
<td>36</td>
</tr>
<tr>
<td>Countably</td>
<td>36</td>
</tr>
<tr>
<td>Continuous</td>
<td>29</td>
</tr>
<tr>
<td>Measurable</td>
<td>37</td>
</tr>
<tr>
<td>Monotonic</td>
<td>33</td>
</tr>
<tr>
<td>Pathwise continuous</td>
<td>29</td>
</tr>
<tr>
<td>Strict</td>
<td>48</td>
</tr>
<tr>
<td>Strict extension</td>
<td>48</td>
</tr>
<tr>
<td>Well-defined</td>
<td>49</td>
</tr>
<tr>
<td>Generator</td>
<td>55</td>
</tr>
<tr>
<td>Hausdorff space</td>
<td>30</td>
</tr>
<tr>
<td>Hidden Input/Output</td>
<td>67</td>
</tr>
<tr>
<td>Hierarchical set</td>
<td>94</td>
</tr>
<tr>
<td>Remainder set</td>
<td>94</td>
</tr>
<tr>
<td>Hybrid system</td>
<td>88</td>
</tr>
<tr>
<td>Infinitesimal</td>
<td>81</td>
</tr>
<tr>
<td>Input/output pair</td>
<td>126</td>
</tr>
<tr>
<td>Interpretation</td>
<td>148</td>
</tr>
<tr>
<td>Itô formula</td>
<td>93</td>
</tr>
<tr>
<td>Itô process</td>
<td></td>
</tr>
<tr>
<td>seeSDE</td>
<td>66</td>
</tr>
<tr>
<td>Itô-Taylor expansion</td>
<td>95</td>
</tr>
<tr>
<td>Lattice</td>
<td>33</td>
</tr>
<tr>
<td>Complete</td>
<td>33</td>
</tr>
<tr>
<td>LCI</td>
<td>5</td>
</tr>
<tr>
<td>Limit in distribution</td>
<td>40, 51</td>
</tr>
<tr>
<td>Lyapunov function</td>
<td>136</td>
</tr>
<tr>
<td>Markov chain</td>
<td></td>
</tr>
<tr>
<td>Continuous time</td>
<td>108</td>
</tr>
<tr>
<td>Discrete time</td>
<td>99</td>
</tr>
<tr>
<td>MDP</td>
<td>182</td>
</tr>
<tr>
<td>Extensional representation</td>
<td>185</td>
</tr>
<tr>
<td>Intensional representation</td>
<td>186</td>
</tr>
<tr>
<td>Policy</td>
<td>182</td>
</tr>
<tr>
<td>Reward function</td>
<td>182</td>
</tr>
<tr>
<td>Measurable space</td>
<td>36</td>
</tr>
<tr>
<td>Measure</td>
<td>36</td>
</tr>
<tr>
<td>Measure space</td>
<td>36</td>
</tr>
<tr>
<td>Metric</td>
<td>34</td>
</tr>
<tr>
<td>Module</td>
<td>67</td>
</tr>
<tr>
<td>Closed</td>
<td>67</td>
</tr>
<tr>
<td>Open</td>
<td>67</td>
</tr>
<tr>
<td>Semantics</td>
<td>77</td>
</tr>
<tr>
<td>Sequential module</td>
<td>92</td>
</tr>
<tr>
<td>Multiple index</td>
<td>94</td>
</tr>
<tr>
<td>Empty multiple index</td>
<td>94</td>
</tr>
<tr>
<td>Length</td>
<td>94</td>
</tr>
<tr>
<td>Total number</td>
<td>94</td>
</tr>
<tr>
<td>Parameterized PCN</td>
<td>80</td>
</tr>
<tr>
<td>Limiting Semantics</td>
<td>81</td>
</tr>
<tr>
<td>Module</td>
<td>80</td>
</tr>
<tr>
<td>Robustness</td>
<td>129</td>
</tr>
<tr>
<td>Semantics</td>
<td>81</td>
</tr>
<tr>
<td>Partial order</td>
<td></td>
</tr>
<tr>
<td>Complete</td>
<td>32</td>
</tr>
</tbody>
</table>

286
Partial order
  Directed subset, 32
  Flat partial order, 31
  Greatest element, 31
  Greatest lower bound (glb), 32
  Least element, 31
  Least upper bound (lub), 32
  Linear order, 31
  Product partial order, 31
  Sub-partial order, 31
  Total order, 31

PATL
  Frame, 148
  Model, 148
  Satisfaction, 149
  Satisfiable, 146
  Semantics, 149
  Syntax, 145

PATTL
  Real-time operators, 152
  Size of operator, 153

PCN, see Probabilistic Constraint Net
Poisson process, 89
  Memoryless property, 90
  Regeneration, 90

POMDP, 181, 206

Probabilistic Constraint Net
  Clock, 64
  Closed, 65
  Connections, 64
  DC-PCN, 106
  DD-PCN, 99
  Location, 64
  Input, 64

Output, 64
  Open, 64
  Semantics, 75
  Subnet, 67
  Synch-fin, 197
  Syntax, 64
  Probability distribution function, 38
  Probability measure, 37
  Probability space, 37

Random variable, 37

Relation
  Partial order, 31
Requirements specification
  Bounded response, 122
  Reachability, 122
  Safety, 122
  Reversible system, 164
  Robotic system, 3

Sample space, 28
Sample stochastic trace, 58
SDE, 43
Signature, 48
  Function Symbol, 48
  Mapping type, 48
  Sort, 48
  Stability, 175
    Asymptotic, 175
    Exponential, 175
  Stochastic differential equation
    Existence and uniqueness, 96
  Stochastic event space, 53
  Stochastic event trace, 52
  Stochastic linear set, 40