Quantitative Verification of Real-Time Properties with Application to Medical Devices

Marco Diciolla
Trinity College,
Oxford

A Thesis submitted for the degree of
Doctor of Philosophy in Computer Science
Hilary Term 2014

Abstract

Probabilistic model checking is a powerful technique used to ensure the correct functioning of systems which exhibit real-time and stochastic behaviours. Many such systems are embedded and used in safety-critical situations, to mention implantable medical devices. This thesis aims to develop a formal model-based framework that is tailored for the analysis and verification of cardiac pacemakers. The contributions are novel approaches for the automatic verification and validation of real-time properties over continuous-time models, which are applicable to software embedded in medical devices.

First, we address the problem of model checking *continuous-time Markov chain* (CTMC) models against real-time specifications given in the form of temporal logic, namely, *metric temporal logic* (MTL) and *linear duration properties* (LDP), or as timed automata (TA). The main question that we address is "given a continuous-time Markov chain, what is the probability of the set of timed paths that satisfy the real-time property under consideration?". We provide novel algorithms to approximate the probability through generating systems of linear inequalities over variables that represent the waiting times in system states, and then solving multidimensional integrals over this set.

Second, we present a model-based framework to support the design and verification of pacemakers against real-time properties. The pacemaker is modelled as a network of timed automata, whereas the human heart is modelled either as a network of timed automata or as a network of hybrid automata. Our framework can be instantiated with personalised heart models whose parameters can be learnt from patient data, and we have done so [LB13] to validate our approach. We introduce property patterns and the *counting metric temporal logic* (CMTL) in order to specify the properties of interest. We provide new verification algorithms for networks of timed or hybrid automata against property patterns and CMTL. Finally, we pose and solve the parameter synthesis problem, i.e., given a network of timed automata containing model parameters, an objective function and a CMTL formula, find the set of parameter valuations, whenever existing, which satisfy the CMTL formula and maximise the objective function.

The framework has been implemented using Simulink, Matlab and Python code. Extensive experimental results on pacemaker models have been carried out and discussed in detail. The techniques developed in this thesis can assist in the design and verification of

software embedded in medical devices.

Acknowledgements

This work would not have been possible without the help of many people.

First, I would like to thank my supervisor, Professor Marta Kwiatkowska. Marta has provided invaluable guidance, support and critique throughout the three years of my DPhil in Oxford. She had confidence in me when I doubted myself and brought the good ideas out of me. She has been constantly helpful and spent a lot of her valuable time to generously guide me with this research project. She gave me a chance of pursuing a DPhil in one of the most prestigious University in the world. I will always be grateful to her for that. Without my supervisor this thesis would simply not exist.

I would also like to express my appreciation to my colleagues and friends Alexandru Mereacre and Taolue Chen. With them I spent many sleepless nights working on academic problems (and not only). Taolue has showed me that working hard and perseverance almost always pay off in the end. Alexandru has been more than a colleague; he has been a real friend. I have spent with him most of the time of my DPhil. We have worked hard, coded, written papers, but also had many crazy nights and parties together. I hope our friendship continues even after my DPhil.

My thanks also go to all the colleagues and collaborators inside and outside the department which made sure that my time spent working (and not) was fun.

Of course, I cannot conclude the acknowledgements section without mentioning my partner, Anastasia. She has been understanding like no others. She spent days and nights in my office while I was working on research topics with my colleagues. She has sacrificed many of her weekends for the sake of my DPhil. Thank you for being so supportive and patient.

Last but not least, I would like to express my gratitude to my family. Each member of my family has contributed to this thesis in his/her own way. Thank you all for believing in me.

This work was supported by ERC advanced grant VERIWARE (http://www.veriware.org).

Alla mia famiglia

Questa tesi e' dedicata alla mia famiglia,
fidati compagni del mio campo di battaglia.

Il campo di battaglia, per chi non l'avesse capito,
e' il mio dottorato ... hai recepito?

Mamma, papa' e fratelli
son stati a me vicino in momenti brutti e belli.

Seppur la strada non sempre rosea e' stata
con la lor compagnia la corsa fu meno affannata.

A ciascun di loro meriti furon attribuiti,
il minimo che posso far e' che i loro nomi sian ben scanditi

Padre Raimondo e madre Maria,
fecero da balia a questa vita mia.

Da mio padre appresi che calma, sacrificio ed impegno,
son le virtu' che di esser uomo ti fan degno.

Lui che da umili origini il cammin fu segnato,
mi insegno' che nella vita non bisogna dar nulla per scontato.

Non ti rassegnar difronte alle difficolta', un giorno mi disse,
la vita e' bella perche' ti stupisce.

Non giudicare gli altri tuoi pari,
cerca di comprenderli invece che marcarli come tuoi avversari.

Che dire ora di della donna che in grembo mi porto' per nove mesi
per lei saro' sempre il cuore di mamma ... ci siamo intesi??

Le sue cure, le sue attenzioni e le sue carezze,
han piu' di una volta scacciato via le mie amarezze.

Se di conforto avevo bisogno di cercare,
le sue braccia erano li, pronte ad aspettare.

Il suo gentile abbraccio e' stato per me riparo,
sai? nei momenti difficili, questo e' un dono raro.

Giuseppe e Gianluca so' i nomi dei due fratelli,
entrambi loro a me serviron da modelli.

Il loro incoraggiamento ed il loro grande cuore,
han fatto di me un uomo migliore.

Per concluder, permettetemi di lasciare un messaggio banale ...
lo so, un semplice grazie non sembra davvero nulla di speciale.

Abbiate pazienza pero', nato non fui per esser scrittore,
accettate questa modesta poesia ... vi assicuro che e' stata scritta con amore.

Marco
Oxford
30/01/2014

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The aim of this dissertation is to provide a theoretical framework which enables formal verification of medical devices with respect to real-time properties. In this chapter we introduce the main concepts that are relevant for the thesis, such as *model checking*, *stochasticity*, *hybrid systems*, *real time* and *medical devices*. We also state in Section 1.1 the main contribution with respect to the literature and highlight the structure of the thesis in Section 1.2. We conclude the chapter with Section 1.3, where we present a brief summary of the contributions of the author of this thesis to published joint-authored papers.

Model checking

Our reliance on the correct functioning of electronic systems is growing rapidly. High speed trains, autonomous cars, Internet banking, aeroplanes and smartphones are just a few examples of the myriad of complex systems that surround our daily life. We expect them to function flawlessly and we heavily rely on their outputs. A failure in our Internet banking system could result in severe financial loss. A glitch in the code employed by financial firms to perform high-frequency algorithmic transactions could make millions of pounds vanish in just a few seconds. It is not all about money though. Financial losses and personal disappointments, although annoying, do not constitute a threat to our lives. In some cases, errors in the software could be catastrophic. Think, for example, of the software embedded in medical devices, such as pacemakers. Pacemakers must work correctly 24 hours per day, seven days per week. A fault in the device could cause patient discomfort and in the most pessimistic case even death. For such a reason, researchers have focused their attention on *formal verification* techniques to gain trust in the software embedded in such complex systems. Formal verification aims to establish system correctness through mathematical rigour.

A very successful formal verification technique is *model checking*. Model checking is an automated technique that, given a finite-state model of a system and a formal property

statement, systematically checks whether this property is true in the model of the system. A model checker, the tool that performs model checking, explores all possible system states in a brute-force manner, essentially checking all possible system scenarios to determine whether the property holds or not. Using clever algorithms and tailored data structures, very large state spaces, up to even 10^{20} states and beyond, have been model checked [BK08, BCM+90].

Stochasticity

The term *stochastic* is related to anything *pertaining to chance* and comes from the Greek word "stokhastikos". It is used to describe subjects that contain randomness or uncertainty. The difference between a deterministic system and a stochastic one is that, in the former case, for any given input the system returns the same output; in the latter case, the same input may produce different outputs distributed according to some probability distribution. Stochastic systems are ubiquitous in *physics* (any physical system is subject to uncertainty), *biology* (e.g., binding and unbinding of RNA polymerase to a promoter), *artificial intelligence* (to solve problems such as simulated annealing, stochastic neural networks and stochastic optimization), *medicine* (e.g., stochastic effect, or "chance effect" is one classification of radiation effects that refers to the random, statistical nature of the damage) and *computer science* (e.g., randomised Internet protocols and Bluetooth technology).

Hybrid systems

Hybrid systems [ACHH92] combine discrete events and continuous-time dynamics and can serve as models of a variety of systems. Their expressiveness is powerful enough to describe highway systems [LGS96], air traffic management systems [PJ08], unmanned aerial vehicles [KHM+98], manufacturing and embedded systems [CPW01]. For example, hybrid systems arise in embedded control when digital controllers, computers and subsystems modelled as finite-state machines are coupled with controllers and plants modelled by partial or ordinary differential equations. Thus, such systems arise whenever one combines logical decision making with the generation of continuous-valued control laws. We show here one of the real-world examples of hybrid systems, taken from [Bra05], namely a thermostatically-controlled room.

Thermostatically-controlled room. A thermostatically-controlled room can be modelled as the hybrid automaton (see Section 5.2.2 for the definition of a hybrid automaton) in Figure 1.1. We write x for the variable representing the actual temperature of the room and we write $\dot{x}$ to express its first derivative. The dynamics of the system evolves as follows. It starts with the furnace turned on and the temperature of 10 degrees (the initial condition $x = 10$ is omitted in the automaton). The temperature starts increasing at con-

stant rate of 1 unit per time from the point when the furnace is turned on. When the room reaches 30 degrees, the furnace is turned off and the temperature decreases at constant rate of -1 unit per time. If the value of the temperature drops down to 10 degrees, the system re-activates the furnace. The notation $!(x \geq c)$ denotes that the transition *must* be taken when enabled.

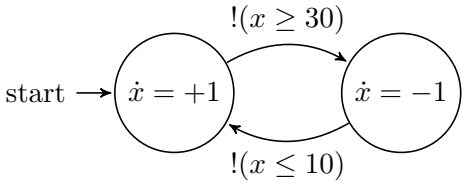


Figure 1.1: Automaton of the thermostatically-controlled room

Real time

According to the definition given in [Hen91] *real-time* systems are:

(A) Pertaining to the processing of data by a computer in connection with another process outside the computer according to the time requirements imposed by the outside process (this term is also used to describe systems operating in conversational mode and processes that can be influenced by human intervention while they are in progress); or

(B) Pertaining to the actual time during which a physical process evolves, for example, the performance of a computation during the actual time that the related physical process takes place, in order that results of the computation can be used in guiding the physical process.

Some real-world systems meet the task they have been designed to accomplish only if they relate properly with the passage of time.

Consider, for instance, the thermostatically-controlled room in Figure 1.1. An example of a real-time property that one may want to check in the system of Figure 1.1 is that in the next 3 hours the temperature in the room never drops to 10 degrees.

Predicting the behaviour of real-time systems by mere inspection is difficult and often impossible. Therefore, real-time systems are a prime target for formal verification.

Logical formalisms, e.g., *Temporal Logic*, have provided crucial help in analysing real-time systems and their behaviours. However, one shortcoming of conventional temporal logic is that it admits only the treatment of qualitative timing requirements, such as the demand that an event occurs "eventually". Due to this limitation, standard temporal logic is inadequate for the study of real-time systems, whose correctness depends crucially on the actual times at which events occur. More powerful real-time formalisms must generalise

the temporal logic methodology to encompass the analysis of real-time behaviour. In this thesis we consider specifications that achieve this goal, namely, *Metric Temporal Logic* (MTL) and variants.

Medical devices

A medical device is an instrument that is used to diagnose, treat or prevent a disease. The difference between a medical device and a drug is the way in which medical devices operate. Specifically, medical devices act by physical, mechanical or thermal means, rather than achieving their purpose through chemical actions within or on the body.

Medical devices vary greatly in complexity and application. Examples range from simple tongue depressors and medical thermometers to more complex pacemakers and neurostimulators. The global medical device market is nowadays estimated to be around 150bn dollars and it will expand to reach 250bn dollars by 2017 [Wik].

It is clear that one feature shared across most of medical devices is that a fault in the system or in the embedded software could be dangerous. Thus, the benefits of effective verification and validation activities in the medical device domain would include: increased usability and reliability; decreased failure rate and recalls; and reduced risks to patients and users.

In this thesis we focus our attention on implantable medical devices. Implantable medical devices, such as cardiac pacemakers, must be designed and programmed to the highest levels of safety and reliability. Unfortunately, according to the US Food and Drug Administration (FDA), errors in embedded software have led to a substantial increase in safety alerts, costly device recalls or even patient death. Combined with the relative lack of standardisation in the field of medical devices, there is an urgent need to develop methodologies for ensuring correct behaviour of embedded pacemaker software. The goal of this thesis is to provide a framework which is capable of performing automated quantitative verification of pacemaker software.

1.1 Contribution

The technical contribution of this thesis can be summarised as follows.

- We propose approximate verification algorithms for continuous-time Markov chains (CTMC) against real-time properties specified either as Metric Temporal Logic formulas, as Timed Automata, or as Linear Duration Properties. Our algorithms use a new technique consisting of generating systems of linear inequalities over variables which represent the waiting times in system states and then solving multidimensional integrals over this set. Each verification algorithm is presented with complexity analysis, error bounds on the probability of satisfaction of a given formula in the system and a prototype implementation in Matlab [MAT13].

- We propose a model-based framework to support the design and validation of implantable medical devices such as cardiac pacemakers. The components of the framework are: a model of the human heart, a model of the pacemaker and a property specification to check. The pacemaker is modelled as a network of Timed Automata, whereas the human heart is modelled either as a network of Timed Automata or as a network of Hybrid Automata. Our framework can be instantiated with personalised heart models, whose parameters can be learnt from real data [LB13]. We introduce property patterns and a generalisation of MTL called *Counting Metric Temporal Logic* (CMTL) in order to specify properties of interest such as average beat rate of the human heart and energy consumption of the pacemaker. We provide new verification algorithms for networks of Timed or Hybrid Automata against property patterns and CMTL. Finally, we pose and solve the parameter synthesis problem, i.e., given a network of Timed Automata containing model parameters, an objective function and a CMTL formula, find the parameter valuations which satisfy the CMTL formula and maximise the objective function.

- We implement the framework using Simulink [SIM13], Matlab and Python code and present extensive experimental results on pacemaker models.

Both theoretical and practical contribution can impact the research community in the longer term in the following way. The algorithms presented in Chapter 4 to model check continuous-time Markov chains against real-time specifications can be optimised in order to deal with real-life examples. For instance one could introduce clever Monte-Carlo techniques, such as important sampling, in order to deal with the state space explosion originated from the path enumeration. Once the complexity of the verification algorithms is reduced, similar techniques to the one presented in this thesis could be applied to different real-time models to solve real-life problems.

Chapter 5 constitutes the basis of what the author sees as first attempt to generate a model based framework tailored for the verification of medical devices. We recognise that from a medical perspective, pacemakers (which are main focus of Chapter 5) are very simple devices which work correctly in most cases. For such a reason, the medical community is usually interested in more complex medical devices and/or problems, such as defibrillators and ablation (which is not considered in this thesis). We emphasise here that the techniques introduced in Chapter 5 still apply with minor modifications to the verification and design of such complex medical devices. More specifically, the framework in Chapter 5 can be instantiatied with a model of a defibrillator (once such model is defined) and verify real-time properties of defibrillators rather than pacemakers. Thus, in the longer term, researchers can reuse the framework presented in this thesis, or an augmented version of it, and work alongside to doctors in order to improve the quality and trust of medical devices.

1.2 Structure of the dissertation

This dissertation is organised as follows.

- **Chapter 2.** The chapter provides a broad overview of the literature which is closely related to the contents of this thesis. Specifically, we summarise related work on probabilistic model checking, real-time logics, continuous-time Markov chains and parameter synthesis.

- **Chapter 3.** The chapter introduces the background material necessary to understand this dissertation. We describe the models that we use in later chapters, such as discrete-time Markov chains and continuous-time Markov chains. We also present the real-time formalisms that we use to specify properties in this thesis, namely, Linear Temporal Logic, Metric Temporal Logic, Linear Duration Properties and Timed Automata.

- **Chapter 4.** The chapter describes model checking techniques for continuous-time Markov chains over properties specified either as Metric Temporal Logic formulas, as Timed Automata, or as Linear Duration Properties. The central question that we address is: given a CTMC $\mathcal{C}$ and a real-time property φ, "what is the probability of the set of timed paths of $\mathcal{C}$ that satisfy the property φ over a time interval of fixed, bounded length?". The link between this chapter and the next chapter on medical devices is twofold. First, the models analysed in the two chapters are continuous-time models. Second, there is a strong connection between the solution techniques that we use in both chapters. In particular, the techniques developed in Chapter 4 have proved a valuable tool for model checking medical devices, in that they were adopted with minor changes to tackle problems specific to their safety and energy efficiency. Although Chapter 4 and Chapter 5 share multiple similarity, they also have substantial differences. The techniques presented in Chapter 4 are by far analytical. The reason for that is principally the choice of the model formalism. CTMCs are models with nice mathematical properties, such as the exponential residence time in system states and the memoriless property, which are amenable for analytical solutions. It is not the case for the more complex models of networks of Timed I/O Automata and Hybrid I/O Automata introduced in Chapter 5 for which already back in the 90's it was shown that the simple reachability property is undecidable for Hybrid Automata, see Henzinger *et al.* in [HKPV95] for details. For such a reason, in this thesis, we resort to approximation techniques in order to answer the model checking questions of interest.

- **Chapter 5.** The chapter tackles two main problems: model checking networks of Hybrid I/O Automata; and synthesising model parameters for networks of Timed

I/O Automata. The model checking problem takes as input a network of Hybrid I/O Automata which represents the human heart, a network of Timed I/O Automata which represents a pacemaker specification, a property pattern which represents a safety property that we want to verify, and applies approximate verification algorithms to determine the probability of the property being satisfied in the composed system. The parameter synthesis problem takes as input a network of Timed I/O Automata for modelling the human heart and the pacemaker, a Counting Metric Temporal Logic formula and an objective function. The algorithms then finds the parameter valuations such that the objective function is maximised and the Counting Metric Temporal Logic formula is satisfied in the system. As mentioned earlier, some of the techniques developed in Chapter 4 are reused here in Chapter 5. We emphasise here that we needed new logic formalisms different from the ones introduced in the previous chapter. The logic formalisms that we consider are purely driven by the application domain in which we operate. It is clear that when talking about pacemakers, the simplest features that one needs to monitor is that the number of heart beats is within a given safety bound. Such a property already hints for the need of a logics which is capable of counting, and hence the introduction of Counting Metric Temporal Logic and property patters.

- **Chapter 6.** The chapter summarises the contributions of this dissertation and suggests some future research directions.

1.3 Published papers and contribution to joint-authored articles

The two core chapters of the thesis, namely Chapter 4 and Chapter 5, are based on a series of published and submitted conference and journal papers which are joint work with my colleagues in Oxford.

- **Chapter 4.** The contents of this chapter is based on two conference papers [CDKM11, CDKM12b] and one journal publication [CDKM13b]. The author of this thesis has worked jointly with the other authors of [CDKM11, CDKM12b, CDKM13b] to develop algorithms and proofs. The Matlab implementation of [CDKM11] has been entirely developed and coded by the author of this thesis and likewise the numerical examples in [CDKM11] have also been designed and solved by the author. The extension to prefix-accumulation assertions of [CDKM13b] is mainly the work of the author of this thesis.

- **Chapter 5.** The contents of this chapter is based on two published conference papers [CDKM12a, CDKM13c], one report awaiting submission [DKM13] and one

journal publication [CDKM13a]. The author of this thesis has worked jointly with the other authors of [CDKM12a, CDKM13c, CDKM13a] to develop the algorithms and proofs. The Matlab implementation and experimental results of [CDKM12a] are joint work of the author of this thesis and his colleague Alexandru Mereacre. The Simulink implementation and experimental results of [CDKM13c, CDKM13a] are joint work of the author of this thesis and his colleague Alexandru Mereacre. The author of this thesis has worked jointly with the other authors of [DKM13] to develop the parameter synthesis algorithms. The technical proofs of [DKM13] are mainly the work of the author of this thesis, whereas the Python implementation is the work of his colleague Alexandru Mereacre.

Chapter 2

Literature review

In this chapter we give a broad overview of the literature that is closely related to the topics of this thesis. More specifically, the chapter is divided into five sections: *probabilistic model checking* (see Section 2.1), *real-time specifications* (see Section 2.2), *parameter synthesis* (see Section 2.3), *model checking continuous-time Markov chains* (see Section 2.4) and *model checking medical devices* (see Section 2.5). In each section we describe the main results in the field, highlighting the main contributions of this thesis.

2.1 Probabilistic model checking

Model checking [BK08, CES86, CGP99] is an automated technique to determine whether a model of a system satisfies a property specification, which is usually provided as a temporal logic formula. The model checking algorithm explores all possible system states in a brute-force manner [BK08]. As a result, it is possible to show that a given system model satisfies a certain property or produces a counterexample if it does not. In the past, much attention has been focused on model checking for qualitative properties. For instance, model checking temporal logics has been addressed in [CES86] and [SC85]. In [CES86] the authors give an efficient procedure for verifying that finite-state systems meet specifications expressed as *Computational Tree Logic* (CTL). In [SC85] the authors propose model checking algorithms for *Linear Temporal Logic* (LTL) formulas. CTL and LTL are capable of expressing respectively, branching and linear-time properties that system executions should satisfy.

The success of model checking techniques for temporal specifications has driven researchers to extend the model checking algorithms to probabilistic models, i.e., models in which the jumps between system states happen according to given probability distributions. This field is known as *probabilistic model checking*. Probabilistic model checking is an automatic technique that provides a means to model and analyse systems that exhibit probabilistic behaviours against a range of quantitative properties usually expressed in

variants of temporal logics, such as the *Probabilistic Computational Tree Logic* (PCTL) [HJ94], the *Continuous Stochastic Logic* (CSL) [BHHK03], and their extensions. The simplest probabilistic models are *Discrete-Time Markov Chains* (DTMCs). DTMCs are basically labelled transition systems augmented with discrete probability distribution on the transitions. The problem of model checking DTMCs against temporal specifications has been addressed in [Var85, CY95, LS83, CY88, HJ94]. Slightly more complex probabilistic models are *Continuous-Time Markov Chains* (CTMCs). CTMCs are essentially DTMCs whose residence times in system states are exponentially distributed. Algorithms for model checking linear and branching time temporal logics over CTMCs have been presented in [ASSB00, BHHK03].

Probabilistic model checking tools, e.g., PRISM [PRI], use a high-level modelling language (reactive modules for PRISM [KNP11]) to describe the system and then check it against specifications expressed in a probabilistic temporal logic, for example CSL [BHHK03]. For instance, the model checker PRISM has been successfully used to analyse systems modelled as DTMCs, such as Hermans self-stabilisation algorithm [KNP12] for ring networks and the Bluetooth device discovery protocol [DKNP06], as well as systems that can be modelled as CTMCs, such as the DNA walkers of [DHK13, DKTT13].

2.2 Real-time temporal logics

As discussed in [Hen91], a vast range of real systems meet the task they have been designed to accomplish only if they relate properly with the passage of time. Examples of such systems are pacemaker devices which need to send their electric impulses within a certain time delay from the last sensed event, as otherwise their correct behaviour is compromised; circuits and communication protocols whose correctness depends on gate delays and message delays; and aeroplanes that need to react quickly to pilots' manoeuvres in order to avoid possible dangerous collisions. Therefore, researchers have been developing real-time temporal logics.

The main difficulty when analysing real-time systems is the *dense* nature of time. In fact, it is impossible to directly explore every possible system state at any given time due to the fact that there would be infinitely many time points to analyse. For this reason, symbolic verification techniques are typically employed for real-time systems. The problem of model checking real-time temporal logics becomes even more challenging when dealing with probabilistic real-time systems, such as the model of continuous-time Markov chains. Researchers have already proposed different algorithms to model check the validity of real-time specifications on probabilistic and non-probabilistic systems. For example, the authors of [BHHK03] introduce CSL, which is a branching-time temporal logic similar to CTL, with state and path formulas, where the CTL universal and existential path quantifiers are replaced by a probabilistic operator. In [BHHK03] the authors show how

to evaluate CSL formulas on CTMCs. In [Koy90] the authors introduce *Metric Temporal Logic* (MTL), an extension of LTL that allows one to express time constraints at which events happen. For example, with the logic MTL one can express the property that "every pacemaker beat is followed by a natural heart beat within 1 second unless a new pacemaker beat happens before". Metric Temporal Logic formulas have been evaluated against non-probabilistic systems [AFH96], such as *Timed Automata* (TA) [AD94]. Timed automata are labelled transition systems augmented with clocks, i.e., real-time variables that track the passage of time, and guards on transitions which constrain the time at which jumps between system states can happen.

Many important properties, however, depend on the cumulated time that the system spends in certain states, possibly intermittently. Such *duration* properties, following the terminology of *Duration Calculus* (DC) [CHR91], have been studied in the context of Timed Automata [ACH97, BES93, KPSY99]. When evaluated on probabilistic real-time models, duration properties can express, e.g., that "the probability of an alarm bell ringing whenever the button has been pressed, possibly intermittently, for at least 2 seconds in total is at least 95% ".

In this thesis we study different classes of real-time temporal logics. More precisely, we start from the work of [CHKM09] that considers the verification of CTMCs against Timed Automata specifications and extend it by considering, in addition, non-deterministic TAs that were not analysed in [CHKM09]. Non-deterministic TAs differ from deterministic TAs only in the fact that they allow non-deterministic transitions between states. Moreover, we develop model checking algorithms for CTMCs against MTL formulas and *Linear Duration Properties* (LDPs), i.e., properties involving linear constraints over cumulated residence time in system states, which to our knowledge were not addressed before.

In Chapter 5 we also define a logic that we call *Counting Metric Temporal Logic* (CMTL), which extends MTL with basic counting formulas. Counting formulas are used to count the number of actions (events) in a given interval of time.

2.3 Parameter synthesis

Sometimes the model or the specification under consideration cannot be fully determined, i.e., it could contain parameters for which possible different values are all admissible. When a model contains parameters that can take values in a discrete or continuous domain, a general question that one wants to address is whether some of the parameters' values are better than others, in the sense of optimising some objective function.

Much effort has been devoted to developing efficient techniques for parameter synthesis. We cite here works that are related and share similarities with the parameter synthesis algorithms presented in this dissertation. Most of the work on parameter synthesis that we cite is related to parametric Timed Automata. Parametric TAs are essentially TAs

enriched with parameters on the guards of the transitions.

In [HRSV01], the authors study the decidability problem for parametric Timed Automata. They consider a special case of Timed Automata, L/U automata, for which they show that the emptiness problem is decidable. L/U automata are parametric Timed Automata with the further constraint that the parameters on the guards of the transitions appear either as lower bounds or as upper bounds of clocks, but not both. In [Doy07], undecidability for parametric reachability problem on TAs is proved. The parametric reachability problem asks whether there exists a set of parameters of the system that allows one to reach a given system state.

In [ACFE08, AF10] the authors describe an approach to derive the constraints on parameters of the TAs such that the behaviours of the TAs are time-abstract equivalent, starting from a reference valuation rather than a logic formula. In words, the parameters' values in the time-abstract equivalence class produce the same time behaviour of the system.

The parameter synthesis problem for branching-time logic *parametric timed* CTL (PTCTL) is studied in [BR07], where parameters are given both in the model and the formula. They show the decidability for a fragment of CTL where equality is not allowed. In [KP12], the authors apply the bounded model checking procedure to solve the synthesis problem for the existential fragment of PTCTL without the next operator.

In this thesis we concentrate on parameters in the model and not in the specification. In fact, we perform parameter synthesis for a subclass of parametric Timed Automata. In contrast to all the works cited above, we present in Chapter 5 algorithms for parameter synthesis from specifications given in a generalisation of the linear-time logic MTL, rather than a branching-time logic or a reference valuation.

2.4 Model checking continuous-time Markov chains

The focus of model checking for Continuous-Time Markov Chains (CTMCs) has primarily been on algorithms for specifications expressed in stochastic temporal logics, including *branching-time* variants, such as CSL [BHHK03, ASSB00], as well as *linear-time* temporal logic (LTL). The verification of LTL properties reduces to applying well-known algorithms [Var85, CY95] to embedded discrete-time Markov chains (DTMCs). Like CTL model checking, CSL model checking of finite CTMCs proceeds by a recursive descent over the parse tree of the CSL formula. One of the key ingredients is that time-bounded reachability probabilities can be approximated arbitrarily closely by a reduction to transient analysis in CTMCs (see [BHHK03] for more details). In [BCH+07] the authors define the logic asCSL where path properties are characterised by (time-bounded) regular expressions over actions and state formulas. In order to evaluate path formulas of asCSL one has to consider also state formulas in intermediate states. Thus, asCSL is strictly more expressive than CSL

[BCH+07]. The model checking algorithm for asCSL transforms the regular expression into an automaton, computes the product between the CTMC and this automaton and as the last step calculates time-bounded reachability probabilities in the product obtained. As expressive as asCSL is CSL^{TA} introduced in [DHS09]. There, a timed automaton with a single clock is used to specify time constraints of until modalities. Model checking CSL^{TA} is reduced to computing the reachability probabilities in a DTMC whose transition probabilities are given by subordinate CTMCs.

Linear-time properties equipped with timing constraints have only recently been considered. For example, in [CHKM09, CHKM11, BCH+11] the authors consider the problem of model checking real-time properties given as *Deterministic Timed Automata* (DTAs) over CTMCs. DTAs can express properties of the form "what is the probability to reach a given target state within the deadline, while avoiding unsafe states and not staying too long in any of the dangerous states on the way?". Such properties cannot be expressed in CSL nor in its dialects [BCH+07, DHS09]. Model checking DTA properties can be achieved by a reduction to computing the reachability probabilities in *piecewise-deterministic Markov processes* (PDMPs, [Dav93]). The approach is based on the product construction between the CTMC and the DTA.

For duration calculus (DC), which is based on interval temporal logic that differs from the setting considered in this thesis, the focus has been on so called *linear durational invariants* (LDI, see [CJLX94]). Again, TA (and their subclasses or extensions) are considered, and different techniques are proposed, for instance, reduction to linear programming or CTL, discretisation, etc. We mention the work in [LHZ97, TH04, ZHL08], which are specific to TA and cannot be adapted to CTMCs. There is only scant work addressing probabilistic/stochastic extensions of DC [HZ99, GH10, HZ07]. However, algorithmic verification is not addressed.

Linear Duration Properties are closely related to *Markovian Reward Models* (MRM, see [BHHK00]), which are CTMCs augmented with multiple reward structures assigning real-valued rewards to each state in the model. Properties of MRMs can be expressed in continuous stochastic reward logic (CSRL, see [BHHK00]). CSRL model checking for MRMs [HCH+02, Clo06] involves timed-bounded and/or reward-bounded reachability problems. In this thesis we will establish a link between LDPs over CTMCs and rewards in MRM.

Part of the work in this thesis builds on [CHKM09] with the distinction that we consider properties expressed in MTL or general TAs (see Chapter 4), which allow *nondeterminism*. Approximation algorithms are proposed, based on paths exploration of the CTMC, constraints generation and reduction to volume computation. We are mostly interested in "time-bounded" verification. By time-bounded we refer to the fact that only timed paths over a fixed interval of time are considered, for example, the probability of an alarm bell ringing whenever the button has been pressed for at least 2 seconds continuously. How-

ever, it was shown in [ACH97] that the expressiveness of TA/MTL is limited and cannot express duration-bounded causality properties which constrain the accumulated satisfaction times of state predicates along an execution path, visited possibly intermittently. We overcome this limitation in Chapter 4, where we present an algorithm for verifying CTMCs against LDP, i.e., properties stated as conjunctions of linear constraints over the total duration of time spent in states that satisfy a given property.

2.5 Model checking medical devices

As mentioned in Chapter 1, implantable medical devices, such as cardiac pacemakers, must be designed and programmed to the highest levels of safety and reliability. For these reasons, researchers in recent years have focused their attention on developing formal techniques aimed at deepening our understanding and improving the functioning of such devices. For example, Jiang et al. in [JPM+12b] developed a model-based framework for automatically verifying cardiac pacemakers in the real-time setting. Working from descriptions by Boston Scientific, a leading manufacturer, [JPM+12b] develop a detailed model of a basic pacemaker as a network of Timed Automata (see [AD94] for a detailed presentation on TAs). A network of TAs is essentially a group of TAs enriched with input and output actions which allow communication with other TAs in the network. The authors of [JPM+12b] also provide a model of the human heart as a TA, and perform verification using the model checker UPPAAL [LPY97].

In [JPC+10, JPM12a] the authors develop a real-time Virtual Heart Model (VHM), which can be used for simulation and testing, whereas in [JPM12a] they devise a framework for testing and validation of implantable cardiac devices. Tuan *et al* [TZT10] propose a real-time formal model for a pacemaker and its environment. The authors use the PAT model checker to verify a number of time constraints. The main difference between all these works and the content of this thesis is that probabilities are not considered.

There is an extensive literature that formulates human heart models. In [GJM93, LKM10], the authors develop a model of the cardiac conduction system of the human heart that addresses the stochastic behaviour of the heart, validated via simulation. Grosu *et al.* in [GSC+09] carry out automated formal analysis of a realistic cardiac cell model. The cardiac cell is modelled with a hybrid automaton (see Section 5.2.2). Hybrid automata are TAs enriched with more general variables that can update their values at any rate. Grosu *et al.* in [GBF+11] propose a method to learn and to detect emergent behaviours that may lead to diseases of the heart, such as the one called ventricular fibrillation. The main difference between all these works and the content of this thesis is that the composition with the pacemaker is not considered.

More work on verifying medical devices has come from the theorem proving community [MLF08, GO09, MS09]. Theorem proving is a technique that uses a combination of two

disciplines: mathematics and logics. Implementation and specification are expressed as formulas in predicate logic and then axioms and rules of inference are used to derive properties of the system. Again, probabilities were not considered.

Our work presented in Chapter 5 builds on [GSC+09, GBF+11] and on [JPM+12b]. We, in fact, reuse the cardiac cell model introduced in [GSC+09, GBF+11] and the pacemaker model of [JPM+12b], which we also enhance to analyse energy consumption and pacing noise. We formulate a generic model-based framework for pacemaker software which can be instantiated with human heart models, potentially different from the ones that we have analysed in this thesis, and is amenable to verification.

2.6 Summary

In this chapter we reviewed the work related to the research presented in this thesis. In Section 2.1, we provide a general overview of the research area of probabilistic model checking. Then, in Section 2.2, we discuss real-time logics and Timed Automata, which are the formalisms that we use in this thesis to specify real-time system properties. In Section 2.3 we talk about parameter synthesis. Parameter synthesis is addressed in Chapter 5. Section 2.4 presents a brief summary of the work on continuous-time Markov chains, which is the main focus of Chapter 4. The chapter is concluded with Section 2.5 which describes recent advances in modelling and analysis of medical devices. More specifically, we discuss the importance of formal verification for medical devices and present relevant publications to show the differences with the work in this thesis. Formal verification of medical devices is the main focus of Chapter 5.

Chapter 3

Preliminaries

This chapter presents an overview of the background material needed to understand the rest of the thesis.

In Section 3.1, we introduce the probabilistic models of *discrete-time Markov chains* and *continuous-time Markov chains*. We conclude the chapter with Section 3.2, where we present an overview of relevant *real-time specification notations*, including, *Metric Temporal Logic*, *Linear Duration Properties* and *Timed Automata*.

3.1 Models

In this section we formally introduce the main models that we use in this dissertation. As mentioned in the introduction we are interested in real-time models. Real-time models can be divided into non-probabilistic models and probabilistic models. The difference between probabilistic and non-probabilistic models lies in the fact that in probabilistic models jumps among transitions happen following a probability distribution. We will introduce the probabilistic models of *Discrete-Time Markov Chains* DTMCs (see Section 3.1.1) and *Continuous-Time Markov Chains* CTMCs (see Section 3.1.2). Briefly, DTMCs are labelled transition systems augmented with probability distributions on transitions. On the other hand, CTMCs are DTMCs in which the residence times in system states are exponentially distributed.

3.1.1 Discrete-time Markov chains

Discrete-time Markov chains (DTMCs) [BK08] is one of the simplest probabilistic models. DTMCs are basically labelled transition systems in which successor states are chosen according to a probability distribution. Systems that evolve in a fully defined deterministic or probabilistic fashion can both be modelled by DTMCs. For instance, DTMCs have been used to analyse Herman' self-stabilisation algorithm [KNP12] for ring networks, the Bluetooth device discovery protocol [DKNP06], and many others.

Definition 3.1.1 (DTMC) *A (labelled) discrete-time Markov chain* DTMC *is a tuple* $\mathcal{D} = (S, \mathrm{AP}, \alpha, L, \mathbf{P})$ *where :*

- S *is a finite set of states;*
- AP *is a finite set of atomic propositions;*
- α *is the initial distribution over* S*;*
- $L : S \to 2^{\mathrm{AP}}$ *is the labelling function;*
- $\mathbf{P} : S \times S \to [0, 1]$ *is a stochastic matrix.*

The stochastic matrix $\mathbf{P}$ specifies for each state $s \in S$ the probability $\mathbf{P}(s, s')$ of moving from s to s' in one step, namely in one transition. In order for $\mathbf{P}$ to be a stochastic matrix we impose that $\mathbf{P}(s, s') \geqslant 0$ for every $s, s' \in S$ and for all states s we have that $\sum_{s' \in S} \mathbf{P}(s, s') = 1$. The labelling function assigns atomic propositions to states to label them with properties of interest.

An *infinite* path in $\mathcal{D}$ is an infinite sequence of states

$$\varsigma = s_0 \longrightarrow s_1 \longrightarrow s_2 \cdots \longrightarrow s_n \ldots,$$

for which $\mathbf{P}(s_i, s_{i+1}) > 0$ for all i.

A *finite* path is a finite sequence of states

$$\varsigma = s_0 \longrightarrow \cdots \longrightarrow s_n,$$

for which $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i < n$.

We define $|\varsigma| := n$ to be the length of a finite path ς. We write $\varsigma[0..n]$ for the discrete path ς' of length n obtained from ς. For a finite or infinite path ς, $\varsigma[n] := s_n$ is the $(n+1)$-th state of ς. Moreover, we indicate with ς^i the prefix of length i of ς. We also define $Paths^{\mathcal{D}}$ to be the set of all infinite paths of the DTMC $\mathcal{D}$. For more details about DTMCs refer to [BK08].

A DTMC $\mathcal{D}$ yields a probability measure $\Pr_\alpha^{\mathcal{D}}$ on $Paths^{\mathcal{D}}$ as follows. Let $s_0, \ldots, s_k \in S$ with $\mathbf{P}(s_i, s_{i+1}) > 0$ for $0 \leqslant i < k$. Let $C(s_0, \ldots, s_k)$ denote the *basic cylinder set* consisting of all $\varsigma \in Paths^{\mathcal{D}}$ such that $\varsigma[i] = s_i$ $(0 \leqslant i \leqslant k)$. $\mathcal{F}(Paths^{\mathcal{D}})$ is the smallest σ-algebra on $Paths^{\mathcal{D}}$, which contains all sets $C(s_0 \ldots, s_k)$ for all state sequences $(s_0, \ldots, s_k) \in S^{k+1}$ with $\mathbf{P}(s_i, s_{i+1}) > 0$ for $(0 \leqslant i < k)$. The *probability measure* $\Pr_\alpha^{\mathcal{D}}$ on $\mathcal{F}(Paths^{\mathcal{D}})$ is the unique measure defined by induction on k by $\Pr_\alpha^{\mathcal{D}}(C(s_0)) = \alpha(s_0)$ and for $k > 0$:

$$\Pr_\alpha^{\mathcal{D}}(C(s_0, \ldots, s_k)) = \Pr_\alpha^{\mathcal{D}}(C(s_0, \ldots, s_{k-1})) \cdot \mathbf{P}(s_{k-1}, s_k).$$

Sometimes we write Pr instead of $\Pr_\alpha^{\mathcal{D}}$ when $\mathcal{D}$ and α are clear from the context.

Example 3.1.1 *Consider the* DTMC *in Figure 3.1 and the simple cylinder set* $C(s_0, s_1, s_2)$. *The probability of* C *in* $\mathcal{D}$ *can be calculated as:*

$$\begin{aligned} \Pr^{\mathcal{D}}(C(s_0, s_1, s_2)) &= \mathbf{P}(s_0, s_1) \cdot \mathbf{P}(s_1, s_2) \\ &= 1 \cdot \frac{1}{5} \end{aligned}$$

The next definitions that we introduce, i.e., Definition 3.1.2 and Definition 3.1.3, can be applied to any directed graph and for such a reason are shared between DTMCs and CTMCs (see Section 3.1.2).

Definition 3.1.2 (SCC) *Let* $\mathcal{D} = (S, \mathrm{AP}, \alpha, L, \mathbf{P})$ *be a* DTMC. *A set of states* $S' \subseteq S$ *is a* strongly connected component *(SCC) of* $\mathcal{D}$ *if, for any two states* $s, s' \in S'$, *there exists a path* $\varsigma = s_0 \to s_1 \to \ldots \to s_n$ *such that* $s_i \in S'$ *for* $0 \leqslant i \leqslant n$, $s_0 = s$ *and* $s_n = s'$.

Definition 3.1.3 (BSCC) *An SCC* B *is a* bottom strongly connected component *(BSCC) if no state outside* B *is reachable from any state in* B.

Example 3.1.2 *A simple example of a* DTMC *pacemaker model is pictured in Figure 3.1. The role of the pacemaker is to deliver electric impulses to the human heart in order to induce a heart beat. The pacemaker starts in the state "start". In "start" the pacemaker is ready to deliver a beat via an electric impulse and the system evolves by moving to state "try". Once in "try" the electric impulse can either be delivered with probability* $\frac{4}{5}$ *and the system moves to "delivered" or lost with probability* $\frac{1}{5}$ *moving the system to state "lost". The pacemaker tries to send the electric impulse again if it was lost while proceeding to state "try", or moves back to "start" if the electric impulse was delivered.*

The DTMC $\mathcal{D} = (S, \mathrm{AP}, \alpha, L, \mathbf{P})$ *of the pacemaker described above is shown in Figure 3.1. The set of states is* $S = \{s_0, s_1, s_2, s_3\}$, *the set of atomic propositions is* $\mathrm{AP} = \{start, try, lost, delivered\}$ *and the initial distribution is* $\alpha(s_0) = 1$ *and* $\alpha(s_i) = 0$ *for* $i \in \{1, 2, 3\}$. *The stochastic matrix* $\mathbf{P}$ *is:*

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & \frac{4}{5} \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The labelling function L *is as follows:* $L(s_0) =$ "*start*", $L(s_1) =$ "*try*", $L(s_2) =$ "*lost*" *and* $L(s_3) =$ "*delivered*". *An example of an infinite path corresponding to the system trying unsuccessfully to deliver an electric impulse is* $\varsigma = s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow s_1 \ldots$ *whereas an example of a finite path representing the system that delivers the electric*

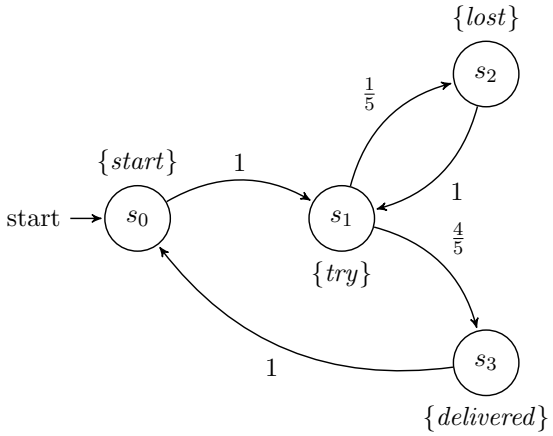


Figure 3.1: An example of DTMC

impulse is $\varsigma = s_0 \longrightarrow s_1 \longrightarrow s_3 \longrightarrow s_0$. *In the* DTMC *of Figure 3.1 the whole state space, namely* $S = \{s_0, s_1, s_2, s_3\}$, *is a strongly connected component, as well as* $S' = \{s_1, s_2\}$ *and* $S'' = \{s_0, s_1, s_3\}$. *The* DTMC *has no bottom strongly connected components apart from its own state space* S.

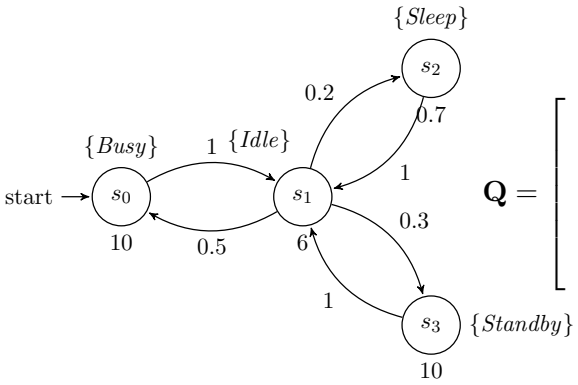
3.1.2 Continuos-time Markov chains

The second model that we consider is that of *Continuous-Time Markov Chains* (CTMCs). CTMCs allow the modelling of real-time passage in conjunction with stochastic evolution governed by exponential distributions. They can be thought of as state transition systems, in which the system resides in a state on average for $1/r$ time units, where r is the exit rate, and transitions between the states are determined by a discrete probability distribution.

Definition 3.1.4 (CTMC) *A* (*labelled*) *continuous-time Markov chain* (CTMC) *is a tuple* $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$ *where :*

- S *is a finite set of states;*
- AP *is a finite set of atomic propositions;*
- $L : S \to 2^{\mathrm{AP}}$ *is the labelling function;*
- α *is the initial distribution over* S*;*
- $\mathbf{P} : S \times S \to [0, 1]$ *is a stochastic matrix; and*
- $E : S \to \mathbb{R}_{>0}$ *is the exit rate function.*

In a CTMC $\mathcal{C}$, state residence times are *exponentially* distributed. More precisely, the residence time of the state $s \in S$ is a random variable governed by an exponential



$$\mathbf{Q} = \begin{bmatrix} -10 & 10 & 0 & 0 \\ 3 & -6 & 1.2 & 1.8 \\ 0 & 0.7 & -0.7 & 0 \\ 0 & 5 & 0 & -5 \end{bmatrix}$$

Figure 3.2: An example CTMC and its associated infinitesimal generator matrix

distribution with parameter $E(s)$. Hence, the probability to exit state s in t time units is given by $\int_0^t E(s) \cdot e^{-E(s)\tau} d\tau$; and the probability to take the transition from s to s' in t time units equals $\mathbf{P}(s, s') \cdot \int_0^t E(s) \cdot e^{-E(s)\tau} d\tau$. A state s is *absorbing* if $\mathbf{P}(s, s) = 1$.

Example 3.1.3 *As a concrete example of a system and property studied, consider the dynamic power management system (DPMS) from [QQM01], analysed in [NPK+05] against properties such as average power consumption. The DPMS includes a queue of requests, which have an exponentially distributed inter-arrival time, a power management controller and a service provider. The power management controller issues commands to the service provider depending on the power management policy, which involves switching between different power-saving modes. Figure 3.2 depicts a* CTMC *model of the service provider for a Fujitsu disk drive. The service provider model in Figure 3.2 is composed of four states: Busy, Idle, Sleep and Standby. The system makes a transition from Idle to Busy whenever a request arrives for service. Similarly, it makes a transition from Busy to Idle whenever it finishes the service of a request. Transitions between Sleep, Standby and Idle are controlled by the power management. The power management switches the service provider on, i.e., the service provider moves to Idle, when the service request queue is full and the service provider is in Sleep or Standby.*

Formally, the CTMC *in Figure 3.2 can be given as* $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$ *where: the set of states is* $S = \{s_0, s_1, s_2, s_3\}$*; the set of atomic propositions is* $\mathrm{AP} = \{$*Busy, Idle, Sleep, Standby*$\}$*; the labelling function* L *is* $L(s_0) =$ *"Busy"*, $L(s_1) =$ *"Idle"*, $L(s_2) =$ *"Sleep"*, $L(s_3) =$ *"Standby"*; *and the initial distribution is* $\alpha(s_0) = 1$ *(in this case, a Dirac distribution). The exit rates are indicated at the states, whereas the transition probabilities are attached to the transitions, characterising* E *and* $\mathbf{P}$ *respectively.*

Example 3.1.4 *Consider the* CTMC *in Figure 3.2 and suppose that the system is in state* s_1 *at a given time. The probability of leaving* s_1 *in* 5 *time units is equal to:* $\int_0^5 6 \cdot e^{-6\tau} d\tau = 1 - e^{-30}$ *(which is almost* 1*). Similarly, the probability to take the transition from* s_1 *to* s_3 *in* 5 *time units, given that the system is in* s_1*, is equal to* $0.3 \cdot (1 - e^{-30})$.

We also define the *infinitesimal generator* $\mathbf{Q}$ of $\mathcal{C}$ as

$$\mathbf{Q} = \mathbf{E} \cdot \mathbf{P} - \mathbf{E},$$

where $\mathbf{E}$ is the diagonal matrix with exit rates on the diagonal. Figure 3.2 shows the infinitesimal generator matrix for the CTMC of Example 3.1.3. Occasionally we use $X(t)$ to denote the underlying *stochastic process* of $\mathcal{C}$.

We write $\pi(t)$ for the *transient probability distribution*, where, for each $s \in S$,

$$\pi_s(t) = \Pr\left(\{X(t) = s\}\right)$$

is the probability to be in state s at time t. It is well known that $\pi(t)$ completely depends on the initial distribution α and the infinitesimal generator $\mathbf{Q}$ [BHHK03], i.e., it is the solution of the Chapman-Kolmogorov equation

$$\frac{d\pi(t)}{dt} = \pi(t)\mathbf{Q}, \qquad \pi(0) = \alpha \ .$$

Similarly to DTMCs, an *infinite timed* path in $\mathcal{C}$ is an infinite sequence of states and residence times

$$\rho = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \cdots \xrightarrow{t_{n-1}} s_n \ldots,$$

for which $\mathbf{P}(s_i, s_{i+1}) > 0$ for all i.

A *finite timed* path is a finite sequence of states and residence times

$$\sigma = s_0 \xrightarrow{t_0} s_1 \cdots \xrightarrow{t_{n-1}} s_n,$$

for which $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i < n$.

Intuitively, a timed path ρ suggests that the CTMC $\mathcal{C}$ starts in state s_0 and stays in this state for t_0 time units, and then jumps to state s_1, staying there for t_1 time units, and then jumps to s_2, and so on.

In the rest of the thesis we follow the convention to let ρ (resp. σ) range over infinite (resp. finite) timed paths, unless otherwise stated. We define $|\sigma| := n$ to be the length of a finite timed path σ. In both cases we assume that $t_i \in \mathbb{R}_{>0}$ for each $i \geqslant 0$; moreover, we write $\rho[0..n]$ for the discrete timed path σ of length n obtained from ρ. For a finite or infinite path θ, $\theta[n] := s_n$ is the $(n+1)$-th state of θ and $\theta\langle n \rangle := t_n$ is the time spent in state s_n, and let $\theta@t$ denote the state occupied in θ at time $t \in \mathbb{R}_{\geqslant 0}$, i.e., $\theta@t := \theta[n]$, where n is the smallest index such that $\sum_{i=0}^{n} \theta\langle i \rangle > t$. Moreover, we indicate with θ^i the prefix of length i of θ.

An example timed path is $\rho = s_0 \xrightarrow{3} s_1 \xrightarrow{2} s_0 \xrightarrow{1.5} s_1 \xrightarrow{3.4} s_2 \ldots$ with $\rho[2] = s_0$ and $\rho@4 = \rho[1] = s_1$.

We say that the DTMC $\mathcal{D}$, denoted by

$$\mathcal{D} = (S, \mathrm{AP}, \alpha, L, \mathbf{P}),$$

defined according to Definition 3.1.1 is the *embedded* DTMC of the CTMC $\mathcal{C}$. Given a finite discrete path $\varsigma = s_0 \to \cdots \to s_n$ of length n and $x_0, \ldots, x_{n-1} \in \mathbb{R}_{>0}$, we define $\varsigma[x_0, \ldots, x_{n-1}]$ to be the finite *timed* path σ such that $\sigma[i] := s_i$ and $\sigma\langle i \rangle := x_i$ for each $0 \leqslant i < n$. Let $\Gamma \subseteq \mathbb{R}^n_{>0}$, then

$$\varsigma[\Gamma] = \{\varsigma[x_0, \ldots, x_{n-1}] \mid (x_0, \ldots, x_{n-1}) \in \Gamma\}.$$

Given a finite (resp. infinite) discrete path ς and a finite (resp. infinite) timed path ρ, we say ς is the *skeleton* of ρ if, for each $i \geqslant 0$, $\varsigma[i] = \rho[i]$. We write $\mathbb{S}(\rho)$ for the skeleton of ρ, and, for a set of (finite or infinite) timed paths Ξ, we write $\mathbb{S}(\Xi) = \{\mathbb{S}(\rho) \mid \rho \in \Xi\}$. Moreover, given a *finite* discrete path ς, we define $C_d(\varsigma) = \{\varsigma\varsigma' \mid \varsigma' \text{ is an infinite discrete path}\}$ to be the set of all infinite discrete paths with the same common prefix ς. Let $Paths^{\mathcal{C}}$ denote the set of infinite timed paths in $\mathcal{C}$, with abbreviation $Paths$ when $\mathcal{C}$ is clear from the context.

The definition of a *Borel space* on timed paths of CTMCs follows [BHHK03]. A CTMC $\mathcal{C}$ yields a probability measure $\Pr^{\mathcal{C}}_{\alpha}$ on $Paths^{\mathcal{C}}$ as follows. Let $s_0, \ldots, s_k \in S$ with $\mathbf{P}(s_i, s_{i+1}) > 0$ for $0 \leqslant i < k$ and $I_0, \ldots, I_{k-1}$ be nonempty intervals in $\mathbb{R}_{\geqslant 0}$. Let $C(s_0, I_0, \ldots, I_{k-1}, s_k)$ denote the *basic cylinder set* consisting of all $\rho \in Paths$ such that $\rho[i] = s_i$ $(0 \leqslant i \leqslant k)$ and $\rho\langle i \rangle \in I_i$ $(0 \leqslant i < k)$. $\mathcal{F}(Paths)$ is the smallest σ-algebra on $Paths$, which contains all sets $C(s_0, I_0, \ldots, I_{k-1}, s_k)$ for all state sequences $(s_0, \ldots, s_k) \in S^{k+1}$ with $\mathbf{P}(s_i, s_{i+1}) > 0$ for $(0 \leqslant i < k)$ and $I_0, \ldots, I_{k-1}$ ranging over all sequences of nonempty intervals in $\mathbb{R}_{>0}$. The *probability measure* $\Pr^{\mathcal{C}}_{\alpha}$ on $\mathcal{F}(Paths)$ is the unique measure defined by induction on k by $\Pr^{\mathcal{C}}_{\alpha}(C(s_0)) = \alpha(s_0)$ and for $k > 0$:

$$\Pr^{\mathcal{C}}_{\alpha}(C(s_0, I_0, \ldots, I_{k-1}, s_k)) = \Pr^{\mathcal{C}}_{\alpha}(C(s_0, I_0, \ldots, I_{k-2}, s_{k-1})) \cdot \int_{I_{k-1}} \mathbf{P}(s_{k-1}, s_k) E(s_{k-1}) \cdot e^{-E(s_{k-1})\tau} d\tau.$$

Sometimes we write Pr instead of $\Pr^{\mathcal{C}}_{\alpha}$ when $\mathcal{C}$ and α are clear from the context. As the reader can see, the probability measure of CTMCs extends the probability measure of DTMCs with integrations over the residence times in system states.

Example 3.1.5 *Consider, the* CTMC $\mathcal{C}$ *in Figure 3.2 and the simple cylinder set* $C(s_0, I_0, s_1, I_1, s_2)$ *where* $I_0 = [0, 3]$ *and* $I_1 = [0, 5]$. *The probability of* C *in* $\mathcal{C}$ *can be calculated as:*

$$\begin{aligned} Pr^{\mathcal{C}}(C(s_0, I_0, s_1, I_1, s_2)) &= \mathbf{P}(s_0, s_1)\mathbf{P}(s_1, s_2)E(s_0)E(s_1) \\ &\quad \times \int_{I_0} \int_{I_1} e^{-E(s_0)\tau_1 - E(s_1)\tau_2} d\tau_2 d\tau_1 \\ &= 1 \cdot 0.2 \cdot 10 \cdot 6 \\ &\quad \times \int_0^3 \int_0^5 e^{-10\tau_1 - 6\tau_2} d\tau_2 d\tau_1 \end{aligned}$$

Elements of the σ-algebra denote *events* in the probability space. We now define two such events that will be needed later in Section 4.3.

Definition 3.1.5 *Given a CTMC $\mathcal{C}$ and $B \subseteq S$, we define:*

- $\Diamond^{\leqslant T} B = \left\{ \rho \in Paths^C \mid \exists n. \rho[n] \in B \text{ and } \sum_{i=0}^{n-1} \rho\langle i \rangle \leqslant T \right\}$, *i.e., $\Diamond^{\leqslant T} B$ denotes the set of (infinite) timed paths which reach B in time interval $[0, T]$. Note that $\Pr^{\mathcal{C}}(\Diamond^{\leqslant T} B)$ can be computed by a reduction to the computation of the transient probability distribution; see [BHHK03].*

- $\Diamond B = \{\rho \in Paths^C \mid \exists n. \rho[n] \in B\}$, *i.e., $\Diamond B$ denotes the set of (infinite) timed paths which reach B. (This is the unbounded variant of $\Diamond^{\leqslant T} B$.) Note that $\Pr^{\mathcal{C}}(\Diamond B)$ is essentially the reachability probability of B in the embedded DTMC of $\mathcal{C}$; see [BHHK03].*

Example 3.1.6 *Consider the* CTMC *in Figure 3.2. The set $\Diamond^{\leqslant 5} s_2$ denotes the set of all the timed paths that reach the "Sleep" state in no more than 5 time units. On the other hand, the set $\Diamond s_3$ denotes the set of all the timed paths which eventually go into "Standby" mode.*

In general, the probability of reaching a given set of states can be computed as a solution of a system of linear equalities [BK08]. The procedure, which is shared between DTMCs and CTMCs, is the following. Let $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$ be a CTMC and $B \subseteq S$ a set of goal states. Let the variable x_s denote the probability of reaching B from an arbitrary $s \in S$. The goal is to compute $x_s = \Pr(s \models \Diamond B)$ for all the states $s \in S$. It should be clear that if B is not reachable from s then $x_s = 0$. Similarly, if $s \in B$ then $x_s = 1$. A simple graph analysis suffices to identify the states that fall in the above mentioned two categories, namely the ones that either have $x_s = 0$ or $x_s = 1$. For all the other states we can write:

$$x_s = \sum_{s' \in S \setminus B} \mathbf{P}(s, s') \cdot x_{s'} + \sum_{s'' \in B} \mathbf{P}(s, s'') \tag{3.1}$$

Intuitively, Equation 3.1 states that either the set B is reached in one step, i.e., from s we jump to s'' which belongs to B, or in more than one step, i.e., from s we jump to s' and then we reach B from s'.

We show how to compute the reachability probabilities with Example 3.1.7.

Example 3.1.7 *Let $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$ be the* CTMC *in Figure 3.2. Let $B = \{s_3\}$ and suppose we want to find the probability of the system being eventually in "Standby" mode, i.e., we are looking to compute $\Pr^{\mathcal{C}}(\Diamond B)$. In this case we obtain the following system*

of linear equalities:

$$\begin{aligned} x_{s_0} &= x_{s_1} \\ x_{s_1} &= \frac{1}{2}x_{s_0} + \frac{1}{5}x_{s_2} + \frac{3}{10}x_{s_3} \\ x_{s_2} &= x_{s_1} \\ x_{s_3} &= 1 \end{aligned}$$

Solving the system of linear equalities one finds that the probability of eventually reaching s_3 from any other state in $\mathcal{C}$ is equal to 1. *In fact, the results are confirmed by the fact that the* CTMC *in Figure 3.2 is a BSCC, which in turn yields that all its states will be visited infinitely often with probability* 1.

3.2 Real-time specifications

As described in the introduction, a vast range of real systems meet the task they have been designed to accomplish only if they relate properly with the passage of time. Standard temporal logic, such as the *linear time temporal logic* (LTL) described in Section 3.2.1, is inadequate for the study of real-time systems, whose correctness depends crucially on the actual time at which events occur. More powerful real-time formalisms must generalise the temporal logic methodology to encompass the analysis of real-time behaviour. For such a reason, in this thesis, we use three distinct formalisms which can be used to specify real-time properties: *Metric Temporal Logic* (MTL) described in Section 3.2.2, the *Linear Duration Properties* (LDPs) introduced in Section 3.2.3, and Timed Automata (TAs) discussed in Section 3.2.4.

3.2.1 Linear temporal logic

The Linear Temporal Logic LTL was first introduced in [Pnu77]. LTL is capable of expressing linear-time properties that system executions should satisfy. With LTL, users can specify the property that, for example, "eventually in the future, a pacemaker beat will happen". LTL is not a real-time temporal logic in the sense that it does not allow users to specify the actual time at which events occur. In fact, it is not possible to say with LTL that "every pacemaker beat is followed by a natural heart beat within 1 second unless a new pacemaker beat happens before".

We recall now here the syntax and semantics of LTL [SC85]:

Definition 3.2.1 (Syntax of LTL) *Let* AP *be an arbitrary non-empty, finite set of atomic propositions. The logic* LTL *can be inductively defined as:*

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U} \varphi_2$$

where $p \in$ AP and φ_1, φ_2 are LTL *formulas.*

In Definition 3.2.2 we define a variant of acceptance condition for an infinite discrete word and an LTL formula φ. The reason why we introduce a bounded semantics for LTL, and not the standard unbounded semantics, is because in Chapter 4 we will relate LTL formulas to MTL formulas (see Section 3.2.2), for which we define a bounded semantics as well.

In the reminder of this section we assume a CTMC $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$. However, the concepts can be easily generalised to any transition system model as long as we define a labelling function L for the states of the model.

Definition 3.2.2 (Bounded Semantics of LTL) *Given an* LTL *formula* φ*, a* finite *discrete path* ς *and* $i \in \mathbb{N}$*, the satisfaction relation* $(\varsigma, i) \models \varphi$ *is inductively defined as follows:*

$$
\begin{array}{lcl}
(\varsigma, i) \models p & \Leftrightarrow & p \in L(\varsigma_i) \text{ and } i \leqslant |\varsigma| \\
(\varsigma, i) \models \neg\varphi_1 & \Leftrightarrow & (\varsigma, i) \not\models \varphi_1 \\
(\varsigma, i) \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow & (\varsigma, i) \models \varphi_1 \wedge (\varsigma, i) \models \varphi_2 \\
(\varsigma, i) \models \varphi_1 \mathcal{U} \varphi_2 & \Leftrightarrow & \exists i'.\ i \leqslant i' \leqslant |\varsigma| \text{ s.t. } (\varsigma, i') \models \varphi_2 \wedge \\
 & & \forall i''.\ i \leqslant i'' < i' \Rightarrow (\varsigma, i'') \models \varphi_1
\end{array}
$$

where $p \in \mathrm{AP}$*,* φ_1*,* φ_2 *are* LTL *formulas and* $i', i'' \in \mathbb{N}$*. For an* infinite *discrete path* ς*, we define* $\varsigma \models \varphi$ *if there exists some* $k \geqslant 0$ *such that the finite discrete path* $(\varsigma^k, 0) \models \varphi$*.*

Definition 3.2.3 (Positive normal form) *We say that an* LTL *(*MTL*) formula* φ *is in normal form if negations appear only in front of atomic propositions. Any* LTL *(*MTL*) formula* φ *can be transformed into an equivalent* LTL *(*MTL*) formula in positive normal form. We refer the reader to [BK08] for a detailed algorithm which transforms any* LTL *formula into an equivalent* LTL *formula in positive normal form.*

Example 3.2.1 *Consider two sample discrete paths,* $\varsigma_1 = s_0 \longrightarrow s_1$*,* $\varsigma_2 = s_1 \longrightarrow s_2$*, and the* LTL *formula* $\varphi = a\mathcal{U}b$*. Moreover, consider that the atomic proposition "*a*" belongs to the labelling of state* s_0*, i.e.,* $a \in L(s_0)$*, but not to* s_1 *and* s_2*, i.e.,* $a \notin L(s_1)$ *and* $a \notin L(s_2)$*, and that the atomic proposition "*b*" belongs to the labelling of* s_1*,* $b \in L(s_1)$*, but* $b \notin L(s_0)$ *and* $b \notin L(s_2)$*. Following the semantics introduced in Definition 3.2.2 we obtain that* $\varsigma_1 \models \varphi$*, but* $\varsigma_2 \not\models \varphi$*. Since* φ *does not contain any negation,* φ *is in positive normal form.*

3.2.2 Metric temporal logic

The Metric Temporal Logic (MTL) was first introduced in [Koy90]. MTL is a linear-time temporal logic which allows to specify the exact time at which events occur. An example of MTL property, as mentioned in Section 3.2.1, is the following: "every pacemaker beat

is followed by a natural heart beat within 1 second unless a new pacemaker beat happens before".

We start by defining the syntax of MTL.

Definition 3.2.4 (Syntax of MTL) *Let* AP *be an arbitrary, non-empty, finite set of atomic propositions. Let* $I = [a, b]$ *be an interval such that* $a, b \in \mathbb{N} \cup \{\infty\}$. *The* Metric Temporal Logic *[AH93, Koy90] is inductively defined as:*

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}^I \varphi_2$$

where $p \in$ AP *and* φ_1, φ_2 *are* MTL *formulas.*

We introduce as usual the $\Diamond, \Box$ operators defined as follows: $\Diamond^I \varphi = \neg\mathbf{true} U^I \varphi$ and $\Box^I = \neg\Diamond^I \neg\varphi$, where φ is a MTL formula. The $\Diamond$ operator is usually referred to as the "eventuality" operator since it expresses the possibility of something happening in the future. The $\Box$ operator is usually referred to as the "always" operator since it expresses the possibility of an event that continuously happens in the future.

In the reminder of this section we assume a CTMC $\mathcal{C} = (S, \text{AP}, L, \alpha, \mathbf{P}, E)$. However, the concepts can be easily generalised to any transition system model as long as we define a labelling function L for the states of the model.

We introduce two time-bounded semantics for MTL, as follows.

Definition 3.2.5 (Continuous Semantics) *Given an* MTL *formula* φ, *a time bound* T, *a timed path* ρ *and a variable* $t \in \mathbb{R}_{\geq 0}$, *the satisfaction relation* $(\rho, t) \models^c_T \varphi$ *is inductively defined as follows:*

$$\begin{aligned}
(\rho,t) \models^c_T p \quad &\Leftrightarrow \quad p \in L(\rho@t) \wedge t \leqslant T \\
(\rho,t) \models^c_T \neg\varphi_1 \quad &\Leftrightarrow \quad (\rho,t) \not\models^c_T \varphi_1 \\
(\rho,t) \models^c_T \varphi_1 \wedge \varphi_2 \quad &\Leftrightarrow \quad (\rho,t) \models^c_T \varphi_1 \wedge (\rho,t) \models^c_T \varphi_2 \\
(\rho,t) \models^c_T \varphi_1 \mathcal{U}^I \varphi_2 \quad &\Leftrightarrow \quad \exists t'.\ t \leqslant t' \leqslant T \ \textit{s.t.}\ t' - t \in I \wedge (\rho,t') \models^c_T \varphi_2 \wedge \\
&\qquad\quad \forall t''.\ t \leqslant t'' < t' \Rightarrow (\rho,t'') \models^c_T \varphi_1
\end{aligned}$$

where $p \in$ AP *and* φ_1, φ_2 *are* MTL *formulas.*

Definition 3.2.6 (Pointwise Semantics) *Given an* MTL *formula* φ, *a time bound* T, *a timed path* ρ *and* $i \in \mathbb{N}$, *the satisfaction relation* $(\rho, i) \models^p_T \varphi$ *is inductively defined as follows:*

$$\begin{aligned}
(\rho,i) \models^p_T p \quad &\Leftrightarrow \quad p \in L(\rho[i]) \wedge \sum_{k=0}^{i} \rho\langle k\rangle \leqslant T \\
(\rho,i) \models^p_T \neg\varphi_1 \quad &\Leftrightarrow \quad (\rho,i) \not\models^p_T \varphi_1 \\
(\rho,i) \models^p_T \varphi_1 \wedge \varphi_2 \quad &\Leftrightarrow \quad (\rho,i) \models^p_T \varphi_1 \wedge (\rho,i) \models^p_T \varphi_2 \\
(\rho,i) \models^p_T \varphi_1 \mathcal{U}^I \varphi_2 \quad &\Leftrightarrow \quad \exists i'.\ i \leqslant i' \ \textit{s.t.} \sum_{k=i}^{i'} \rho\langle k\rangle \in I \wedge (\rho,i') \models^p_T \varphi_2 \wedge \\
&\qquad\quad \forall i''.\ i \leqslant i'' < i' \Rightarrow (\rho,i'') \models^p_T \varphi_1
\end{aligned}$$

where $p \in \mathrm{AP}$, φ_1, φ_2 *are* MTL *formulas and* $i', i'' \in \mathbb{N}$.

Example 3.2.2 *Consider two sample timed paths,* $\rho_1 = s_0 \xrightarrow{2.5} s_1$, $\rho_2 = s_0 \xrightarrow{4.5} s_1$, *and the* MTL *formula* $\varphi = a\mathcal{U}^{[2,3]}b$. *Moreover, consider that the atomic proposition "*a*" belongs to the labelling of state* s_0, *i.e.,* $a \in L(s_0)$, *but not to* s_1, *i.e.,* $a \notin L(s_1)$, *and that the atomic proposition "*b*" belongs to the labelling of* s_1, $b \in L(s_1)$, *but* $b \notin L(s_0)$. *Following the two semantics introduced in Definition 3.2.5 and Definition 3.2.6, we can obtain that* $\rho_1 \models \varphi$, *but* $\rho_2 \not\models \varphi$.

3.2.3 Linear duration properties

Although LTL and MTL are powerful logics that allow users to define real-time properties, as pointed out in [ACH97] their expressiveness is limited and cannot express duration-bounded causality properties which constrain the accumulated satisfaction times of state predicates along an execution path, visited possibly intermittently. An example of such a property is that "the accumulated time spent in unsafe states must be less than or equal to one tenth of the accumulated time spent in safe states". *Linear Duration Properties* (LDPs) first introduced in [CDKM12b, CDKM13b] overcome this limitation.

We first introduce a language which includes the propositional calculus augmented with the *duration function* $\int$ and linear inequalities. In the reminder of this section we assume a CTMC $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$. However, the concepts can be easily generalised to any model as long as we define a labelling function L for the states of the model.

State formulas are defined inductively as

$$\mathrm{sf} ::= ap \mid \neg \mathrm{sf} \mid \mathrm{sf}_1 \wedge \mathrm{sf}_2,$$

where $ap \in \mathrm{AP}$. Given a state formula sf and a state $s \in S$ we say that s satisfies the state formula sf, denoted $s \models \mathrm{sf}$, iff

$$\begin{aligned} s \models ap \quad &\Leftrightarrow \quad ap \in L(s) \\ s \models \neg \mathrm{sf} \quad &\Leftrightarrow \quad s \not\models \mathrm{sf} \\ s \models \mathrm{sf}_1 \wedge \mathrm{sf}_2 \quad &\Leftrightarrow \quad s \models \mathrm{sf}_1 \text{ and } s \models \mathrm{sf}_2 \end{aligned}$$

The *duration function* $\int$ is interpreted over a *finite* timed path. Let sf be a state formula and $\sigma = s_0 \xrightarrow{t_0} \ldots \xrightarrow{t_{n-1}} s_n$.

The value of $\int \mathrm{sf}$ for σ, denoted $[\![\mathrm{sf}]\!]_\sigma$, is defined as $\sum_{\substack{0 \leqslant i < n, \\ \sigma[i] \models \mathrm{sf}}} t_i$. That is, the value of $\int \mathrm{sf}$ equals the sum of durations spent in states satisfying sf.

A *linear duration property* (LDP) is of the form:

$$\varphi = \bigwedge_{j \in J} \left(\sum_{k \in K_j} c_{jk} \int \mathrm{sf}_{jk} \leqslant M_j \right), \tag{3.2}$$

where $c_{jk}, M_j \in \mathbb{R}$, sf_{jk} are state formulas, and J, K_j for $j \in J$ are finite index sets.

Definition 3.2.7 *Given a finite timed path* $\sigma = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{n-1}} s_n$ *and an LDP* φ *of the form defined in Equation* (3.2)*, we write* $\sigma \models \varphi$ *if, for each* $j \in J$*,*

$$\sum_{k \in K_j} c_{jk} \cdot [\![\mathrm{sf}_{jk}]\!]_\sigma \leqslant M_j.$$

Example 3.2.3 *For the* CTMC *in Figure 3.2, the LDP* $\varphi = \int \mathit{Idle} - \frac{1}{3} \int \mathit{Busy} \leqslant 0$ *expresses the constraint that during the evolution of the* CTMC *the accumulated time spent in the "Idle" state must be less than or equal to one third of the accumulated time spent in the "Busy" state.*

Inspired by the notation of [CJLX94], we shall also work wit a slight extension of LDP, i.e., formulas of the form:

$$\Phi := \int 1 \leqslant T \to \varphi,$$

where 1 denotes a label belonging to any state, $\to$ denotes "implication", $T \in \mathbb{R}_{\geqslant 0} \cup \{\infty\}$, $\int 1$ denotes the total time spent on a finite timed path σ. Note that, $\int 1 \leqslant T \to \varphi$ is a single formula. Hence $\sigma \models \Phi$ if φ holds whenever the total time of σ is less or equal than T. The LDP formula Φ degenerates in φ if $T = \infty$.

Definition 3.2.8 *Let* $\rho = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \ldots$ *be an infinite timed path and* φ *(or* Φ*) be an LDP. We introduce the following two satisfaction conditions:*

- Finitary satisfaction condition. *Given a set of* goal *states* $G \subseteq S$*, we write* $\rho \models^G \varphi$ *if there exists some* $i \in \mathbb{N}$ *such that:*

  (1) $\rho[i] \in G$ *and for any* $0 \leqslant j < i$, $\rho[j] \notin G$*; and*

  (2) $\rho[0..i] \models \varphi$ *(see Definition 3.2.7).*

  Furthermore, we write $\rho \models^G_T \varphi$ *for a given* $T \in \mathbb{R}_{\geqslant 0}$ *if, in addition to (1) and (2),* $\sum_{j=0}^{i-1} \rho\langle j \rangle \leqslant T$ *holds.*

- Infinitary satisfaction condition. *We write* $\rho \models^\star \varphi$ *if, for any* $n \geqslant 0$, $\rho[0..n] \models \varphi$ *(cf. Definition 3.2.7).*

Intuitively the *finitary satisfaction condition* represents the fact that the LDP formula φ is satisfied in the timed path ρ. The *infinitary satisfaction condition* is much stronger. A path ρ satisfies the LDP φ if, at any instant of time, φ is valid in ρ. We illustrate the meaning of the two definitions with an example.

Example 3.2.4 *Consider the* CTMC *in Figure 3.2, the LDP* $\varphi = \int \mathit{Idle} - \frac{1}{3} \int \mathit{Busy} \leqslant 0$ *and the finite timed path* $\sigma = s_0 \xrightarrow{2} s_1 \xrightarrow{0.3} s_0 \xrightarrow{3} s_1 \xrightarrow{0.4} s_2$*. Moreover, consider that the*

set of goal states is $G = \{s_2\}$ *and that* $T = 10$. *We have* $\rho \models_T^G \varphi$ *but* $\rho \not\models^\star \varphi$ *since there is an index* $n \geqslant 0$ *where* $\rho[0..n] \not\models \varphi$, *namely,* $n = 1$.

3.2.4 Timed automata

Timed Automata (TAs) were first introduced in [AD94]. TAs are essentially labelled transition systems augmented with clocks, i.e., real-time variables that track the passage of time, and guards on transitions which constraints the time at which jumps between system states can happen.

TA specifications are useful for two reasons:

1. Some of the properties that can be expressed with TAs cannot be expressed in MTL (or LTL and LDP). See Example 4.2.1 in Section 4.2 for a concrete example of such a property.

2. TAs allow one to perform product constructions between the model and the specification, which is often a useful technique to enable model checking.

Before describing TAs in detail we introduce some definitions and notations.

Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of *nonnegative* real-valued variables, called *clocks*. An $\mathcal{X}$-valuation is a function $\eta : \mathcal{X} \to \mathbb{R}_{\geqslant 0}$ assigning to each variable x a nonnegative real value $\eta(x)$.

A *clock constraint* on $\mathcal{X}$, denoted by g, is a conjunction of expressions of the form $x \bowtie y$ for clock $x \in \mathcal{X}$, comparison operator $\bowtie \in \{<, \leqslant, >, \geqslant\}$ and $y \in \{\mathbb{N}\}$. We write $x \in g$, for $x \in \mathcal{X}$, if the guard g contains a constraint on clock x and $g.x := (\bowtie, y)$ with $g.x(1) = \bowtie$ and $g.x(2) = y$ if $x \bowtie y$ is a constraint of g.

Let $\mathcal{B}(\mathcal{X})$ denote the set of clock constraints over $\mathcal{X}$. An $(\mathcal{X})$-valuation η *satisfies* a constraint $x \bowtie y$, denoted $\eta \models x \bowtie y$, if and only if $\eta(x) \bowtie y$ and $y \in \mathbb{N}$; it satisfies a conjunction of such expressions if and only if η satisfies all of them.

Let $\mathbf{0}$ denote the valuation that assigns 0 to all clocks. For a subset $X \subseteq \mathcal{X}$, the reset of X, denoted $\eta[X := 0]$, is the valuation η' such that $\forall x \in X.\ \eta'(x) := 0$ and $\forall x \notin X.\ \eta'(x) := \eta(x)$. For $\delta \in \mathbb{R}_{>0}$ and $\mathcal{X}$-valuation η, $\eta{+}\delta$ is the $\mathcal{X}$-valuation η'' such that $\forall x \in \mathcal{X}.\ \eta''(x) := \eta(x){+}\delta$, meaning that all clocks proceed at the same speed.

Given a set $\mathcal{H}$, let Pr: $\mathcal{F}(\mathcal{H}) \to [0, 1]$ be a *probability measure* on the measurable space $(\mathcal{H}, \mathcal{F}(\mathcal{H}))$, where $\mathcal{F}(\mathcal{H})$ is a σ-algebra over $\mathcal{H}$. Let $Distr(\mathcal{H})$ denote the set of probability measures on this measurable space.

Formally, a TA can be defined as follows.

Definition 3.2.9 (Timed Automaton) *A* timed automaton *[AD94] is a tuple* $\mathcal{A} = (\Sigma, \mathcal{X}, Q, q_0, Q_F, \to)$ *where:*

- Σ *is a finite* alphabet*;*

- $\mathcal{X}$ *is a finite set of* clocks*;*
- Q *is a non-empty finite set of* modes *with initial mode* $q_0 \in Q$*;*
- $Q_{\boldsymbol{F}}$ *is a set of* final *modes; and*
- *the relation* $\rightarrow \subseteq Q \times \Sigma \times \mathcal{B}(\mathcal{X}) \times 2^{\mathcal{X}} \times Q$ *is an* edge relation*.*

We refer to $q \xrightarrow{a,g,X} q'$ as an *edge*, where $a \in \Sigma$ is an input symbol, the *guard* g is a clock constraint on the clocks of $\mathcal{A}$, X is the set of clocks that must be reset and q' is the successor mode. Intuitively, the edge $q \xrightarrow{a,g,X} q'$ asserts that the TA $\mathcal{A}$ can move from mode q to mode q' when the input symbol is a and the guard g holds, while the clocks in X should be reset when entering q'. In case no guard is satisfied in a mode for a given clock valuation, time can progress. For the sake of simplicity we omit invariants from the definition of TAs.

Definition 3.2.10 *Given a timed automaton* $\mathcal{A}$*, we define the following notions.*

- *A* discrete *path of* $\mathcal{A}$ *is a sequence of states* $\varsigma = q_0 \rightarrow q_1 \ldots \rightarrow q_n \ldots$ *where each* $q_i \in Q$ *and the edge between* q_i *and* q_{i+1} *exists for all* i*.*
- *A* timed *path of* $\mathcal{A}$ *is of the form* $\theta = q_0 \xrightarrow{a_0,t_0} q_1 \xrightarrow{a_1,t_1} \ldots q_{n-1} \xrightarrow{a_{n-1},t_{n-1}} q_n \ldots$ *such that* η_i *is the clock evaluation when entering* q_i *and* $\eta_0 = \mathbf{0}$*. For all* $i \geqslant 0$ *with* $a_i \in \Sigma$ *it holds that* $t_i > 0$*,* $\eta_i + t_i \models g_i$ *where* g_i *is the guard on the* i*-th transition. For all* $i \geqslant 0$ *we have that the clock evaluation of step* $i+1$ *is updated according to the following rule:* $\eta_{i+1} = (\eta_i + t_i)[X_i := 0]$*. We say that* θ is accepting *if there exists some* $n \geqslant 0$ *such that* $q_n \in Q_{\mathbf{F}}$*. As for* CTMC*s we write* ρ *(respectively* σ*), instead of* θ*, for infinite (respectively finite) timed paths.*

Definition 3.2.11 (Time-bounded Acceptance) *Assume a* TA $\mathcal{A} = (\Sigma, \mathcal{X}, Q, q_0, Q_{\mathbf{F}}, \rightarrow)$ *and a time bound* $T \in \mathbb{R}_{\geqslant 0}$*. A timed path* $\theta = q_0 \xrightarrow{a_0,t_0} q_1 \xrightarrow{a_1,t_1} \ldots$ *is accepted by* $\mathcal{A}$ *according to the time-bounded acceptance condition if there* exists $i \in \mathbb{N}_{>0}$ *such that* $q_i \in Q_{\boldsymbol{F}}$ *and* $\sum_{i=0}^{n-1} t_i \leqslant T$*. We write* $\theta \models_T \mathcal{A}$ *to denote that the timed path* θ *is accepted by* $\mathcal{A}$ *within the time bound* T*.*

Example 3.2.5 *As an example consider the TA* $\mathcal{A} = (\Sigma, \mathcal{X}, Q, q_0, Q_{\mathbf{F}}, \rightarrow)$ *in Figure 3.3 where:* $\Sigma = \{a, b\}$*,* $\mathcal{X} = \{x\}$*,* $Q = \{q_0, q_1\}$*,* $Q_{\mathbf{F}} = \{q_1\}$*. Intuitively,* $\mathcal{A}$ *expresses the property that the first "a" is seen within* 5 *time units. Moreover, consider two finite timed paths* $\sigma_1 = q_0 \xrightarrow{b,0.3} q_0 \xrightarrow{a,2} q_1 \xrightarrow{a,7} q_1$*,* $\sigma_2 = q_0 \xrightarrow{b,7.5} q_0 \xrightarrow{a,3.9} q_1 \xrightarrow{a,2.8} q_1$ *and the time bound* $T = 3$*. According to Definition 3.2.11 we have that* $\sigma_1 \models_T \mathcal{A}$*, whereas* $\sigma_2 \not\models_T \mathcal{A}$*.*

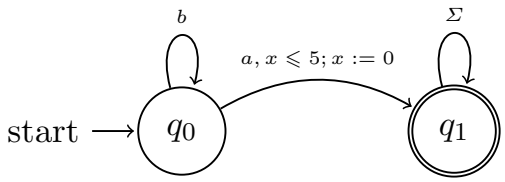


Figure 3.3: An example TA

3.3 Summary

In this chapter we have introduced the background material that is needed to understand the content of this thesis. We have overviewed the probabilistic models of discrete and continuous-time Markov chains and their applications in Section 3.1. The chapter concludes with Section 3.2, where we have presented the formalisms of *Metric Temporal Logic*, *Linear Duration Properties* and *Timed Automata* that are used in this thesis to specify real-time properties.

Chapter 4

Model checking real-time properties over continuous-time Markov chains

In recent years researchers have focused their attention on developing model checking algorithms for CTMCs against a variety of different temporal logics (see Section 2.4 of Chapter 2 for details). However, the verification of CTMCs against real-time specifications such as MTL, LDP or TAs has not been addressed before. In many real-life examples it is not possible to ignore the real-time constraints that a system must obey in order to function correctly. Think, for instance, to the software embedded in medical devices, such as pacemakers. Pacemakers must work correctly 24 hours per day, seven days per week. We show the importance of real-time properties with Example 4.0.1.

Example 4.0.1 *We recall the dynamic power management system (DPMS)* CTMC *of Figure 3.2 presented in Example 3.1.3 and refer the reader to Section 3.1.2 for a detailed description of the system. From a performance perspective it would be important to check that the system deals with its requests in a timely manner. One might wish to verify that with high probability it is always the case that, once a request has been accepted (the system is in Busy), it is served within* 60 *seconds (the system goes back to Idle). This property, known also as "bounded response", can be expressed in* MTL *as:* $\varphi = \Box(Busy \Rightarrow \Diamond^{[0,60]} Idle)$. *Similarly, a good performance indicator would be to ensure that during an entire day the system spends most of its time being busy serving requests. More specifically, one might want to check that, with high probability, the system spends at least* $\frac{3}{4}$ *of time in Busy during the whole day of* 24 *hours. This property can be specified as LDP* $\Phi = \int 1 \leqslant 24 \rightarrow \varphi$, *where* $\varphi = \int Busy \geqslant \frac{3}{4} \int (Busy + Idle + Sleep + Standby)$, *or with some simplifications as* $\varphi = \int Busy \geqslant 3 \int (Idle + Sleep + Standby)$.

In light of Example 4.0.1, in this chapter we study the time-bounded verification of a finite continuous-time Markov chain (CTMC) $\mathcal{C}$ against a real-time specification. The real-time specification can be provided as a:

- metric temporal logic (MTL) property φ;
- timed automaton (TA) $\mathcal{A}$; or
- linear duration property (LDP) φ.

The key question that we address is the following.

Model checking problem

Input: A CTMC $\mathcal{C}$ and a real-time property φ (or $\mathcal{A}$)

Problem: Find the probability of the set of timed paths of $\mathcal{C}$ that satisfy φ (or are accepted by $\mathcal{A}$) over a time interval of fixed, bounded length.

The model checking algorithms for the real-time specifications mentioned above share several similarities with each other. In particular, we provide approximation algorithms to approximate the solution of these problems. We first derive a bound N such that timed paths of $\mathcal{C}$ with at most N discrete jumps are sufficient to approximate the desired probability up to ε. Then, for each discrete path ς of length at most N, we generate timed constraints over variables determining the residence time of each state along ς, depending on the real-time specification under consideration. The probability of the set of timed paths, determined by the discrete path and the associated timed constraints, can thus be formulated as a multidimensional integral. Summing up all such probabilities yields the result.

The content of this chapter is based on two conference papers [CDKM11, CDKM12b] and one journal publication [CDKM13b].

The chapter is divided into three sections. Section 4.1 tackles the model checking problem for specifications given as MTL formulas. In Section 4.2 we present algorithms to address the model checking problem when specifications are given as Timed Automata. The algorithms for MTL and TAs in Section 4.1 and Section 4.2 are very similar. We conclude the chapter with Section 4.3, where we solve the model checking problem for real-time specifications given as LDPs.

4.1 Verifying continous-time Markov chains against MTL

In this section we study the problem of model checking CTMCs against MTL properties. The content of this section is based on a published conference paper [CDKM11] by the

author of this thesis and his colleagues at Oxford.

The problem, as mentioned in the introduction of this chapter, can be summarised as follows.

Model checking problem

Input: A CTMC $\mathcal{C}$ and a MTL property φ

Problem: Find the probability of the set of timed paths of $\mathcal{C}$ that satisfy φ over a time interval of fixed, bounded length.

We start off with some definitions that we will use throughout the section. First, recall that the syntax and semantics of the real-time temporal logic MTL was defined in Section 3.2.2. Let $\Pr^{\mathcal{C}}_T(\varphi) := \Pr^{\mathcal{C}}(\{\rho \in \mathit{Paths}^{\mathcal{C}}_T \mid (\rho, 0) \models^{\mathsf{c}}_T \varphi\})$ denote the probability that the CTMC $\mathcal{C}$ satisfies the MTL formula φ, for a given time bound T. In words, $\Pr^{\mathcal{C}}_T(\varphi)$ is the probability of the set of timed paths ρ of $\mathcal{C}$ which satisfy φ. Note that here the definition of $\Pr^{\mathcal{C}}_T(\varphi)$ is for the continuous semantics of MTL, but we present an algorithm to deal also with MTL in pointwise semantics later. Instead of computing $\Pr^{\mathcal{C}}_T(\varphi)$, we give a procedure to compute $\Pr^{\mathcal{C}}_{T,<N}(\varphi) := \Pr^{\mathcal{C}}(\mathit{Paths}^{\mathcal{C}}_{T,<N}(\varphi))$, where $\mathit{Paths}^{\mathcal{C}}_{T,<N}(\varphi)$ is the set of timed paths of length less than N which satisfy the MTL formula φ, for a given time bound T. The number N should be chosen sufficiently large in order to ensure that $\Pr^{\mathcal{C}}_T(\varphi) - \Pr^{\mathcal{C}}_{T,<N}(\varphi) < \varepsilon$ for arbitrarily small $\varepsilon \in \mathbb{R}_{>0}$. This yields an approximation algorithm. The measurability of the set of $\mathit{Paths}^{\mathcal{C}}_{T,<N}(\varphi) := \{\rho \in \mathit{Paths}^{\mathcal{C}}_{T,<N} \mid (\rho, 0) \models^{\mathsf{c}}_T \varphi\}$ can be shown as in [SK11]. In this section we present a detailed algorithm to compute $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$. The algorithm can be decomposed into five steps:

Step 1. Bound the number of jumps N in the interval of time $[0, T]$ in order to get the desired error ε;

Step 2. Transform the MTL formula φ to the untimed linear-time temporal logic formula (LTL, see Section 3.2.1) form $\widetilde{\varphi}$ to reduce the complexity of the model checking algorithm;

Step 3. Construct the non-deterministic finite automaton $\mathcal{A}_{\widetilde{\varphi}}$ out of $\widetilde{\varphi}$;

Step 4. Build the product $\mathcal{C} \times \mathcal{A}_{\widetilde{\varphi}}$;

Step 5. Search for all the discrete paths ς in $\mathcal{C} \times \mathcal{A}_{\widetilde{\varphi}}$ of length at most N and, for each of those, generate the set of linear inequalities $\mathcal{S}$ and calculate the probability of ς under the constraints in $\mathcal{S}$.

In the rest of the Section 4.1 we will discuss each of the five steps above separately.

Step 1: A bound on the number of discrete jumps

We give a bound, N, on the number of discrete jumps that occurs between $[0, T]$, where T is a time bound. The intuition here is that, for a given time interval $[0, T]$, the probability of the set of timed paths which "jump" very frequently is actually very small. Thus, in our algorithm, we do not need to consider paths that jump very often. Throughout this section we assume a CTMC $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$.

Remark 4.1.1 *The procedure of finding a bound on the number of discrete jumps on the* CTMC *is shared with Section 4.2. The reason is that a bound on the number of jumps that occur in* $[0, T]$ *is equivalent to restricting the number of timed paths of the* CTMC *that we need to consider and it is independent from the chosen property specification formalism.*

Our goal here is to find the integer N, also referred as a step bound, such that the set of timed paths which jump more than N times in $[0, T]$ has a very small probability. We introduce now some lemmas which will help finding the right step bound N.

We start with Lemma 4.1.1, which gives an integral form to the probability of all the paths that have more than N jumps in $[0, T]$. Theorem 4.1.3 then demonstrates how to bound analytically the integral form previously given in Lemma 4.1.1. At this point we have proved that the probability of all the timed path that have more than N jumps between $[0, T]$ is always smaller than a given upper bound. Finally, Proposition 4.1.4 shows how to choose a natural number N such that the probability of all the timed paths that have more than N jumps in $[0, T]$ is smaller than a given ε, where ε is the desired error bound.

For any $n \in \mathbb{N}$, we define $V^n(s, x) : S \times \mathbb{R}_{\geqslant 0} \to [0, 1]$ as follows: $V^0(s, x)$=1 and

$$V^{n+1}(s, x) = \int_0^x E(s) e^{-E(s)\tau} \cdot \sum_{s' \in S} \mathbf{P}(s, s') \cdot V^n(s', x - \tau) d\tau \ .$$

Lemma 4.1.1 *For all* $N \in \mathbb{N}$, $\Pr^{\mathcal{C}}(Paths^{\mathcal{C}}_{T, \geqslant N}(s)) = V^N(s, T)$.

Proof By induction on N.

1. $N = 0$: $V^0(s, T) = 1$. This is exactly the probability of all paths $\{\rho \in Paths^{\mathcal{C}}(s) \mid \rho\langle 0 \rangle \leqslant T\}$, that is, $\Pr^{\mathcal{C}}(Paths^{\mathcal{C}}_{T, \geqslant 0}(s))$. $\Pr^{\mathcal{C}}(Paths^{\mathcal{C}}_{T, \geqslant 0}(s))$ is the probability to have 0 jumps in the interval of time $[0, T]$, which is equal to $e^{-E(s)T}$, plus the probability of having more than one jump in $[0, T]$, that is, $1 - e^{-E(s)T}$. Summing up the two probabilities yields 1 as the result.

2. Induction step.

$$\begin{aligned} V^{N+1}(s,T) &= \int_0^T E(s)e^{-E(s)\tau} \cdot \sum_{s' \in S} \mathbf{P}(s,s')V^N(s,T-\tau)d\tau \\ &= \int_0^T E(s)e^{-E(s)\tau} \cdot \sum_{s' \in S} \mathbf{P}(s,s')\Pr^{\mathcal{C}}(Paths^{\mathcal{C}}_{T,\geqslant N}(s))d\tau \\ &= \Pr^{\mathcal{C}}(Paths^{\mathcal{C}}_{T,\geqslant N+1}(s)) \quad \blacksquare \end{aligned}$$

Next, in Lemma 4.1.2, we show how to bound $V^N(s,T)$ analytically. Given a CTMC $\mathcal{C}$, let $\Lambda = \max_{s \in S} E(s)$ and $\epsilon(T,N) = e^{-\Lambda T} \cdot \left(\sum_{i=N}^{\infty} \frac{(\Lambda T)^i}{i!} \right)$.

Lemma 4.1.2

$$\epsilon(T, N+1) = \int_0^T \Lambda e^{-\Lambda\tau} \cdot \epsilon(T-\tau, N)d\tau \ .$$

Proof

$$\begin{aligned} &\int_0^T \Lambda e^{-\Lambda\tau} \cdot \epsilon(T-\tau,N)d\tau \\ &= \int_0^T \Lambda e^{-\Lambda\tau} \cdot e^{-\Lambda(T-\tau)} \cdot (\sum_{i \geqslant N} \frac{(\Lambda(T-\tau))^i}{i!})d\tau \\ &= e^{-\Lambda T} \int_0^T \Lambda \cdot (\sum_{i \geqslant N} \frac{(\Lambda(T-\tau))^i}{i!})d\tau \\ &= e^{-\Lambda T} \sum_{i \geqslant N} \int_0^T \Lambda \cdot (\frac{(\Lambda(T-\tau))^i}{i!})d\tau \\ &= e^{-\Lambda T} \sum_{i \geqslant N} -\frac{(\Lambda(T-\tau))^{i+1}}{(i+1)!}|_0^T \\ &= e^{-\Lambda T} \sum_{i \geqslant N} \frac{(\Lambda T)^{i+1}}{(i+1)!} \\ &= e^{-\Lambda T} \sum_{i \geqslant N+1} \frac{(\Lambda T)^i}{(i)!} \\ &= \epsilon(T, N+1). \quad \blacksquare \end{aligned}$$

We are now ready for the main result of this section, Theorem 4.1.3. Theorem 4.1.3 gives an error bound to the probability of the all timed paths which jump more than N times in $[0,T]$. The bound is obtained combining Lemma 4.1.1 and Lemma 4.1.2 together. More specifically, we have the following

Theorem 4.1.3 *Given a* CTMC $\mathcal{C}$*, a time bound* T *and* $N \in \mathbb{N}$*, we have that*

$$\Pr^{\mathcal{C}}(Paths^{\mathcal{C}}_{T,\geqslant N}) \leqslant \epsilon(T,N).$$

Proof By induction on N. The base case (i.e., $N = 0$) is straightforward. For $N + 1$ we have

$$\begin{aligned}
\Pr^{\mathcal{C}}(Paths^{\mathcal{C}}_{T,\geqslant N+1}) &= \int_0^T E(s)e^{-E(s)\tau} \cdot \sum_{s' \in S} \mathbf{P}(s,s') \cdot \Pr^{\mathcal{C}}(Paths^{\mathcal{C}}_{T-\tau,\geqslant N}(s'))d\tau \\
&\leqslant \int_0^T E(s)e^{-E(s)\tau} \cdot \sum_{s' \in S} \mathbf{P}(s,s') \cdot \epsilon(T-\tau, N)d\tau \\
&\leqslant \int_0^T \Lambda e^{-\Lambda\tau} \cdot \sum_{s' \in S} \mathbf{P}(s,s') \cdot \epsilon(T-\tau, N)d\tau
\end{aligned}$$

From Lemma 4.1.2 we have that

$$\begin{aligned}
&\int_0^T \Lambda e^{-\Lambda\tau} \cdot \sum_{s' \in S} \mathbf{P}(s,s') \cdot \epsilon(T-\tau, N)d\tau \\
&= \sum_{s' \in S} \mathbf{P}(s,s') \cdot \int_0^T \Lambda e^{-\Lambda\tau} \cdot \epsilon(T-\tau, N)d\tau \\
&= \sum_{s' \in S} \mathbf{P}(s,s') \cdot \epsilon(T, N+1) \\
&= \epsilon(T, N+1).
\end{aligned}$$

It follows that $\Pr^{\mathcal{C}}(Paths^{\mathcal{C}}_{T,\geqslant N+1}) \leqslant \epsilon(T, N+1)$, which completes the induction step. ■

Proposition 4.1.4 shows how to pick the right step bound N given a CTMC $\mathcal{C}$, a time bound T and a maximum error tolerance ε.

Proposition 4.1.4 *Let $\varepsilon \in \mathbb{R}_{>0}$ and $T \in \mathbb{R}_{\geqslant 0}$. For any $N \geqslant \Lambda T e^2 + \ln(\frac{1}{\varepsilon})$ we have that $\epsilon(T, N) < \varepsilon$.*

Proof We have that

$$\begin{aligned}
\epsilon(T, N) &= e^{-\Lambda T} \cdot \left(\sum_{i=N}^{\infty} \frac{(\Lambda T)^i}{i!} \right) \\
&= e^{-\Lambda T} \cdot e^{\Lambda T} \cdot \frac{(\Lambda T)^N}{N!} \\
&\leqslant \frac{(\Lambda T)^N}{(N/e)^N} = \left(\frac{\Lambda T e}{N} \right)^N \\
&\leqslant \left(\frac{1}{e} \right)^{\ln(1/\varepsilon)} = \varepsilon
\end{aligned}$$

■

.

Remark 4.1.2 *Readers who are familiar with Poisson distributions will immediately notice that the bound we obtained is actually the probability that there are at least N Poisson arrivals in an interval of time $[0, T]$, with rate Λ. If the CTMC $\mathcal{C}$ is uniform (i.e., each*

state of $\mathcal{C}$ has the same exit rate), then one could obtain the bound in a straightforward way. However, for the general case, this cannot be achieved directly. Moreover, we point out here that, in order to verify an MTL *formula φ or a TA $\mathcal{A}$, one* cannot *apply the uniformisation technique, which is ubiquitous in* CTMC *model checking. The reason for that is that uniformisation is used for reachability properties.* MTL *properties are more expressive than simple reachability.*

Step 2: Transform the MTL formula φ to the untimed linear-time temporal logic formula (LTL, see Section 3.2.1) form $\widetilde{\varphi}$

The basic idea of this step is to exclude those CTMC timed paths which definitely fail φ. We define an LTL formula $\widetilde{\varphi}$ such that, if a discrete path of $\mathcal{C}$ fails $\widetilde{\varphi}$, then any timed path with the discrete path as skeleton must fail φ. This is formally stated in Lemma 4.1.5. Notice that, since we consider the time-bounded semantics of MTL, we need a variant of acceptance for an infinite discrete word and an LTL formula $\widetilde{\varphi}$, which is given in Definition 3.2.2. We then construct a non-deterministic finite automaton (NFA) out of $\widetilde{\varphi}$, such that only those finite discrete CTMC paths which are accepted by the NFA are the prefixes of the potential skeletons of timed paths satisfying φ. Then we apply the standard product construction, which suffices to identify those finite discrete paths analysed in the next step.

Remark 4.1.3 *We remark again here that the following is an optimisation step and it can actually be skipped if one is not concerned with the complexity of the verification algorithm.*

Now we show how it is possible to transform an MTL formula φ to its untimed LTL version $\widetilde{\varphi}$. Given any MTL φ in positive normal form (see Definition 3.2.3), we define an (untimed) LTL formula $\widetilde{\varphi}$ as follows:

$$\begin{array}{lcl}
\varphi = p & \Rightarrow & \widetilde{\varphi} = p \\
\varphi = \neg p & \Rightarrow & \widetilde{\varphi} = \neg p \\
\varphi = \varphi_1 \vee \varphi_2 & \Rightarrow & \widetilde{\varphi} = \widetilde{\varphi}_1 \vee \widetilde{\varphi}_2 \\
\varphi = \varphi_1 \wedge \varphi_2 & \Rightarrow & \widetilde{\varphi} = \widetilde{\varphi}_1 \wedge \widetilde{\varphi}_2 \\
\varphi = \varphi_1 \mathcal{U}^I \varphi_2 & \Rightarrow & \widetilde{\varphi} = \widetilde{\varphi}_1 \mathcal{U} \widetilde{\varphi}_2 \\
\varphi = \Box^I \varphi_1 & \Rightarrow & \widetilde{\varphi} = \text{TRUE}\ \mathcal{U} \widetilde{\varphi}_1
\end{array}$$

where φ_1 and φ_2 are MTL formulas and $\widetilde{\varphi}_1$ and $\widetilde{\varphi}_2$ are LTL formulas.

Before reading further, we remind the reader that the transformation is a purely practical step used for optimisation purposes. Transforming φ to $\widetilde{\varphi}$ allows us to eliminate those CTMC paths for which we are 100% sure that the original MTL formula φ is not satisfied. Once we eliminate the paths that fail $\widetilde{\varphi}$, we check the timed formula φ on the remaining paths, see Algorithm 1.

Example 4.1.1 *Consider, for example, the* CTMC *in Figure 3.2 and the* MTL *formula* $\varphi = \mathit{Idle}\ \mathcal{U}^{[0,5]}\ \mathit{Busy}$. *The formula represents the set of all timed paths that stay between* 0 *and* 5 *time units in "Idle" before moving to "Busy", without going first to Sleep or Standby. The* MTL *formula* φ *is transformed into the equivalent untimed* LTL $\widetilde{\varphi} = \mathit{Idle}\ \mathcal{U}\ \mathit{Busy}$. *It should be clear that, if a path never satisfies the condition that the system stays in Idle continuously before moving to Busy, then the same path cannot satisfy the same condition constrained to the case of that happening between* 0 *and* 5 *time units.*

In the transformation from the MTL formula φ to LTL formula $\widetilde{\varphi}$ we only define the $\neg$ operator for atomic propositions because φ is already in positive normal form. Notice that we transform $\Box^{[a,b]}\varphi$ into $\text{TRUE}\ \mathcal{U}\widetilde{\varphi}$ instead of a seemingly more natural $\Box\widetilde{\varphi}$, because otherwise in the next step we would not consider timed paths ρ such that $(\rho, 0) \models \varphi$ while $\mathbb{S}(\rho) \not\models \widetilde{\varphi}$. Recall here that $\mathbb{S}(\rho)$ is the skeleton of the timed path ρ (see Section 3.1.2 for more details). Such paths do exist. For instance, consider the MTL formula $\Box^{[0,2]}p$ and the path $\rho = s_0 \xrightarrow{2.5} s_1 \cdots$ with $L(s_0) = \{p\}$ and $L(s_1) = \{\neg p\}$. Then $(\rho, 0) \models^{\mathsf{c}}_T \Box^{[0,2]}p$ and $\mathbb{S}(\rho) \not\models \Box p$ (but $\mathbb{S}(\rho) \models \text{TRUE}\ \mathcal{U}p$ as we defined). To conclude, one cannot transform $\Box^{[a,b]}$ by simply removing the time constraints $[a, b]$.

The next lemma that we introduce, Lemma 4.1.5, proves that our transformation from an MTL formula φ to its untimed version $\widetilde{\varphi}$ is correct. In words, Lemma 4.1.5 says that if the MTL formula φ is satisfied in a timed path ρ then it must be the case that the same timed path ρ satisfies $\widetilde{\varphi}$, where $\widetilde{\varphi}$ is the untimed version of φ.

Lemma 4.1.5 *Let* φ *be an* MTL *formula and* ρ *be a timed path in* $\mathcal{C}$. *We have that*

$$(\rho, t) \models^{\mathsf{c}}_T \varphi \Rightarrow (\mathbb{S}(\rho), x) \models \widetilde{\varphi}$$

where $\rho@t = \varsigma[x]$.

Proof For the rest of the proof, we assume $t, t', t'' \in \mathbb{R}$ and $x, i, j \in \mathbb{N}$. The proof of the lemma for the base case ($\varphi = p = \widetilde{\varphi}$) and the cases with the untimed modalities $\varphi = \Phi = \widetilde{\varphi}$, where $(\Phi = \neg p) \vee (\Phi = \varphi_1 \wedge \varphi_2)$, are trivial. The only interesting cases are:

$$(\rho, t) \models^{\mathsf{c}}_T \varphi_1 \mathcal{U}^I \varphi_2 \quad \Rightarrow \quad (\mathbb{S}(\rho), x) \models \widetilde{\varphi}_1 \mathcal{U} \widetilde{\varphi}_2 \tag{4.1}$$

$$(\rho, t) \models^{\mathsf{c}}_T \Box^I \varphi_1 \quad \Rightarrow \quad (\mathbb{S}(\rho), x) \models \text{true}\ \mathcal{U}\widetilde{\varphi}_1 \tag{4.2}$$

1. By the semantics of MTL :

$$(\rho, t) \models^{\mathsf{c}}_T \varphi_1 \mathcal{U}^I \varphi_2 \Rightarrow \begin{cases} \exists t' > t \wedge t' \leqslant T \quad \text{such that.} \\ \quad t' - t \in I \ \wedge\ (\rho, t') \models^{\mathsf{c}}_T \varphi_2 \wedge \\ \quad \forall t''.\ t \leqslant t'' < t'.\ (\rho, t'') \models^{\mathsf{c}}_T \varphi_1 \end{cases} \tag{4.3}$$

Let $\rho@t = \varsigma[x]$ and $\rho@t' = \varsigma[i]$. By induction hypothesis on the second and third row of Equation 4.3:

$$(\mathbb{S}(\rho), x) \models \widetilde{\varphi}_1 \mathcal{U} \widetilde{\varphi}_2 \Leftarrow \begin{cases} \exists i \geqslant x \quad \text{such that.} \\ \quad (\mathbb{S}(\rho), i) \models \widetilde{\varphi}_2 \wedge \\ \quad \forall j.\ 0 \leqslant j < i.\ (\mathbb{S}(\rho), j) \models \widetilde{\varphi}_1 \end{cases}$$

which concludes the proof for Equation 4.1.

2. By the semantics of MTL :

$$(\rho, t) \models^{\mathsf{c}}_{T} \Box^{I} \varphi_1 \Rightarrow \begin{cases} \forall t' \geqslant t \wedge t' \leqslant T. \\ \quad t' - t \in I \ \wedge \ (\rho, t') \models^{\mathsf{c}}_{T} \varphi_1 \end{cases} \tag{4.4}$$

Let $\rho@t = \varsigma[x]$.By induction hypothesis on the first row of Equation 4.4:

$$(\mathbb{S}(\rho), x) \models \text{true}\ \mathcal{U} \widetilde{\varphi}_1 \Leftarrow \begin{cases} \exists i \geqslant x \quad \text{such that.} \\ \quad (\mathbb{S}(\rho), i) \models \widetilde{\varphi}_1 \wedge \\ \quad \forall j.\ 0 \leqslant j < i.\ (\mathbb{S}(\rho), j) \models \ \text{true} \end{cases}$$

which concludes the proof for Equation 4.2. ■

Step 3: Construct the non-deterministic finite automaton $\mathcal{A}_{\widetilde{\varphi}}$ out of $\widetilde{\varphi}$
As the next step, we construct an NFA $\mathcal{A}_{\widetilde{\varphi}}$ which accepts all the prefixes of infinite paths satisfying the formula $\widetilde{\varphi}$ according to Definition 3.2.2.

We propose here two possible ways to build $\mathcal{A}_{\widetilde{\varphi}}$:

1. Use the construction in [GH01]. The authors give a modification of the Vardi-Wolper algorithm for finite-trace (or bounded in other words) semantics of LTL.

2. Use the construction in [KV01]. In [KV01] there is an automata construction for detecting bad prefixes of LTL formulas. Starting from $\widetilde{\varphi}$, we build $\widehat{\varphi} = \neg\widetilde{\varphi}$. Then we construct an automaton to detect the bad prefixes of $\widehat{\varphi}$ following [KV01]. The bad prefixes of $\widehat{\varphi}$ coincide with the good prefixes of $\widetilde{\varphi}$.

Step 4: Build the product $\mathcal{C} \times \mathcal{A}_{\widetilde{\varphi}}$;
We then build the product of $\mathcal{C}$ and $\mathcal{A}_{\widetilde{\varphi}}$.

Definition 4.1.1 (Product $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$) *Given a* CTMC $\mathcal{C} = (S, \mathrm{AP}, L, s_0, \mathbf{P}, E)$ *and an* NFA $\mathcal{A}_{\widetilde{\varphi}} = (Q, 2^{\mathrm{AP}}, \delta, q_0, F)$*, we define the product* $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$ *to be the tuple* $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}} = (Loc, l_0, Loc_{\boldsymbol{F}}, \rightsquigarrow)$ *where:*

- $Loc = S \times Q$;

- $l_0 = \langle s_0, q_0 \rangle$;
- $Loc_{\mathbf{F}} = S \times F$;
- $\rightsquigarrow \subseteq Loc \times Loc$ *such that*

$$\frac{\mathbf{P}(s, s') > 0 \wedge q \xrightarrow{L(s)} q'}{\langle s, q \rangle \rightsquigarrow \langle s', q' \rangle}.$$

A path of the product $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$ is a sequence of locations $l_0 \to \ldots \to l_n$ for which $(l_i, l_{i+1}) \in \rightsquigarrow$. The set of accepted paths in $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$ is defined by $\Diamond Loc_{\mathbf{F}}$. Notice that we are only interested in the discrete paths of $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$. Therefore, we do not assign probabilities to the transition relation $\rightsquigarrow$ when computing the product. The product is used to check which discrete paths in the CTMC verify the formula $\widetilde{\varphi}$.

Recall that, given a CTMC $\mathcal{C}$, we write $C_d(\varsigma)$ for the set of all infinite discrete paths of $\mathcal{C}$ with the same common prefix ς (see Section 3.1.2).

The next proposition that we introduce, i.e., Proposition 4.1.6, proves that the following operations are correct:

- Transform an MTL formula φ into the untimed equivalent LTL formula $\widetilde{\varphi}$;
- Construct the NFA $\mathcal{A}_{\widetilde{\varphi}}$;
- Build the product $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$;
- Compute the set of accepted timed paths of $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$ defined as $\Diamond Loc_{\mathbf{F}}$;
- Project back the set of accepted timed paths of the product onto CTMC timed paths $Loc_{\mathbf{F}} \downarrow_1$, where $Loc_{\mathbf{F}} \downarrow_1$ is the projection of $Loc_{\mathbf{F}}$ onto its first component.

In words, Proposition 4.1.6 states that, given a CTMC $\mathcal{C}$ and an MTL property φ, the following statement must be true. The skeleton of all the timed paths of $\mathcal{C}$ that satisfy φ, i.e., $\mathbb{S}(Paths^{\mathcal{C}}_T(\varphi))$ is a subset of the set of all the infinite discrete paths which have ς as a prefix, i.e., $C_d(\varsigma)$, where ς are all the accepted discrete paths of $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$, i.e., $\Diamond Loc_{\mathbf{F}}$, projected onto the first component of the product.

Proposition 4.1.6 *For any* CTMC $\mathcal{C}$ *and* NFA $\mathcal{A}_{\widetilde{\varphi}}$, $\mathbb{S}(Paths^{\mathcal{C}}_T(\varphi)) \subseteq \{C_d(\varsigma) \mid \varsigma \in \Diamond Loc_{\mathbf{F}} \downarrow_1\}$.

Proof Let $\rho = s_0 \xrightarrow{x_0} s_1 \xrightarrow{x_1} s_2 \xrightarrow{x_2} \ldots$ be a timed path in $Paths^{\mathcal{C}}_T(\varphi)$. Note that, for each $i \geqslant 0$, $\mathbf{P}(s_i, s_{i+1}) > 0$. From Lemma 4.1.5 we have that $(\mathbb{S}(\rho), 0) \models \widetilde{\varphi}$.

$\mathcal{A}_{\widetilde{\varphi}}$ accepts all the prefixes of infinite words that satisfy $\widetilde{\varphi}$. Let w be the word generated by ρ, that is, $w = L(\rho[0])L(\rho[1])L(\rho[2])\ldots$. Obeserve that w is an accepting word for $\mathcal{A}_{\widetilde{\varphi}}$ and it produces an accepting run θ on $\mathcal{A}_{\widetilde{\varphi}}$ such that:

$$\theta = q_0 \xrightarrow{L(s_0)} q_1 \xrightarrow{L(s_1)} q_2 \xrightarrow{L(s_2)} \ldots \xrightarrow{L(s_{n-1})} q_n \ldots$$

where $q_n \in F$. Combining θ with the condition that in ρ, for each $i \geqslant 0$, $\mathbf{P}(s_i, s_{i+1}) > 0$, we can construct a path ν in $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$ such that:

$$\nu = \langle s_0, q_0 \rangle \longrightarrow \langle s_1, q_1 \rangle \longrightarrow \langle s_2, q_2 \rangle \longrightarrow \ldots \longrightarrow \langle s_n, q_n \rangle \ldots$$

where again $q_n \in F$, and consequently $\langle s_n, q_n \rangle \in Loc_{\mathbf{F}}$. This is an accepting path for $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$, which means that $\nu \in \Diamond Loc_{\mathbf{F}}$. It must be the case then, for any $\rho \in Paths^{\mathcal{C}}_T(\varphi)$, that $\mathbb{S}(\rho) \subseteq \{C_d(\varsigma) \mid \varsigma \in \Diamond Loc_{\mathbf{F}} \restriction_1\}$, where $\varsigma = \nu \restriction_1$ for the first n terms of ν, which concludes the proof. ■

Step 5: Compute all the discrete paths of $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$ of length at most N and calculate probabilities.

We divide this step into further four substeps.

Step 5.1. Search the graph $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$ to get all the discrete accepting paths ς of $\mathcal{C}$ of length at most N;

Step 5.2. Generate constraints. Run Algorithm 1 on each discrete path ς of length $n \leqslant N$ to obtain the system of linear inequalities $\mathcal{S}$;

Step 5.3. Compute the probability of $\varsigma[\mathcal{S}]$;

Step 5.4. Sum up all the probabilities for each discrete path to obtain $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$.

Step 5.1: Search for discrete paths of length at most N

The step can be accomplished by standard graph search techniques, such as *depth-first* or *breadth-first*.

Step 5.2: Constraints generation

The set of linear constraints is generated by means of Algorithm 1. Algorithm 1 takes as input a discrete path ς of length n and an MTL formula φ which in Algorithm 1 is evaluated for the continuous semantics. We will show later in Algorithm 2 how to generate the same set of constraints when the formula φ is evaluated considering the pointwise semantics. Algorithm 1 returns a family of linear constraints $\mathcal{S} = \bigvee_{i \in I} \bigwedge_{j \in J_i} c_{ij}$ where c_{ij} is a linear inequality over the set of variables $t_0, \ldots, t_{n-1}$. Intuitively, the variables $t_0, \ldots, t_{n-1}$ will be associated with state residence times in the CTMC path under consideration. Given a system of linear constraints $\mathcal{S}$ we define the set of feasible solutions to be the tuples $(x_0, \ldots, x_{n-1}) \in \mathbb{R}^n$ such that $(x_0, \ldots, x_{n-1}) \in \mathcal{S}$.

First, the algorithm executes the function `Constr_Gen(`ς`,0,`φ`)`. The result is a set of constraints $\mathcal{S}'$ in first-order theory of $(\mathbb{R}, +, -, 0, 1, \leqslant)$. Second, the algorithm executes the Fourier-Motzkin procedure in order to eliminate all existential and universal quantifiers. This results in a family of linear constraints containing only the variables $t_0, \ldots, t_{n-1}$.

Algorithm 1 Constraints generation for continuous semantics

Require: A finite discrete path ς of length $n > 0$, an MTL formula φ and a time bound T

Ensure: Family of linear inequalities $\mathcal{S}$ over $t_0, \ldots, t_{n-1}$

1: $\mathcal{S}' :=$`Constr_Gen(`ς`,`0`,`φ`)`

2: $\mathcal{S} :=$`Fourier_Motzkin(`$\mathcal{S}'$`,`t_0`,...,`t_{n-1}`)`

3: **return** $\mathcal{S}$

4:

5: **Function** `Constr_Gen(`ς`,`t`,`φ`)`

6: **case**(φ) :

$$
\begin{array}{lcl}
\varphi = p & : & \textbf{return } \Big(\bigvee_{k=0}^{n} p \in L(\varsigma[k]) \wedge \sum_{i=0}^{k} t_i \geqslant t \wedge \sum_{i=0}^{k-1} t_i < t \Big) \wedge \\
 & & \qquad t < T \\
\varphi = \neg\varphi_1 & : & \mathcal{S}' := \neg\texttt{Constr\_Gen(}\varsigma\texttt{,}t\texttt{,}\varphi_1\texttt{)} \\
\varphi = \varphi_1 \wedge \varphi_2 & : & \mathcal{S}' := \texttt{ Constr\_Gen(}\varsigma\texttt{,}t\texttt{,}\varphi_1\texttt{) } \wedge \texttt{ Constr\_Gen(}\varsigma\texttt{,}t\texttt{,}\varphi_2\texttt{)} \\
\varphi = \varphi_1 \mathcal{U}^{[a,b]} \varphi_2 & : & \mathcal{S}' := \exists t'. \big(t \leqslant t' < T \wedge t' - t \geqslant a \wedge t' - t < b \ \wedge \\
 & & \qquad \texttt{Constr\_Gen(}\varsigma\texttt{,}t'\texttt{,}\varphi_2\texttt{)} \ \wedge \\
 & & \qquad \forall t''.\ t \leqslant t'' < t' \Rightarrow \texttt{Constr\_Gen(}\varsigma\texttt{,}t''\texttt{,}\varphi_1\texttt{)} \big)
\end{array}
$$

7: **return** $\mathcal{S}'$

Although Algorithm 1 returns a set of linear inequalities $\mathcal{S}$ over the time at which jumps occur in the input discrete path ς, according to the input MTL formula φ, we have no certainty that the set of linear inequalities returned is actually correct. In order for $\mathcal{S}$ to be correct we need that any solution of the system must satisfy the formula φ when plugged back into the discrete path ς. Moreover, if there is a sequence of time jumps for ς that satisfies φ then that sequence must be a solution to $\mathcal{S}$. Theorem 4.1.7 proves the correctness of the system of linear inequalities returned by Algorithm 1.

Theorem 4.1.7 *Given a discrete path ς of length n, an* MTL *formula φ and a time bound T, we have that $(\varsigma[x_0, \ldots, x_{n-1}], 0) \models^{\mathsf{c}}_T \varphi$ iff $(x_0, \ldots, x_{n-1}) \in \mathcal{S}$, where $\mathcal{S}$ is returned by Algorithm 1. Recall here that the notation $(x_0, \ldots, x_{n-1}) \in \mathcal{S}$ indicates that the values of $(x_0, \ldots, x_{n-1})$ are solutions to the set of linear inequalities $\mathcal{S}$.*

Proof We show the theorem in two directions, as follows.

$(\varsigma[x_0, \ldots, x_{n-1}], 0) \models^{\mathsf{c}}_T \varphi \Rightarrow (x_0, \ldots, x_{n-1}) \in \mathcal{S}$. The proof proceeds by induction on the length of the formula φ. Let $\sigma = \varsigma[x_0, \ldots, x_{n-1}]$.

- `Base Case`. $\varphi = p$. By the semantics of MTL $(\varsigma[x_0, \ldots, x_{n-1}], t) \models^{\mathsf{c}}_T p \ \Rightarrow p \in \sigma[k] \wedge t < T$, where $\sigma[k] = \sigma@t$ for the smallest index k such that $\sum_{i=0}^{k} \sigma\langle i \rangle \geqslant t$. Since

k is the *smallest* index such that $\sum_{i=0}^{k} \sigma\langle i\rangle \geqslant t$, it follows that $\sum_{i=0}^{k-1} \sigma\langle i\rangle < t$. Combining all together we obtain: $p \in \sigma[k] \wedge t < T \wedge \sum_{i=0}^{k} \rho\langle i\rangle \geqslant t \wedge \sum_{i=0}^{k-1} \sigma\langle i\rangle < t$. Giving as input ς, p, T to Algorithm 1 yields: $\big(\bigvee_{k=0}^{n} p \in L(\varsigma[k]) \wedge \sum_{i=0}^{k} t_i \geqslant t \wedge \sum_{i=0}^{k-1} t_i < t\big) \wedge t < T$. The conjunction is satisfied for $\varsigma[k] = \rho[k]$.

- `Induction Step`. For $\varphi = \neg\varphi_1 \vee \varphi = \varphi_1 \wedge \varphi_2$, the proof follows by induction hypothesis.

  For $\varphi = \varphi_1 \mathcal{U}^{[a,b]} \varphi_2$ we have that:

$$(\sigma, t) \models^{\mathsf{c}}_{T} \varphi_1 \mathcal{U}^{[a,b]} \varphi_2 \Rightarrow \begin{cases} \exists t'.\ t \leqslant t' < T \wedge a \leqslant t' - t < b \wedge (\sigma, t') \models^{\mathsf{c}}_{T} \varphi_2 \\ \forall t''.\ t \leqslant t'' < t' \Rightarrow (\sigma, t'') \models^{\mathsf{c}}_{T} \varphi_1 \end{cases}$$

$$\texttt{Constr\_Gen}(\varsigma, t, \varphi_1 \mathcal{U}^{[\mathsf{a},\mathsf{b}]} \varphi_2) \Rightarrow \begin{cases} \exists t'.\ t \leqslant t' < T \wedge a \leqslant t' - t < b \wedge \\ \texttt{Constr\_Gen}(\varsigma, t', \varphi_2) \wedge \\ \forall t''.\ t \leqslant t'' < t' \Rightarrow \texttt{Constr\_Gen}(\varsigma, t'', \varphi_1) \end{cases}$$

  By induction hypothesis $(\sigma, t') \models^{\mathsf{c}}_{T} \varphi_2 \Rightarrow \texttt{Constr\_Gen}(\varsigma, t', \varphi_2)$ and $(\sigma, t'') \models^{\mathsf{c}}_{T} \varphi_1 \Rightarrow \texttt{Constr\_Gen}(\varsigma, t'', \varphi_2)$, concluding the proof.

$(x_0, \ldots, x_{n-1}) \in \mathcal{S} \Rightarrow (\varsigma[x_0, \ldots, x_{n-1}], 0) \models^{\mathsf{c}}_{T} \varphi$**.** The proof proceeds in a way similar to the opposite direction. ■

We now show the functioning of Algorithm 1 by means of an example.

Example 4.1.2 *Let $\mathcal{C}$ be a* CTMC *and let ς be the following finite discrete path on $\mathcal{C}$: $\varsigma = s_0 \to s_1 \to s_2 \to s_3$. Let $a, b \in$ AP, let $L(s_0) = \{a\}, L(s_1) = \{a\}, L(s_2) = \{a, b\}, L(s_3) = \{\varnothing\}$ and let $\varphi = a\ \mathcal{U}^{[1,2]} b$. The first step of Algorithm 1 consists of computing `Constr_Gen(ς,0,φ)`, which returns the following family of linear constraints $\mathcal{S}'$:*

$$\exists t'.\ 0 \leqslant t' < T \wedge t' \geqslant 1 \wedge t' < 2 \wedge \begin{cases} t_0 + t_1 + t_2 & \geqslant & t' \\ t_0 + t_1 & < & t' \end{cases} \wedge \tag{4.5}$$

$$\forall t''.\ 0 \leqslant t'' < t' \Rightarrow \left(t_0 \geqslant t'' \vee \begin{cases} t_0 + t_1 & \geqslant & t'' \\ t_0 & < & t'' \end{cases} \vee \begin{cases} t_0 + t_1 + t_2 & \geqslant & t'' \\ t_0 + t_1 & < & t'' \end{cases} \right). \tag{4.6}$$

The constraints in Equation (4.6) can always be verified given the constraints in Equation (4.5). Moreover, after the `Fourier_Motzkin` elimination for t', t'' in $\mathcal{S}'$ we obtain as a result the family of constraints $\mathcal{S}$:

$$\mathcal{S} = \begin{cases} t_0 + t_1 & < & 2 \\ t_0 + t_1 + t_2 & \geqslant & 1 \end{cases}.$$

The system $\mathcal{S}$ *can be represented using the matrix notation:* $\mathcal{S} := \{\mathbf{t} \in \mathbb{R}^n_{>0} \mid \mathbf{A} \cdot \mathbf{t} \trianglelefteq \mathbf{b}\}$, *for a given matrix* $\mathbf{A} \in \mathbb{R}^{m \times n}$, *vector* $\mathbf{b} \in \mathbb{R}^m$ *and* $\trianglelefteq \in \{<, \leqslant\}$. *The notation* $\mathbb{R}_{>0}$ *stands for the semi-closed interval* $(0, \infty) \subset \mathbb{R}$. *The matrices* $\mathbf{A}$, $\mathbf{t}$ *and* $\mathbf{b}$ *in* $\mathcal{S}$ *are:* $\mathbf{A} \in \mathbb{R}^{2\times 3}$, $\mathbf{t} \in \mathbb{R}^3_{>0}$ *and* $\mathbf{b} \in \mathbb{R}^2$. *More specifically:*

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad ; \quad \mathbf{t} = \begin{bmatrix} t_0 \\ t_1 \\ t_2 \end{bmatrix} \quad ; \quad \mathbf{b} = \begin{bmatrix} 2 \\ -1 \end{bmatrix} \quad .$$

In Algorithm 2 we also present a procedure that generates a family of linear constraints from a given MTL formula φ under the pointwise semantics. Notice that we do not need to use the `Fourier_Motzkin` elimination procedure, as the family of constraints obtained from `Constr_Gen(`ς`,0,`φ`)` contains no quantifiers.

Algorithm 2 Constraints generation for pointwise semantics

Require: A finite discrete path ς of length $n > 0$, an MTL formula φ and a time bound T

Ensure: Family of linear inequalities $\mathcal{S}$ over $t_0, \ldots, t_{n-1}$

1: **return** `Constr_Gen(`ς`,0,`φ`)`

2:

3: **Function** `Constr_Gen(`ς`,`i`,`φ`)`

4: **case**(φ) :

$\varphi = p$: **if** $p \in L(\varsigma[i])$ **return** $\sum_{k=0}^{i} t_k \leq T$ **else return** false

$\varphi = \neg\varphi_1$: $\mathcal{S} :=$ `¬Constr_Gen(`ς`,`i`,`φ_1`)`

$\varphi = \varphi_1 \wedge \varphi_2$: $\mathcal{S} :=$ `Constr_Gen(`ς`,`i`,`φ_1`)` $\wedge$ `Constr_Gen(`ς`,`i`,`φ_2`)`

$\varphi = \varphi_1 \mathcal{U}^{[a,b]} \varphi_2$: $\mathcal{S} := \big(\bigvee_{i'=i}^{n} \big($ `Constr_Gen(`ς`,`i'`,`φ_2`)` $\wedge\, a \leq \sum_{k=i}^{i'} t_k \leq b \wedge$ $\bigwedge_{i''=i}^{i'-1} \big($ `Constr_Gen(`ς`,`i''`,`φ_1`)` $\big)\big)$

5: **return** $\mathcal{S}$

Let $\mathcal{S}$ be the family of linear constraints obtained from Algorithm 1 and Algorithm 2. $\mathcal{S}$ is always defined as a *union* of convex polyhedra in $\mathbb{R}^n$, i.e., $\mathcal{S} = \bigvee_{i \in I} \bigwedge_{j \in J_i} c_{ij}$ where, for each $i \in I$, $\bigwedge_{j \in J_i} c_{ij}$ is a convex polyhedron.

Step 5.3: Computing probabilities

Given a CTMC $\mathcal{C}$, a discrete path ς of length N and the family of linear constraints $\mathcal{S}(t_0, \ldots, t_{N-1})$ obtained from Algorithm 1, the main task here is to compute the probability of $\varsigma[\mathcal{S}]$, i.e., $\Pr^{\mathcal{C}}(\varsigma[\mathcal{S}])$. To this end, we first add more constraints to $\mathcal{S}$, namely, for

$\mathcal{S} = \bigvee_{i\in I} \bigwedge_{j\in J_i} c_{ij}$ we obtain

$$\overline{\mathcal{S}} = \bigvee_{i\in I} \left(\bigwedge_{j\in J_i} c_{ij} \wedge (t_0 + \ldots + t_{N-1} > T \wedge t_0 + \ldots + t_{N-2} < T) \wedge \bigwedge_{0\leqslant k<N} t_k > 0 \right). \quad (4.7)$$

These new constraints are used to ensure that there are *exactly* N discrete jumps during the time interval $[0, T]$, and that each residence time is positive.

Now we have N random variables $t_0, \cdots, t_{N-1}$ corresponding to the residence time of each state $\varsigma[i]$ for $i \leqslant N$. The probability $\Pr^{\mathcal{C}}(\varsigma[\overline{\mathcal{S}}])$ is thus formulated as the joint probability $\Pr^{\mathcal{C}}(\overline{\mathcal{S}}(t_0, \cdots, t_{N-1}))$, where $t_i \sim \mathrm{Exp}(E(\varsigma[i]))$ for each $0 \leqslant i < N$, and $t_0, \cdots, t_{N-1}$ are bounded by the obtained family of linear constraints $\overline{\mathcal{S}}$. The value of the joint probability can be computed through the following multidimensional integration:

$$\Pr^{\mathcal{C}}(\varsigma[\overline{\mathcal{S}}]) = \underbrace{\int \cdots \int}_{N}_{\overline{\mathcal{S}}(\tau_0,\ldots,\tau_{N-1})} \prod_{i=0}^{N-1} E(s_i) \cdot P(s_i, s_{i+1}) \times e^{-E(s_i)\tau_i} d\tau_i \,. \quad (4.8)$$

The efficient algorithm that we will present shortly to compute $\Pr^{\mathcal{C}}(\varsigma[\overline{\mathcal{S}}])$ requires that the domain of integration is a convex polytope. We must be sure then, that we can express the set of linear inequalities $\mathcal{S}$ as a convex polytope (or disjoint union of convex polytopes).

Now we introduce Proposition 4.1.8 which shows a sufficient condition under which a set of linear inequalities defines a convex polyhedron.

Proposition 4.1.8 ([HUL94]) *Given any family of linear inequalities* $\overline{\mathcal{S}} = \bigvee_{i\in I} \bigwedge_{j\in J_i} c_{ij}$. *For each* $i \in I$, *we can write* $\bigwedge_{j\in J_i} c_{ij}$ *in matrix form* $\mathbf{A_i} \cdot \mathbf{t} \trianglelefteq \mathbf{b_i}$ *where* $\trianglelefteq \in \{<, \leqslant\}$, *and it is a polyhedron.*

Proof The proof can be found in [HUL94].

From Proposition 4.1.8, we have that $\overline{\mathcal{S}} = \bigvee_{\ell=0}^{k} C_\ell$ where each $C_\ell = \{\mathbf{t} \in \mathbb{R}^n_{>0} | \mathbf{A}_\ell \cdot \mathbf{t} \trianglelefteq \mathbf{b}_\ell\}$ defines a convex set. In general, we have no certainty that the union $\bigvee_{\ell=0}^{k} C_\ell$ is convex as well. In case that the union $\bigvee_{\ell=0}^{k} C_\ell$ is not convex, we use the inclusion-exclusion principle to compute $\Pr^{\mathcal{C}}(\varsigma[\overline{\mathcal{S}}])$ as follows:

$$\Pr^{\mathcal{C}}(\varsigma[\overline{\mathcal{S}}]) = \sum_{\ell=0}^{k} \Pr^{\mathcal{C}}(\varsigma[C_\ell]) - \sum_{i,j:0\leqslant i<j\leqslant k} \Pr^{\mathcal{C}}(\varsigma[C_i \wedge C_j]) +$$
$$\sum_{i,j,h:0\leqslant i<j<h\leqslant k} \Pr^{\mathcal{C}}(\varsigma[C_i \wedge C_j \wedge C_h]) - \cdots + (-1)^{k-1} \Pr^{\mathcal{C}}(\varsigma[C_0 \wedge \cdots \wedge C_k])$$

Remark 4.1.4 *In our case, the difference between $<$ and $\leqslant$ in the constraints is marginal, as they would yield the same probability, which can be seen from Equation (4.8).*

For an index set $L \subseteq \{0, \ldots, k\}$ we write $D = \bigwedge_{\ell \in L} C_\ell$, where C_ℓ defines a polyhedron. By Proposition 4.1.8, D defines a polyhedron as well. We rewrite $\Pr^{\mathcal{C}}(\varsigma[D])$ as:

$$\begin{aligned} \Pr^{\mathcal{C}}(\varsigma[D]) &= \prod_{i=0}^{N-1} E(s_i) \cdot P(s_i, s_{i+1}) \cdot \underbrace{\int \cdots \int}_{N}{}_{D} \prod_{i=0}^{N-1} e^{-E(s_i)\tau_i} d\tau_i \\ &= \prod_{i=0}^{N-1} E(s_i) \cdot P(s_i, s_{i+1}) \cdot \underbrace{\int \cdots \int}_{N}{}_{D} e^{-\vec{E}\cdot\vec{\tau}} d\vec{\tau}, \end{aligned} \tag{4.9}$$

where $\vec{E} = [E(s_0), \ldots, E(s_{N-1})]$, $\vec{\tau} = [\tau_0, \ldots, \tau_{N-1}]$ and $\vec{E} \cdot \vec{\tau} = \sum_{i=0}^{N-1} E(s_i) \cdot \tau_i$. We use the algorithm of [LZ01] (see Example 4.1.3) to compute efficiently the multidimensional integral $\int \cdots \int_D e^{-\vec{E}\cdot\vec{\tau}} d\vec{\tau}$ based on the Laplace transform.

Remark 4.1.5 *We remark here that on the positive side, the algorithm in [LZ01] is an exact algorithm. It does not approximate the value of the multidimensional integral $\int \cdots \int_D e^{-\vec{E}\cdot\vec{\tau}} d\vec{\tau}$ but it calculates the exact value of the integral.*

On the negative side, the algorithm in [LZ01] is not general and cannot be used for arbitrary probability distributions. The reason for that is that the algorithm in [LZ01] is based on Laplace transform to compute volume integrals. Laplace transforms remain basically unchanged, other than a shift, when multiplied by an exponential function. Thus, due to the CTMC*s property of exponential residence time in system states we can reuse the algorithm of [LZ01] directly to compute the probability of timed-paths of the* CTMC *under consideration. The same would not apply if the probabilistic jumps would follow a different probability distribution. In such cases, one would need to approximate the integrals using sampling techniques, such as Monte Carlo, which have not been explored in this thesis.*

An example of how to compute the integral $\int \cdots \int_D e^{-\vec{E}\cdot\vec{\tau}} d\vec{\tau}$ for a convex set D is given in Example 4.1.3 below.

Example 4.1.3 *We now show a concrete example of how it is possible to compute the truth value of* MTL *formulas over finite paths of a* CTMC*. Let $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$ be the* CTMC *in Figure 3.2. Let $\varphi = a\mathcal{U}^{[1,2]}b$ be an MTL formula, let $\varsigma = s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow s_3$ be a finite discrete path of length three in $\mathcal{C}$, and let $T = 3$ be the time bound for the verification of φ. The first step consists of running Algorithm 1 to find the family of linear constraints $\mathcal{S}$.*

System of linear inequalities. *We have already seen in Example 4.1.2 that Algorithm 1 on ς, T, φ on the* CTMC *in Figure 3.2 returns:*

$$\mathcal{S}' = \begin{cases} t_0 + t_1 < 2 \\ t_0 + t_1 + t_2 \geqslant 1 \end{cases}$$

Figure 4.1: Set of linear inequalities returned by Algorithm 1

Adding the two inequalities which ensure that there are exactly three jumps in T and after simplifications we get the system in Figure 4.2.

$$\mathcal{S} = \begin{cases} t_0 + t_1 < 2 \\ t_0 + t_1 + t_2 \geqslant 3 \end{cases}$$

Figure 4.2: Complete set of linear inequalities

Changing all the inequalities with a "$\geqslant$" to their equivalent with "$\leqslant$", it is possible to express the system in matrix form $\mathcal{S} = \mathbf{At} \leqslant \mathbf{b}$, where: $\mathbf{A} \in \mathbb{R}^{2\times 3}$, $\mathbf{t} \in \mathbb{R}^3$ and $\mathbf{b} \in \mathbb{R}^2$ (see Figure 4.3).

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad ; \quad \mathbf{t} = \begin{bmatrix} t_0 \\ t_1 \\ t_2 \end{bmatrix} \quad ; \quad \mathbf{b} = \begin{bmatrix} 2 \\ -3 \end{bmatrix} \quad .$$

Figure 4.3: Matrix representation of $\mathcal{S}$

The example continues now showing the algorithm in [LZ01] for computing the integration of an exponential over $\mathcal{S}$. Let $\lambda \in \mathbb{R}^3$ be the vector of rates associated to the states s_0, s_1 and s_2 of C. In other words, $\lambda_0 = E(s_0)$, $\lambda_1 = E(s_1)$ and $\lambda_2 = E(s_2)$. Fix $E(s_0) = 2$, $E(s_1) = 1$ and $E(s_2) = 2$. We must calculate:

$$z(b) := \int_{\mathcal{S}(b)} e^{-(\lambda_0\tau_0+\lambda_1\tau_1+\lambda_2\tau_2)} d\tau_0 d\tau_1 d\tau_2$$

with Laplace transform given by:

$$Z(\delta) = \frac{1}{\delta_1\delta_2(\delta_1-\delta_2+1)(\delta_1-\delta_2+2)(-\delta_2+2)} \quad \Re(\delta) > 0 \wedge \Re(\mathbf{A}'\delta + \lambda) > 0$$

where $Z(\delta) : \mathbb{C}^2 \to \mathbb{C}$ and $\Re(\delta)$ indicates the real part of δ.

The inverse Laplace transform of $Z(\delta)$ is:

$$z(b) = \frac{1}{(2\pi\imath)^2} \int_{c_2-\imath\infty}^{c_2+\imath\infty} \int_{c_1-\imath\infty}^{c_1+\imath\infty} \frac{e^{(2\delta_1-3\delta_2)}}{\delta_1\delta_2(\delta_1-\delta_2+1)(\delta_1-\delta_2+2)(-\delta_2+2)} d\delta_1 d\delta_2$$

The real vector $\mathbf{c} \in \mathbb{R}^2$ must satisfy $\mathbf{c} > 0$, $\mathbf{A}'\mathbf{c} - \lambda > 0$

To make calculations easier we evaluate $h(p) = z(pb)$ *at the point* $p = 1$.

$$h(p) = z(pb) = \frac{1}{(2\pi\imath)^2} \int_{c_2-\imath\infty}^{c_2+\imath\infty} \int_{c_1-\imath\infty}^{c_1+\imath\infty} \frac{e^{(2\delta_1-3\delta_2)p}}{\delta_1\delta_2(\delta_1-\delta_2+1)(\delta_1-\delta_2+2)(-\delta_2+2)} d\delta_1 d\delta_2 \quad (4.10)$$

In order to calculate the value in Equation 4.10 we need to follow the steps below.

- ***Determining the vector*** $\mathbf{c}$. *Before computing the integration, we must find suitable values for the elements of the real vector* $\mathbf{c}$ *such that:* $\mathbf{c} > 0$, $\mathbf{A}'\mathbf{c} - \lambda > 0$. *It is left to the reader to verify that* $c_1 = 4$ *and* $c_2 = 1/2$ *satisfy the set of constraints.*

- ***Integration of the exponential function over*** $\mathcal{S}(b)$. *The first step of the integration consist of integrating the function* $h(p)$ *with respect to* δ_1*; that is, evaluate the residues at the poles* $\delta_1 = 0$, $\delta_1 = \delta_2 - 2$ *and* $\delta_1 = \delta_2 - 1$. *Since the argument of the exponential function (with respect to* δ_1*) is positive, we must consider the left-hand side of the line* $c_1 \pm \imath\infty$ *[Con78]. More specifically, in this example, all the poles must be considered since they lie to the left of the line* $c_1 \pm \imath\infty$. *In fact, the real part of* δ_1*, i.e.,* $\Re(\delta_1) = c_1 = 4$*, lies on the right side of the value obtained by substituting to the variable* δ_2*, in each pole, the value* $\Re(\delta_2) = c_2 = 1/2$.

  We obtain:

$$h(p) = \frac{1}{(2\pi\imath)} \int_{c_2-\imath\infty}^{c_2+\imath\infty} (I_1 + I_2 + I_3) d\delta_2 \quad (4.11)$$

  where

$$I_1(\delta_2) = -\frac{e^{-3\delta_2 p}}{\delta_2(\delta_2-2)^2(\delta_2-1)} \quad (4.12)$$

$$I_2(\delta_2) = \frac{e^{(-\delta_2-4)p}}{\delta_2(\delta_2-2)^2} \quad (4.13)$$

$$I_3(\delta_2) = -\frac{e^{(-\delta_2-2)p}}{\delta_2(\delta_2-2)(\delta_2-1)} \quad (4.14)$$

  We must integrate now I_1, I_2 *and* I_3 *with respect to* δ_2. *We start with* I_1 *and its poles, which are:* $\delta_2 = 0$, $\delta_2 = 2$ *(multiple pole) and* $\delta_2 = 1$. *The coefficient of the exponential function (with respect to* δ_2*) is negative. Thus, this time, we must consider the poles at the right side of the line* $c_2 \pm \imath\infty$ *and take the negative value of the residues (the path of integration has a negative orientation). Since* $\Re(\delta_2) = 1/2$ *the poles* $\delta_2 = 2$ *and* $\delta_2 = 1$ *lie on the right side of the line, whereas* $\delta_2 = 0$ *does not. The simple pole* $\delta_2 = 1$ *is computed in the usual way and it yields:*

$$I_{12} = \frac{e^{-3p}}{1} \quad (4.15)$$

  The multiple pole of degree two, $\delta_2 = 2$*, is computed using the Cauchy residue formula for poles of degree greater than one [Con78]. It returns:*

$$I_{13} = -\frac{9e^{-6p}}{4} \quad (4.16)$$

Consider now I_2 and its poles, which are $\delta_2 = 0$ and $\delta_2 = 2$. The coefficient of the exponential function (with respect to δ_2) is negative. We must consider the poles that lie on the right side of the line $c_2 \pm \imath\infty$, in this case only $\delta_2 = 2$. After calculating the residue we get:

$$I_{21} = \frac{3e^{-6p}}{4} \tag{4.17}$$

Finally, when integrating I_3 (with respect to δ_2) we must consider the poles on the right side of the line $c_2 \pm \imath\infty$, i.e., $\delta_2 = 2$ and $\delta_2 = 1$.

$$I_{31} = \frac{e^{-4p}}{2} \tag{4.18}$$

and

$$I_{32} = -\frac{e^{-3p}}{2} \tag{4.19}$$

Hence, adding up the above three partial results yields:

$$h(p) = \frac{e^{-3p}}{1} - \frac{9e^{-6p}}{4} + \frac{3e^{-6p}}{4} + \frac{e^{-4p}}{2} - \frac{e^{-3p}}{1} \tag{4.20}$$

Evaluating $h(1)$ gives us the desired result, which is approximately 0.00544.

The reader should notice that the result does not change if we choose different values for c_1 and c_2 (for instance $c_1 = 3$ and $c_2 = 1/2$). The only restriction is that, at each step i of the integration, no poles must lie on the line $c_i \pm \imath\infty$. However, [LZ01] gives an algorithm to choose the values for the c_i in such a way that no c_i will be on the integration line during each step of integration.

Admittedly, it is costly to apply the inclusion-exclusion principle to compute the probabilities. In the worst case, any union of two components is not convex. However, in practice one could expect that the union of some components is still convex, so they can be computed as a whole. We emphasise that efficient algorithms to decide whether the union of two polyhedra is convex do exist, see e.g., [BFT01].

Step 5.4.: Sum up all the probabilities for each discrete path to obtain $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$

The last step of the algorithm is to sum up all the probabilities in order to obtain the final value of $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$, namely the probability of all the timed paths of the CTMC $\mathcal{C}$ of length at most N that satisfy the MTL formula φ in between $[0, T]$.

4.1.1 Complete algorithm and correctness results

We summarise the time-bounded verification algorithm for a CTMC $\mathcal{C}$ against an MTL formula φ in Algorithm 3. Recall that Λ is the maximal exit rate appearing in $\mathcal{C}$.

Algorithm 3 Time-bounded verification of a CTMC $\mathcal{C}$ against an MTL formula φ

Require: A CTMC $\mathcal{C}$, an MTL formula φ, a time bound T and an error ε

Ensure: Compute the probability of the formula φ being satisfied in the CTMC $\mathcal{C}$ within the time bound T and in at most N steps, namely $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$

1: Choose an integer $N \geqslant \Lambda T e^2 + \ln(\frac{1}{\varepsilon})$
2: Transform the MTL formula φ into its untimed LTL version $\widetilde{\varphi}$ and generate NFA $\mathcal{A}_{\widetilde{\varphi}}$ out of $\widetilde{\varphi}$
3: Compute the product $\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}$
4: **for** each discrete path ς of $(\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}) \downarrow_1$ of length $n < N$ **do**
5: Generate the family of linear constraints $\mathcal{S}(t_0, \ldots, t_{n-1})$ using Algorithm 1 (or Algorithm 2)
6: Calculate the probability p of $\varsigma[\mathcal{S}]$
7: $\Pr^{\mathcal{C}}_{T,<N}(\varphi) := \Pr^{\mathcal{C}}_{T,<N}(\varphi) + p$
8: **end for**
9: **return** $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$

We now introduce some lemmas that are useful in order to prove the correctness of Algorithm 3.

The first lemma, Lemma 4.1.9, shows that the probability of all the timed paths of length less than N that satisfy the MTL formula φ in the interval $[0, T]$ can be obtained as the sum of the probability of the set of all timed paths that satisfy φ and have length exactly $1, 2, \ldots, N-1$. In other words, Lemma 4.1.9 states that the set of linear inequalities returned from paths of length exactly i must be disjoint from the set of linear inequalities returned from paths of length exactly j when $j \neq i$.

Lemma 4.1.9

$$\Pr^{\mathcal{C}}_{T,<N}(\varphi) := \sum_{i=0}^{N-1} \Pr^{\mathcal{C}}_{T,=i}(\varphi),$$

where $\Pr^{\mathcal{C}}_{T,=i}(\varphi)$ *expresses the probability to satisfy the formula* φ *in a path with exactly* i *jumps.*

Proof The proof works by induction on the length of the path. The base case is trivial. For $N+1$ we have:

$$\begin{aligned} Pr^{\mathcal{C}}_{T,<N+1}(\varphi) &= \sum_{i=0}^{N+1} Pr^{\mathcal{C}}_{T,=i}(\varphi) \\ &= \sum_{i=0}^{N} Pr^{\mathcal{C}}_{T,=i}(\varphi) + Pr^{\mathcal{C}}_{T,=N+1}(\varphi)\ . \end{aligned}$$

By induction hypothesis, $Pr^{\mathcal{C}}_{T,=N}(\varphi) = \sum_{i=0}^{N} Pr^{\mathcal{C}}_{T,=i}(\varphi)$ defines disjoint families of linear inequalities. It remains to prove that the probability of a path with exactly $N+1$ jumps in

T defines a family of linear constraints that is disjoint from the family of linear constraints defined by the paths of length exactly N. After running Algorithm 1 (or Algorithm 2) we always add the constraints in Equation 4.7. In particular, considering a path of length N and a path of length $N+1$, we have that:

$$(1)\begin{cases} t_0 + \ldots t_N > T \\ t_0 + \ldots t_{N-1} < T \end{cases} \qquad (2)\begin{cases} t_0 + \ldots t_{N+1} > T \\ t_0 + \ldots t_N < T \end{cases}$$

The system of linear inequalities (1) represents the constraint of having exactly N jumps in T (similarly for (2)). The first inequality in (1) and the second inequality in (2) can never be satisfied at the same time. Hence, the systems (1) and (2) define two disjoint sets, and this concludes the proof. ■

For the correctness, we first note that the error is bounded by $\Pr^{\mathcal{C}}_{T,\geqslant N}(\varphi)$, which is in turn bounded by the probability of the set of timed paths with at least N discrete jumps in $[0,T]$. Then Theorem 4.1.3 yields the bound, as follows.

Lemma 4.1.10 *Given a* CTMC $\mathcal{C}$*, an* MTL *formula* φ*, a time bound* T *and* $N \in \mathbb{N}$

$$\Pr^{\mathcal{C}}_{T}(\varphi) - \Pr^{\mathcal{C}}_{T,<N}(\varphi) \leqslant \epsilon(T,N).$$

Proof Observe that $\Pr^{\mathcal{C}}_{T}(\varphi) - \Pr^{\mathcal{C}}_{T,<N}(\varphi) = \Pr^{\mathcal{C}}_{T,\geqslant N}(\varphi) \leqslant \Pr^{\mathcal{C}}(Paths^{\mathcal{C}}_{T,\geqslant N})$. The claim then follows from Theorem 4.1.3. ■

Theorem 4.1.11 *Algorithm 3 computes* $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$.

Proof We provide a sketch here. Observe the following facts:

1. $\Pr^{\mathcal{C}}_{T,<N}(\varphi) = \Pr^{\mathcal{C}}(Paths^{\mathcal{C}}_{T,<N}(\varphi))$;
2. For all infinite timed paths $\rho \in Paths^{\mathcal{C}}_{T,<N}(\varphi)$, $\mathbb{S}(\rho) \in (\mathcal{C} \otimes \mathcal{A}_{\widetilde{\varphi}}) \restriction_1$. This follows from Proposition 4.1.6;
3. Using Algorithm 1 (or Algorithm 2) we generate the family of linear constraints $\mathcal{S}$ such that $(\varsigma_i[x_0,\ldots,x_{n-1}],0) \models^{\mathsf{c}}_T \varphi$ iff $(x_0,\ldots,x_{n-1}) \in \mathcal{S}$;
4. Each $\varsigma_i[x_0,\ldots,x_{n-1}]$ corresponds to a timed path $\rho \in Paths^{\mathcal{C}}_{T,<N}(\varphi)$;
5. We calculate the probability of $\varsigma_i[\mathcal{S}]$, i.e., the probability of any possible timed path of length less than N;
6. From Lemma 4.1.9 we can then sum up the values of probabilities obtained from each path of length i (for $0 \leqslant i < N$);
7. Therefore, Algorithm 3 computes $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$. ■

4.1.1.1 Complexity

The main complexity of Algorithm 3 comes from the following factors:

- Enumerate all the paths of length up to N. This step is exponential in N;
- Generate the system of linear constraints for the MTL formula φ. This is exponential in the number of nested formulas of φ. However, we remark here that usually the formula under consideration will be small compared to the length N of the paths to generate;
- Compute the probability of the timed paths that satisfy the MTL formula φ through the algorithm of [LZ01]. This is in turn reduced to compute a volume integral over a convex polytope which can be expressed in matrix notation as $\mathcal{S} := \{\mathbf{t} \in \mathbb{R}^n_{>0} \mid \mathbf{A} \cdot \mathbf{t} \trianglelefteq \mathbf{b}\}$, for a given matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, vector $\mathbf{b} \in \mathbb{R}^m$ and $\trianglelefteq \in \{<, \leqslant\}$. The complexity of the Algorithm in [LZ01] is exponential in m. In our case, n is related to the path length and m to the complexity of the formula. Thus, one can assume that m is much smaller than n.

  Summarising, the main bottle neck of the algorithm is the enumeration of all paths of length up to N, which has complexity of $\mathcal{O}(2^N)$.

In this section we have provided an algorithm, Algorithm 3, that takes as input a CTMC $\mathcal{C}$, an MTL formula φ, a time bound T and a maximum tolerable error ε, and computes the probability of φ being true in $\mathcal{C}$ in the interval $[0, T]$, namely $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$. We approximate the real probability $\Pr^{\mathcal{C}}_{T}(\varphi)$ with $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$ bounding to N the maximum number of jumps that happen in the interval $[0, T]$. The approximated value of probability, $\Pr^{\mathcal{C}}_{T,<N}(\varphi)$, is ε-close to the real value $\Pr^{\mathcal{C}}_{T}(\varphi)$.

The algorithm of [LZ01] to compute the complex integral of Equation 4.9 has been implemented in Matlab [MAT13] using the symbolic toolbox.

To our knowledge we are the first to propose a solution to the problem of model checking MTL formulas on CTMCs. Our algorithm needs to perform a vast range of transformations and calculations in order to compute the final value of probability. The main drawbacks of our approach are the following.

1. We need to enumerate all the possible paths of length up to N and this is exponential in N;
2. The Matlab implementation of the algorithm of [LZ01] runs quite slowly due to the use of the symbolic toolbox.

Thus, in order to use Algorithm 3 for real-life examples we need to optimise the Matlab implementation by eliminating the use of symbolic variables, and avoiding the enumeration of all the paths of length up to N. Although not yet explored, we believe that the use

of random algorithms to explore the CTMC could help avoid the enumeration of all the paths of length up to N. We leave this as future work.

4.2 Verifying continuous-time Markov chains against TAs

In this section, we show how the procedure outlined in Section 4.1 can be adapted to verify TA specifications on CTMCs. The content of this section is based on a published conference paper [CDKM11] by the author of this thesis and his colleagues at Oxford. The reason why we want to include properties expressed as TAs is simple. Some real-time properties cannot be expressed as MTL but they can be expressed as a timed automaton. In fact, in [BRP13] the authors show that counting modalities, i.e., operators to count the number of events, enrich the expressiveness of MTL. On the other hand, TAs allow us to express some counting properties (see [BHHK03] page 286).

We illustrate the concept with an example.

Example 4.2.1 *Consider the case in which we want to model a robot that walks in a hotel and has to accomplish multiple tasks. The robot can move freely between different rooms of the hotel. The waiting time of the robot in any room is exponentially distributed. The hotel is modelled as a* CTMC *in which each room is represented as a* CTMC *state. Each time the robot decides whether to stay in the same room or to visit a new one. If the robot chooses to change room the next room is chosen probabilistically according to a discrete probability distribution. In other words, if the robot decides to move to another room then it must follow the transition relation of the* CTMC *from the current state in which the robot resides to a successor state.*

The situation is pictorially represented in Figure 4.4.

We would like to study the reliability of the robot, i.e, we want to determine the tasks that the robot successfully completes with probability of at least 95%*. The tasks are represented as real-time properties. One of the tasks is to reach one of the red rooms in less than* 2 *seconds passing first from one of the green rooms. This seems to be a fairly simple real-time property. However, it is not possible to express the latter property as an* MTL *formula. On the other hand, it is trivial to define an automaton which checks the above-mentioned property. The automaton is shown in Figure 4.5.*

Example 4.2.1 shows that including TAs as a specification formalism would enrich the real-time properties that one can express, making the model checking problem more relevant.

This section is dedicated to the following problem. Given a CTMC $\mathcal{C}$, a TA $\mathcal{A}$ and a timed bound T, find the probability of all the timed paths ρ of $\mathcal{C}$ which are accepted by $\mathcal{A}$.

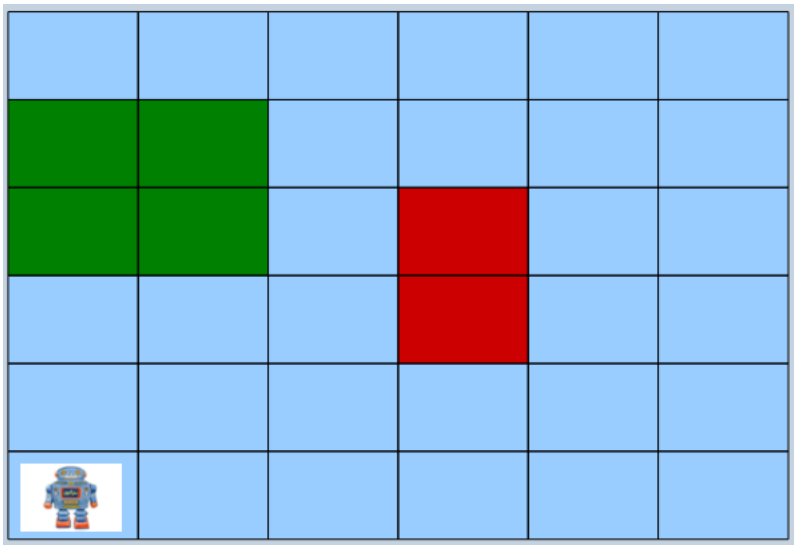


Figure 4.4: Example of Robot in a hotel

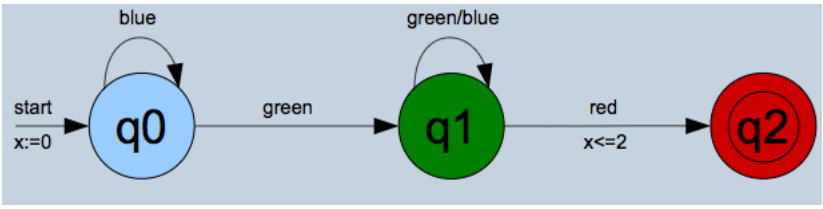


Figure 4.5: Automaton property

Model checking problem

Input: A CTMC $\mathcal{C}$ and a TA $\mathcal{A}$

Problem: Find the probability of the set of timed paths of $\mathcal{C}$ that are accepted by $\mathcal{A}$ over a time interval of fixed, bounded length.

Formally, we intend to compute $\Pr^{\mathcal{C}}_{T}(\mathcal{A}) := \Pr^{\mathcal{C}}(\{\rho \in \mathit{Paths}^{\mathcal{C}}_{T} \mid \rho \models_{T} \mathcal{A}\})$. As in the case of MTL specifications, we bound $\Pr^{\mathcal{C}}_{T}(\mathcal{A})$ by $\Pr^{\mathcal{C}}_{T,<N}(\mathcal{A}) := \Pr^{\mathcal{C}}(\mathit{Paths}^{\mathcal{C}}_{T,<N}(\mathcal{A}))$, such that $\Pr^{\mathcal{C}}_{T}(\mathcal{A}) - \Pr^{\mathcal{C}}_{T,<N}(\mathcal{A}) < \varepsilon$ for $\varepsilon > 0$. The measurability of the set of paths $\mathit{Paths}^{\mathcal{C}}_{T,<N}(\mathcal{A}) := \{\rho \in \mathit{Paths}^{\mathcal{C}}_{T,<N} \mid \rho \models_{T} \mathcal{A}\}$ can be shown as in [CHKM11].

The algorithm is very similar to the one described in Section 4.1 and can be summarised in the following four steps:

Step 1. Bound the number of jumps N in the interval of time $[0, T]$ in order to get the desired error ε;

Step 2. Build the product $\mathcal{C} \times \mathcal{A}$;

Step 3. Search for all the discrete paths ς in $\mathcal{C} \times \mathcal{A}$ of length at most N, and, for each of those, generate the set of linear inequalities $\mathcal{S}$;

Step 4. Calculate the probability of ς under the constraints in $\mathcal{S}$;

Step 5. Sum up all the probabilities.

Step 1: Bound the number of jumps N in the interval of time $[0, T]$ in order to get the desired error ε

The bound is the same as Step 1 of Section 4.1.

Step 2: Build the product $\mathcal{C} \times \mathcal{A}$

We remove all the guards, clocks and invariants from the timed automaton $\mathcal{A}$ and then follow the construction of the product in Step 4 Section 4.1.

Step 3: Search for all the discrete paths ς in $\mathcal{C} \times \mathcal{A}$ of length at most N, and, for each of those, generate the set of linear inequalities $\mathcal{S}$

The main difference between specifications given as MTL formulas or as TAs lies in the set of linear constraints generated. In order to derive the set of linear constraints, we first need to generate a graph $\mathcal{G}$ according to the construction from Alur *et. al.* in [AKV98]. There, the authors show how to, given a discrete path π of TA $\mathcal{A}$, construct a graph $\mathcal{G}$ such that $\mathcal{A}$ has a run over π if and only if $\mathcal{G}$ has no negative cost cycle. The graph $\mathcal{G}$ has exactly n nodes and the number of edges of $\mathcal{G}$ depends on the numbers of guards and invariants in $\mathcal{A}$ (see [AKV98] for details). Each edge $e = (i, j)$ (connecting node i to node j) is labelled with a value c such that $c \in \mathcal{H}$, where

$$\mathcal{H} = \{\ldots -2, -1, 0, 1, 2, \ldots\} \cup \{\ldots -2^-, -1^-, 0^-, 1^-, 2^-, \ldots\} \cup \{-\infty, \infty\}$$

The set $\mathcal{H}$ is used to characterise strict and non-strict constraints in $\mathcal{A}$.

For each discrete path ς of the CTMC $\mathcal{C}$ we define $\Pi_\varsigma = \{\pi \mid \pi_i \xrightarrow{L(\varsigma[i])} \pi_{i+1}$ for all $0 \leqslant i \leqslant n-1\}$.

Theorem 4.2.1 *Given a discrete path ς of length n, a* TA *$\mathcal{A}$ and a time bound T, we have that $\varsigma[t_0, \ldots, t_{n-1}]$ is accepted by $\mathcal{A}$ iff $(t_0, \ldots, t_{n-1}) \in \mathcal{S}$, where $\mathcal{S}$ is returned by Algorithm 4.*

Proof The proof follows from Proposition 1 of [AKV98].

Step 4: Calculate the probability of ς under the constraints in $\mathcal{S}$

Same algorithm as in Step 5.3 of Section 4.1.

Algorithm 4 Constraints generation for TA

Require: A finite discrete path ς of length $n > 0$, a TA $\mathcal{A}$, and a time bound T

Ensure: Family of linear constraints $\mathcal{S}$

1: For the discrete path ς compute the set Π_ς
2: **for** each $\pi \in \Pi_\varsigma$ **do**
3: Generate the graph $\mathcal{G}$
4: $\mathcal{S}_\pi := \varnothing$
5: **for** each edge $e(i,j) \in \mathcal{G}$ labeled with c **do**
6: $\mathcal{S}_\pi := \mathcal{S}_\pi \wedge t_i - t_j < c$
7: **end for**
8: $\mathcal{S} := \mathcal{S} \vee \left(\mathcal{S}_\pi \wedge (t_0 + \ldots + t_{n-1} > T \wedge t_0 + \ldots + t_{n-2} < T) \wedge \bigwedge_{0 \leqslant k < n} t_k > 0 \right)$
9: **end for**
10: **return** $\mathcal{S}$

Step 5: Sum up all the probabilities

The last step of the algorithm is to sum up all the probabilities in order to obtain the final value of $\Pr^{\mathcal{C}}_{T,<N}(\mathcal{A})$, namely, the probability of the timed paths in the CTMC $\mathcal{C}$ of length at most N that are accepted by the TA $\mathcal{A}$ in the interval $[0, T]$.

4.2.1 Complete algorithm and correctness results

Given a timed automaton $\mathcal{A}$, we write $\bar{\mathcal{A}}$ to denote the NFA obtained by removing all the guards, clocks and invariants from $\mathcal{A}$. The product $\mathcal{C} \otimes \bar{\mathcal{A}}$ follows Definition 4.1.1. Similarly to Proposition 4.1.6, we have the following.

Proposition 4.2.2 *For any* CTMC $\mathcal{C}$ *and NFA* $\bar{\mathcal{A}}$, $\mathbb{S}(Paths^{\mathcal{C}}_T(\mathcal{A})) \subseteq \{C_d(\varsigma) \mid \varsigma \in \Diamond Loc_{\mathbf{F}} \downharpoonright_1\}$, *where* $Loc_{\mathbf{F}}$ *is the set of final locations in* $\mathcal{C} \otimes \bar{\mathcal{A}}$.

Proof The proof follows similar reasoning of the proof of Proposition 4.1.6.

The approximation algorithm for time-bounded verification of a TA specification $\mathcal{A}$ is given in Algorithm 5.

Now we present two lemmas, Lemma 4.2.3 and Lemma 4.2.4, which will be used to prove the correctness of Algorithm 5. The lemmas are the equivalent of the lemmas that we have presented in Section 4.1.1 in order to prove the correctness of Algorithm 3.

Lemma 4.2.3

$$\Pr^{\mathcal{C}}_{T,<N}(\mathcal{A}) := \sum_{i=0}^{N-1} \Pr^{\mathcal{C}}_{T,=i}(\mathcal{A}),$$

where $\Pr^{\mathcal{C}}_{T,=i}(\mathcal{A})$ *expresses the probability to satisfy the specification* $\mathcal{A}$ *in a path with exactly* exactly i *jumps.*

Algorithm 5 Time-bounded verification of a TA specification $\mathcal{A}$ against a CTMC $\mathcal{C}$

Require: $\mathcal{C}$, $\mathcal{A}$, T and ε

Ensure: $\Pr^{\mathcal{C}}_{T,<N}(\mathcal{A})$

1: Choose an integer $N \geq \Lambda T e^2 + \ln(\frac{1}{\varepsilon})$
2: **for** each discrete path ς of $(\mathcal{C} \otimes \bar{\mathcal{A}}) \downarrow_1$ of length $n < N$ **do**
3: Calculate the family of linear constraints $\mathcal{S}(t_0, \ldots, t_{n-1})$ with Algorithm 4
4: Calculate the probability p of $\varsigma[\mathcal{S}]$
5: $\Pr^{\mathcal{C}}_{T,<N}(\mathcal{A}) := \Pr^{\mathcal{C}}_{T,<N}(\mathcal{A}) + p$
6: **end for**
7: **return** $\Pr^{\mathcal{C}}_{T,<N}(\mathcal{A})$

Proof The proof is the same as the one presented for Lemma 4.1.9.

Lemma 4.2.4 *Given a* CTMC $\mathcal{C}$*, a* TA *specification* $\mathcal{A}$*, a time bound* T *and* $N \in \mathbb{N}$*, we have that*

$$\Pr^{\mathcal{C}}_{T}(\mathcal{A}) - \Pr^{\mathcal{C}}_{T,<N}(\mathcal{A}) \leqslant \epsilon(T, N).$$

Proof The proof is exactly the same of the one presented for Lemma 4.1.10.

Finally, Theorem 4.2.5 proves the correctness of Algorithm 5.

Theorem 4.2.5 *Algorithm 5 computes* $\Pr^{\mathcal{C}}_{T,<N}(\mathcal{A})$.

Proof The proof is exactly the same of the one presented for Theorem 4.1.11.

4.2.1.1 Complexity

Algorithm 5 follows the same steps of Algorithm 3 from a complexity point of view. Thus, the main bottle neck of Algorithm 5 is the path enumeration which yields a complexity of $\mathcal{O}(2^N)$.

4.3 Verifying continuous-time Markov chains against LDPs

In this section we study the problem of verifying CTMCs against *Linear Duration Properties* (LDP), i.e., properties stated as conjunctions of linear constraints over the total duration of time spent in states that satisfy a given property. The content of this section is based on a journal paper [CDKM13b] by the author of this thesis and his colleagues at Oxford.

As mentioned in the introduction to this chapter, the problem can be summarised as follows.

Model checking problem

Input: A CTMC $\mathcal{C}$ and an LDP φ

Problem: Find the probability of the set of timed paths of $\mathcal{C}$ that satisfy φ over a time interval of fixed, bounded length.

We identify two classes of LDP properties, *eventuality duration properties* (EDP) and *invariance duration properties* (IDP), respectively referring to the reachability of a set of goal states within a time bound; and the continuous satisfaction of a duration property over an execution path. In this section we are interested to address the question of how to compute the probability of the set of infinite timed paths of the CTMC that satisfy a given LDP. We design algorithms that approximate these probabilities up to a given precision, stating their complexity and showing how to bound the approximation error. The algorithms mainly employ an adaptation of uniformisation and the computation of volumes of multi-dimensional integrals under systems of linear constraints, together with different mechanisms to bound the errors.

More specifically, we identify the following classes of LDP problems.

Problem statement. Corresponding to Definition 3.2.8, we focus on algorithmic verification problems for two classes of LDP, i.e., *Eventuality Duration Property* (EDP) and *Invariance Duration Property* (IDP), given below.

- *Verification of EDP.* Formally, given a CTMC $\mathcal{C}$, a set of *goal* states $G \subseteq S$, and an LDP $\Phi = \int 1 \leqslant T \rightarrow \varphi$, compute the probability of the set of infinite timed paths of $\mathcal{C}$ satisfying Φ under the *finitary satisfaction condition*, see Definition 3.2.8. Depending on T, we distinguish two cases:
  - Time-bounded case: $T < \infty$, for which we denote the desired probability by $\mathit{Prob}(\mathcal{C} \models^{G} \Phi)$
  - Unbounded case: $T = \infty$, for which we denote the desired probability by $\mathit{Prob}(\mathcal{C} \models^{G} \varphi)$. Note that this is valid as, in this case, Φ is simply equivalent to φ.

  The algorithms for these two cases are given in Section 4.3.2.1 and Section 4.3.2.2, respectively.

- *Verification of IDP.* Formally, given a CTMC $\mathcal{C}$ and an LDP $\Phi = \int 1 \leqslant T \rightarrow \varphi$, compute the probability of the set of infinite timed paths of $\mathcal{C}$ satisfying Φ under the *infinitary satisfaction condition*, see Definition 3.2.8. We also have two cases, i.e., the time-bounded case and unbounded case, which we denote by $\mathit{Prob}(\mathcal{C} \models^{\star} \Phi)$ and $\mathit{Prob}(\mathcal{C} \models^{\star} \varphi)$, respectively. The algorithms for these two cases are given in Section 4.3.3.2 and Section 4.3.3.1, respectively.

We start in Section 4.3.1 where we establish a link between the EDP of CTMC and the model of *Markovian reward model* (MRM). In Section 4.3.2 we address the verification of EDP over CTMCs. More precisely, in Section 4.3.2.1 we present an algorithm which solves the time-bounded verification problem for EDP over CTMCs, as well as giving error bounds and complexity results. In Section 4.3.2.2 we tackle the time-unbounded EDP verification problem. We conclude the chapter with Section 4.3.3, where we present a solution to the model checking problem for IDP over CTMCs, again addressing error bounds and complexity results.

4.3.1 Relationship to MRMs

In this section, we establish a link between the EDP of CTMC and the model of MRM. We start with some definitions.

Definition 4.3.1 (MRM) *A (labelled) Markovian reward model $\mathcal{M}$ is a pair $(\mathcal{C}, \mathbf{r})$, where $\mathcal{C}$ is* CTMC *and $\mathbf{r} : S \to \mathbb{R}^d$ is a reward structure which assigns to each state $s \in S$ a vector of rewards $(r_1(s), \cdots, r_d(s))$.*

Remark 4.3.1 *The MRM defined in Definition 4.3.1 is more general than the one in [BHHK00], in the sense that we have multiple reward structures, and, more importantly, we allow arbitrary (instead of nonnegative) rewards associated with the states.*

In [BHHK00] the authors study the model checking problem for the logic CSRL (introduced in [BHHK00]) and MRM models. The problem that [BHHK00] addresses is the *reward bounded* verification problem (which we extend to the multiple-reward setting, conforming to Definition 4.3.1), namely: given a set of goal states G and a vector of reward bounds M_j, compute the probability of the paths which reach G and in which the j-th accumulated reward does not exceed M_j for each j. Below we show that this problem is essentially the same as EDP for CTMCs.

Let $\mathcal{C}$ be a CTMC and $\varphi = \bigwedge_{j \in J} \left(\sum_{k \in K_j} c_{jk} \int \mathrm{sf}_{jk} \leqslant M_j \right)$ be an LDP, where $c_{jk}, M_j \in \mathbb{R}$, sf_{jk} are state formulas, and J, K_j for $j \in J$ are finite index sets. We show how to construct an MRM $\mathcal{C}[\varphi]$ that can be used to check whether φ is satisfied in $\mathcal{C}$ or not.

For every state $s_i \in S$, we define

$$r_{ji} = \sum_{\substack{t \in K_j, \\ s_i \models \mathrm{sf}_{jt}}} c_{jt}$$

for all $j \in J$. This yields a multiple reward structure $\mathbf{r}$ with $\mathbf{r}(s_i) = (r_{0i}, \cdots, r_{(|J|-1)i})$. Hence $\mathcal{C}[\varphi] = (\mathcal{C}, \mathbf{r})$. It is straightforward to see that the constraint expressed by LDP can be alternatively formulated as the "reward-bounded" constraint for MRMs, since $\sum_{k \in K_j} c_{jk}$

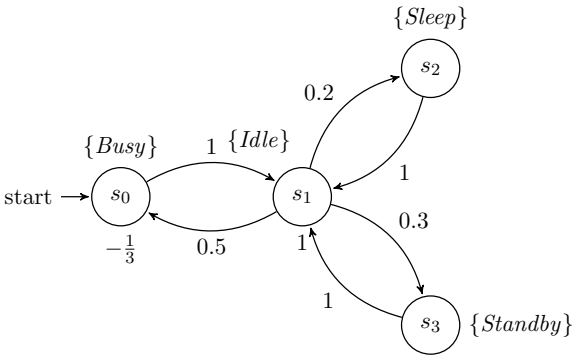


Figure 4.6: An example MRM

$\int \mathrm{sf}_{jk}$ essentially denotes the accumulated rewards along a finite timed path, and hence each M_j can be regarded as the bound of the reward.

On the other hand, given an MRM and a vector of reward bounds M_j for each reward structure, we construct an LDP φ as

$$\bigwedge_{j \in J} \sum_{s \in S} r_j(s) \int @s \leqslant M_j,$$

where $@s$ is an atomic proposition which holds exactly at state s. Hence, the reward-bounded verification problem for MRMs can be encoded into the verification of Linear Duration Properties in CTMCs.

It is straightforward to see that this correspondence, stated in the (time) unbounded case, can be adapted to the time-bounded case without any difficulties.

We illustrate the relationship between LDPs and MRMs with an example.

Example 4.3.1 *Consider the* CTMC $\mathcal{C}$ *in Figure 3.2 and the LDP example formula* $\varphi = \int \mathit{Idle} - \frac{1}{3} \int \mathit{Busy} \leqslant 0$ *introduced in Example 3.2.4. Moreover, suppose that we are interested in checking the validity of* φ *in* $\mathcal{C}$ *with a time bound of* $T = 10$ *time units.*

Now consider the MRM model in Figure 4.6, which closely resembles the CTMC $\mathcal{C}$ *in Figure 3.2. In the MRM model in Figure 4.6 the number below a given state is the reward associated with that state. We omit zero rewards for simplicity. In the transformation, we have assigned the coefficients of the LDP formula* φ *as rewards to the states.*

It should be clear that the problem of model checking the LDP formula $\varphi = \int \mathit{Idle} - \frac{1}{3} \int \mathit{Busy} \leqslant 0$ *in the* CTMC *in Figure 3.2 is equivalent to performing transient analysis of the MRM model in Figure 4.6 and checking whether the reward accumulated up to time* T *is positive or negative. If the accumulated reward is negative, the formula* φ *is satisfied in the* $\mathcal{C}$ *of Figure 3.2. The opposite is true if the reward is positive.*

4.3.2 Verification of EDP

In this section, we show how to verify EDP formulas. Throughout this section, we fix a CTMC $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$, a set of goal states $G \subseteq S$, and an LDP

$$\Phi = \int 1 \leqslant T \rightarrow \underbrace{\bigwedge_{j \in J} \Big(\sum_{k \in K_j} c_{jk} \int \mathrm{sf}_{jk} \leqslant M_j \Big)}_{\varphi}.$$

4.3.2.1 Time-bounded verification of EDP

Our task is to compute $\mathit{Prob}(\mathcal{C} \models^G \Phi)$. In words, we want to compute the probability of the set of infinite timed paths ρ of $\mathcal{C}$ that reach the set of goal states G and satisfy Φ. First observe that the probability that we wish to calculate, $\mathit{Prob}(\mathcal{C} \models^G \Phi)$, can be decomposed into three different terms (see Proposition 4.3.1). Thus, instead of computing $\mathit{Prob}(\mathcal{C} \models^G \Phi)$, one can compute the probability of all the paths that reach the set of goal states G and from this subtract the probability of all the paths that reach G in less than T time units. Basically, we are left with the probability of all the timed paths that reach the set of goal states G in more than T time units. The last step consists in adding to the latter probability the probability of all the timed paths that satisfy φ in $[0, T]$ and reach the set of goal states G, namely $\mathit{Prob}(\mathcal{C} \models^G_T \varphi)$. The reason why we need to keep the probability of all the timed paths that reach the set of goal states G in more than T time units might seem counter-intuitive at first. However, note that the premise of the formula $\Phi = \int 1 \leqslant T \rightarrow \bigwedge_{j \in J} \big(\sum_{k \in K_j} c_{jk} \int \mathrm{sf}_{jk} \leqslant M_j \big)$ becomes false if we wait more than T time units, making the whole formula Φ true. Thus, one has to consider also those paths that reach G in more than T time units.

Proposition 4.3.1 proves formally the decomposition of $\mathit{Prob}(\mathcal{C} \models^G \Phi)$ into the three terms that we have mentioned above, i.e., $\Pr(\Diamond G)$, $\Pr(\Diamond^{\leqslant T} G)$ and $\mathit{Prob}(\mathcal{C} \models^G_T \varphi)$.

Proposition 4.3.1 *Given a* CTMC $\mathcal{C}$ *and an LDP* Φ*, we have:*

$$\mathit{Prob}(\mathcal{C} \models^G \Phi) = \Pr(\Diamond G) - \Pr(\Diamond^{\leqslant T} G) + \mathit{Prob}(\mathcal{C} \models^G_T \varphi).$$

Proof We have that

$$\begin{aligned} \mathit{Prob}(\mathcal{C} \models^G \Phi) &= \Pr(\{\rho \in \mathit{Paths}^{\mathcal{C}} \mid \rho \models^G \Phi\}) \\ &= \Pr\left(\left\{\rho \in \mathit{Paths}^{\mathcal{C}} \mid \rho \models^G \neg\left(\int 1 \leqslant T\right) \vee \varphi\right\}\right), \end{aligned}$$

where $\varphi = \bigwedge_{j \in J} \big(\sum_{k \in K_j} c_{jk} \int \mathrm{sf}_{jk} \leqslant M_j \big)$. We know that

$$\neg\left(\int 1 \leqslant T\right) \vee \varphi = \neg\left(\int 1 \leqslant T\right) \vee \left(\varphi \wedge \int 1 \leqslant T\right).$$

Therefore, we have

$$\begin{aligned}
Prob(\mathcal{C} \models^{G} \Phi) &= \Pr\left(\left\{\rho \in Paths^{\mathcal{C}} \middle| \rho \models^{G} \neg\left(\int 1 \leqslant T\right) \vee \left(\varphi \wedge \int 1 \leqslant T\right)\right\}\right) \\
&= \Pr\left(\left\{\rho \in Paths^{\mathcal{C}} \middle| \rho \models^{G} \neg\left(\int 1 \leqslant T\right) \vee \left(\rho \models^{G} \varphi \wedge \int 1 \leqslant T\right)\right\}\right) \\
&= \Pr\left(\left\{\rho \in Paths^{\mathcal{C}} \middle| \rho \models^{G} \neg\left(\int 1 \leqslant T\right)\right\}\right) \\
&\quad + \Pr\left(\left\{\rho \in Paths^{\mathcal{C}} \middle| \rho \models^{G} \varphi \wedge \int 1 \leqslant T\right\}\right) \\
&= \Pr(\Diamond G) - \Pr(\Diamond^{\leqslant T} G) + Prob(\mathcal{C} \models^{G}_{T} \varphi).
\end{aligned}$$

This completes the proof. ■

Recall that $\Pr(\Diamond G)$ and $\Pr(\Diamond^{\leqslant T} G)$ can be easily computed (cf. Definition 3.1.5). Hence, our task is now to calculate:

$$Prob(\mathcal{C} \models^{G}_{T} \varphi) := \Pr(\{\rho \mid \rho \models^{G}_{T} \varphi\}),$$

i.e., the probability of the set of paths of the CTMC $\mathcal{C}$ which reach G in time interval $[0, T]$ and satisfy the LDP φ before that happens; see Definition 3.2.8.

PDE and Integral Equation Formulations

In order to compute $Prob(\mathcal{C} \models^{G}_{T} \varphi)$, we shall use the link to MRMs established in Section 4.3.1. Essentially, we will reduce the problem of checking φ in $\mathcal{C}$ to the problem of accumulated reward on a MRM. The claim is formally stated in Theorem 4.3.2.

Recall that $\mathcal{C}[\varphi]$ is the MRM obtained from $\mathcal{C}$ and φ. We need an extra transformation over $\mathcal{C}[\varphi]$, namely, making each state $s \in G$ absorbing and setting $\mathbf{r}(s) = (0, \cdots, 0)$ (i.e., the rewards associated with s are all 0). We denote the resulting MRM $\mathcal{C}[\varphi, G]$. Recall that $X(t)$ is the underlying stochastic process of the CTMC $\mathcal{C}$. We denote by $\mathbf{Y}(T)$ the vector of accumulated rewards in the MRM $\mathcal{C}[\varphi]$ (see Section 4.3.1) up to time T, i.e.,

$$\mathbf{Y}(T) = (Y_0(T), \ldots, Y_{|J|-1}(T)) = \int_0^T \mathbf{r}(X(\tau))d\tau$$

and thus each $Y_j(T)$ $(j \in J)$ corresponds to a reward structure in $\mathcal{C}$. The vector of stochastic processes $\mathbf{Y}(T)$ is fully determined by $X(T)$ and the vector of reward structures of the state s_i is $\mathbf{r}(s_i) = (r_{0i}, \ldots, r_{(|J|-1)i})$.

Define $\mathbf{F}(T, \mathbf{y})$ to be the matrix of the joint probability distribution of state and rewards with entries $\mathbf{F}(T, \mathbf{y})[s, s'] = F_s^{s'}(T, \mathbf{y})$ for $s, s' \in S$ and

$$F_s^{s'}(T, \mathbf{y}) = \Pr\left(\left\{X(T) = s', \bigwedge_{j \in J} Y_j(T) \leqslant y_j \mid X(0) = s\right\}\right),$$

where $\mathbf{y} = (y_0, \cdots, y_{|J|-1})$. Note that we define $\mathbf{F}(T, \mathbf{y})$ over the induced MRM $\mathcal{C}[\varphi, G]$.

Next, Theorem 4.3.2 proves that the problem of computing the probability of the set of timed paths that satisfy φ in $\mathcal{C}$ in $[0, T]$ is equivalent of computing the probability of accumulated reward in the induced MRM $\mathcal{C}[\varphi, G]$.

Theorem 4.3.2 *Given a* CTMC $\mathcal{C}$*, an LDP* φ*, a vector* $\mathbf{M} = (M_0, \ldots, M_{|J|-1})$*, where each* M_j *is defined as in* φ *(cf. Equation (3.2)), and a set of goal states* G*, we obtain the induced* MRM $\mathcal{C}[\varphi, G]$*, and we have:*

$$Prob(\mathcal{C} \models_T^G \varphi) = \sum_{s \in S} \sum_{s' \in G} \alpha(s) F_s^{s'}(T, \mathbf{M}). \tag{4.21}$$

Proof Let $s' \in G$ be an absorbing state with $\mathbf{r}(s) = (0, \cdots, 0)$. The probability to be in s' at time T is the same as the probability to reach s' before T (see [BHHK03]). Therefore, we have that:

$$\Pr(\{\rho \in Paths^{\mathcal{C}}(s) \mid \rho \models_T^{\{s'\}} \varphi\}) = \Pr\left(\left\{ X(T) = s', \bigwedge_{j \in J} Y_j(T) \leqslant M_j \mid X(0) = s \right\}\right),$$

which directly follows from the construction in Section 4.3.1. ■

Theorem 4.3.2 suggests a reduction to $\mathbf{F}(t, \mathbf{y})$, which we now characterise in terms of a system of PDEs.

Theorem 4.3.3 *For an* MRM $\mathcal{C}[\varphi, G]$ *the function* $\mathbf{F}(t, \mathbf{y})$ *is given by the following system of PDEs:*

$$\frac{\partial \mathbf{F}(t, \mathbf{y})}{\partial t} + \sum_{j \in J} \mathbf{D}_j \cdot \frac{\partial \mathbf{F}(t, \mathbf{y})}{\partial y_j} = \mathbf{Q} \cdot \mathbf{F}(t, \mathbf{y}), \tag{4.22}$$

where $\mathbf{D}_j$ *is a diagonal matrix such that* $\mathbf{D}_j(s, s) = r_j(s)$.

Proof We want to calculate $F_s^{s'}(t, \mathbf{y})$. Assume that we are in state z at time Δt, for some small Δt. We consider three possible scenarios, and calculate the probability of each of them:

- No jumps before Δt;
- One jump before Δt;
- More than one jump before Δt.

No jumps before Δt.

The probability of this scenario is:

$$(1 + Q(s,s)\Delta t) \cdot F_s^{s'}(t, \mathbf{y} - \mathbf{r}(s)\Delta t).$$

Here we indicate with $\mathbf{y} - \mathbf{r}(s)\Delta t$ the vector operation resulting in:

$$\mathbf{y} - \mathbf{r}(s)\Delta t = (y_0 - r_0(s)\Delta t, \ldots, y_{|J|-1} - r_{|J|-1}(s)\Delta t).$$

One jump before Δt.

We denote the probability of being in state z at time Δt by $g_z(\Delta t)$. In order to derive the probability of this scenario we split it into three different cases:

1. All rewards positive. Let

   $$\mathbf{r}_{max} = (\max_{s\in S}\{r_0(s)\}, \ldots, \max_{s\in S}\{r_{|J|-1}(s)\})$$

   and

   $$\mathbf{r}_{min} = (\min_{s\in S}\{r_0(s)\}, \ldots, \min_{s\in S}\{r_{|J|-1}(s)\}).$$

   The accumulated reward in Δt is at most $\mathbf{r}_{max}\Delta t$ and at least $\mathbf{r}_{min}\Delta t$. It follows that:

   $$Q(s,z)\Delta t \cdot F_z^{s'}(t, \mathbf{y} - \mathbf{r}_{max}\Delta t) \leqslant g_z(\Delta t) \leqslant Q(s,z)\Delta t \cdot F_z^{s'}(t, \mathbf{y} - \mathbf{r}_{min}\Delta t).$$

2. All rewards negative. Let

   $$\mathbf{r}_{max} = (\max_{s\in S}\{|r_0(s)|\}, \ldots, \max_{s\in S}\{|r_{|J|-1}(s)|\})$$

   and

   $$\mathbf{r}_{min} = (\min_{s\in S}\{|r_0(s)|\}, \ldots, \min_{s\in S}\{|r_{|J|-1}(s)|\}).$$

   It follows that:

   $$Q(s,z)\Delta t \cdot F_z^{s'}(t, \mathbf{y} - \mathbf{r}_{max}\Delta t) \leqslant g_z(\Delta t) \leqslant Q(s,z)\Delta t \cdot F_z^{s'}(t, \mathbf{y} - \mathbf{r}_{min}\Delta t).$$

3. Mixed rewards. Let

   $$\mathbf{r}_{max} = (\max_{s\in S}\{r_0(s)|r_0(s) \geq 0\}, \ldots, \max_{s\in S}\{r_{|J|-1}(s)|r_{|J|-1}(s) \geq 0\})$$

   and

   $$\mathbf{r}_{min} = (\min_{s\in S}\{r_0(s)|r_0(s) < 0\}, \ldots, \min_{s\in S}\{r_{|J|-1}(s)|r_{|J|-1}(s) < 0\}).$$

   It follows that:

   $$Q(s,z)\Delta t \cdot F_z^{s'}(t, \mathbf{y} - \mathbf{r}_{max}\Delta t) \leqslant g_z(\Delta t) \leqslant Q(s,z)\Delta t \cdot F_z^{s'}(t, \mathbf{y} - \mathbf{r}_{min}\Delta t).$$

In all three cases above, note that

$$\lim_{\Delta t \to 0} \frac{g_z(\Delta t)}{\Delta t} = Q(s,z) F_z^{s'}(t, \mathbf{y}).$$

More than one jump before Δt.

The probability of this scenario is negligible i.e., $o(\Delta t)$. Note that $\lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t} = 0$.

The joint distribution is given by

$$F_s^{s'}(t + \Delta t, \mathbf{y}) = (1 + Q(s,s)\Delta t) \cdot F_s^{s'}(t, \mathbf{y} - \mathbf{r}(s)\Delta t) + \sum_{z \neq s} g_z(\Delta t) + o(\Delta t).$$

From here on we derive the equations for $F_s^{s'}(\cdot)$ only for nonzero rewards. Let $|\mathbf{y}| = |J|$ be the cardinality of $\mathbf{y}$. We rewrite $F_s^{s'}(t, \mathbf{y})$ as $F_s^{s'}(t, y_0, \ldots, y_{|J|-1})$ to ease the notation and proofs. Given the above notation we can add and subtract terms from the joint distribution of $X(t)$ and $\mathbf{Y}(t)$ as follows:

$$\begin{aligned} F_s^{s'}(t + \Delta t, \mathbf{y}) &= F_s^{s'}(t, \mathbf{y} - \mathbf{r}(s)\Delta t) + Q(s,s)\Delta t \cdot F_s^{s'}(t, \mathbf{y} - \mathbf{r}(s)\Delta t) + \sum_{z \neq s} g_z(\Delta t) + o(\Delta t) \\ &= \Big(F_s^{s'}(t, \mathbf{y}) - F_s^{s'}(t, \mathbf{y})\Big) + F_s^{s'}(t, \mathbf{y} - \mathbf{r}(s)\Delta t) + Q(s,s)\Delta t \cdot F_s^{s'}(t, \mathbf{y} - \mathbf{r}(s)\Delta t) \\ &\quad + \sum_{z \neq s} g_z(\Delta t) + o(\Delta t) \end{aligned}$$

Let $\widehat{\mathbf{D}}(s)$ be a diagonal matrix such that $\widehat{\mathbf{D}}(s)[i,i] = r_i(s)$, for all $i \leq |J| - 1$ such that $r_i(s) \neq 0$. Note that $\widehat{\mathbf{D}}(s)$ is invertible. We observe that

$$\begin{aligned} &F_s^{s'}(t + \Delta t, \mathbf{y}) - F_s^{s'}(t, \mathbf{y}) \\ &= F_s^{s'}(t, \mathbf{y} - \mathbf{r}(s)\Delta t) - F_s^{s'}(t, \mathbf{y}) + Q(s,s)\Delta t \cdot F_s^{s'}(t, \mathbf{y} - \mathbf{r}(s)\Delta t) \\ &\quad + \sum_{z \neq s} g_z(\Delta t) + o(\Delta t) \\ &= \widehat{\mathbf{D}}(s)^{-1} \cdot \widehat{\mathbf{D}}(s) \Big(F_s^{s'}(t, \mathbf{y} - \mathbf{r}(s)\Delta t) - F_s^{s'}(t, \mathbf{y})\Big) + Q(s,s)\Delta t \cdot F_s^{s'}(t, \mathbf{y} - \mathbf{r}(s)\Delta t) \\ &\quad + \sum_{z \neq s} g_z(\Delta t) + o(\Delta t), \end{aligned}$$

and

$$\begin{aligned} &\frac{F_s^{s'}(t + \Delta t, \mathbf{y}) - F_s^{s'}(t, \mathbf{y})}{\Delta t} \\ &= \widehat{\mathbf{D}}(s)^{-1} \cdot \widehat{\mathbf{D}}(s) \left(\frac{F_s^{s'}(t, \mathbf{y} - \mathbf{r}(s)\Delta t) - F_s^{s'}(t, \mathbf{y})}{\Delta t}\right) + Q(s,s) \cdot F_s^{s'}(t, \mathbf{y} - \mathbf{r}(s)\Delta t) \\ &\quad + \sum_{z \neq s} \frac{g_z(\Delta t)}{\Delta t} + o(\Delta t). \end{aligned}$$

Notice that all the three cases result in the same outcome. Taking the limit $\lim_{\Delta t \to 0}$ and renaming the variables we obtain that

$$\frac{\partial F_s^{s'}(t,\mathbf{y})}{\partial t} + \sum_{j\in J} r_j(s)\frac{\partial F_s^{s'}(t,\mathbf{y})}{\partial y_j} = \sum_{z\in S} Q(s,z)F_z^{s'}(t,\mathbf{y}).$$

In matrix notation, one has

$$\frac{\partial \mathbf{F}(t,\mathbf{y})}{\partial t} + \sum_{j\in J} \mathbf{D}_j \cdot \frac{\partial \mathbf{F}(t,\mathbf{y})}{\partial y_j} = \mathbf{Q}\cdot\mathbf{F}(t,\mathbf{y}),$$

which completes the proof. ■

Remark 4.3.2 *The system of PDEs from Theorem 4.3.3 is a special case of the system of PDEs given in [HKNT98, GT07], which is presented for stochastic Petri nets. We remark here that efficient solutions of system of PDEs exist only when the PDEs have dimension less or equal than four. In any other case, the system of PDEs cannot usually be solved and one should resort to sampling techniques such as Monte Carlo.*

Example 4.3.2 *For the* CTMC *depicted in Figure 3.2, with* $r(s_0) = 1$ *and* $r(s_1) = -1$, *we can derive the following system of PDEs:*

$$\begin{aligned}
\frac{\partial F_{s_0}^{s_1}(t,y)}{\partial t} + \frac{\partial F_{s_0}^{s_1}(t,y)}{\partial y} &= 10F_{s_1}^{s_1}(t,y) - 10F_{s_0}^{s_1}(t,y),\\
\frac{\partial F_{s_1}^{s_0}(t,y)}{\partial t} - \frac{\partial F_{s_1}^{s_0}(t,y)}{\partial y} &= -6F_{s_1}^{s_0}(t,y) + 3F_{s_0}^{s_0}(t,y)\\
&\quad +1.2F_{s_2}^{s_0}(t,y) + 1.8F_{s_3}^{s_0}(t,y).
\end{aligned}$$

Note that trivial equations like $0 = 0$ *are simply omitted.*

Next we provide an alternative characterisation of the joint probability distribution in terms of a system of integral equations, as follows.

Theorem 4.3.4 *The solution of the system of PDEs in Equation (4.22) is the least fixpoint of the following system of integral equations:*

$$F_s^{s'}(t,\mathbf{y}) = e^{\mathbf{Q}(s,s)t}F_s^{s'}(0,\mathbf{y}-\mathbf{r}(s)t) + \int_0^t \sum_{z\neq s} e^{\mathbf{Q}(s,s)x}\mathbf{Q}(s,z)F_z^{s'}(t-x,\mathbf{y}-\mathbf{r}(s)x)dx.$$

Proof One possible solution for the hyperbolic system of PDEs obtained is the method of characteristics proposed in [Pat93]. The method consists in finding the characteristic curves $\mathbf{y}(t)$ on which PDEs reduce to ODEs. Let $y(t)$ be an arbitrary curve and consider the derivative of $F_s^{s'}(t,\mathbf{y}(t))$ in t. More specifically,

$$\frac{dF_s^{s'}(t,\mathbf{y}(t))}{dt} = \frac{\partial F_s^{s'}(t,\mathbf{y}(t))}{\partial t}\frac{dt}{dt} + \frac{\partial F_s^{s'}(t,\mathbf{y}(t))}{\partial y}\frac{d\mathbf{y}(t)}{dt}.$$

Note that $\frac{dt}{dt} = 1$, then considering those curves $\mathbf{y}(t)$ such that $\frac{d\mathbf{y}(t)}{dt} = \mathbf{r}(s)$ yields

$$\frac{dF_s^{s'}(t,\mathbf{y}(t))}{dt} = \frac{\partial F_s^{s'}(t,\mathbf{y}(t))}{\partial t} + \sum_{j\in J} \frac{\partial F_s^{s'}(t,\mathbf{y}(t))}{\partial y_j} r_j(s) \tag{4.23}$$

Note here that the right-hand side of Equation (4.23) is the left-hand side of Equation (4.22), which implies that:

$$\frac{dF_s^{s'}(t,\mathbf{y}(t))}{dt} = \sum_{z\in S} Q(s,z) F_z^{s'}(t,\mathbf{y}(t)). \tag{4.24}$$

Equation (4.24) defines a system of ordinary differential equations that can be solved if we fix an initial value for $F_s^{s'}(0,\mathbf{y}(0))$. The solution is given by:

$$F_s^{s'}(t,\mathbf{y}(t)) = e^{Q(s,s)t}\left[\int_0^t e^{-Q(s,s)x} \sum_{z\neq s} Q(s,z) F_z^{s'}(x,\mathbf{y}(x))dx + F_s^{s'}(0,\mathbf{y}(0))\right]. \tag{4.25}$$

The curve $\mathbf{y}(t)$ defined by the ODE $\frac{d\mathbf{y}(t)}{dt} = \mathbf{r}(s)$ has as solution:

$$\mathbf{y}(t) = \mathbf{r}(s)t + \mathbf{C}.$$

We can calculate the value of $\mathbf{C}$, given a time t^* and the value $\mathbf{y}^*$ of the accumulated reward, by

$$\mathbf{C} = \mathbf{y}^* - \mathbf{r}(s)t^*.$$

In order to find the solution for the PDE in Equation (4.22) at a given t^* and $\mathbf{y}^*$, we solve the ODE in Equation (4.24) on the curve given by

$$\mathbf{y}(t) = \mathbf{r}(s)t + \mathbf{y}^* - \mathbf{r}(s)t^* = \mathbf{y}^* - \mathbf{r}(s)(t^* - t),$$

and more specifically, by substituting $x = t^* - x$:

$$F_s^{s'}(t^*,\mathbf{y}^*) = e^{Q(s,s)t^*} F_s^{s'}(0,\mathbf{y}^* - \mathbf{r}(s)t^*) + \int_0^{t^*} \sum_{z\neq s} e^{Q(s,s)x} Q(s,z) F_z^{s'}(t^* - x, \mathbf{y}^* - \mathbf{r}(s)x)dx.$$

This completes the proof. ■

For readers who are familiar with *Piecewise-deterministic Markov processes* (PDMPs) [Dav93], Equation (4.22) can also be obtained as follows. For every state s of the CTMC we assign the system of differential equations: for each $j \in J$,

$$\frac{dx_j(t)}{dt} = r_j(s), \quad x_j(t) \in \mathbb{R}.$$

Note that $x_j(t)$ will denote the total accumulated reward at time t for the reward structure j. This results in a PDMP with the state space $S \times \mathbb{R}^{|J|}$. The function $F_s^{s'}(t,\mathbf{y})$ represents the probability to reach the set of states $\{s'\} \times (-\infty, y_0] \times \cdots \times (-\infty, y_{|J|-1}]$ at time t.

Theorem 4.3.3 and Theorem 4.3.4 imply that, to solve the bounded-time EDP verification problem, we need to solve (first-order) PDEs or integral equations. However, this is usually costly and numerically unstable [Hig02]. We present solutions in the next section, based on uniformisation.

Uniformisation

We present a uniformisation-based algorithm to compute $F_s^{s'}(t, \mathbf{y})$. The *uniformisation* method [Jen53] involves transforming the CTMC $\mathcal{C}$ into a behaviorally equivalent DTMC $\mathcal{D}$. The state space and initial distribution of $\mathcal{D}$ are the same as for $\mathcal{C}$. The probability matrix $\widehat{\mathbf{P}}$ of $\mathcal{D}$ is constructed by $\widehat{\mathbf{P}} = \mathbf{I} + \frac{1}{\Lambda}\mathbf{Q}$, where Λ is the maximal exit rate of $\mathcal{C}$. We obtain:

$$\pi(t) = e^{(\widehat{\mathbf{P}}-\mathbf{I})\Lambda t} = \sum_{n=0}^{\infty} \widehat{\mathbf{P}}^n \frac{(\Lambda t)^n}{n!} e^{-\Lambda t}, \tag{4.26}$$

where with $\pi(t)$ we indicate the transient probability distribution vector.

We now apply the uniformisation technique to efficiently compute $F_s^{s'}(t, \mathbf{y})$. First, we note that the infinite sum in Equation (4.26) is equal to the probability $\frac{(\Lambda t)^n}{n!} e^{-\Lambda t}$ that exactly n Poisson arrivals occur in an interval of time $[0, t]$ multiplied with the probability $\widehat{\mathbf{P}}^n$ to take the state transitions corresponding to the arrivals. Then using Equation (4.26) we obtain:

$$F_s^{s'}(t, \mathbf{y}) = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \cdot \left(\sum_{|\varsigma|=n} \Pr(\{\varsigma \mid X(0) = s\}) \cdot \Pr\left(\{X(n) = s', \mathbf{Y}(t) \leqslant \mathbf{y} \mid \varsigma\}\right) \right),$$

where for a given path $\varsigma = s \to s_1 \to \cdots \to s_{n-1} \to s_n$,

$$\mathit{Prob}(\varsigma) := \Pr\left(\{\varsigma \mid X(0) = s\}\right) = \widehat{\mathbf{P}}(s, s_1) \times \cdots \times \widehat{\mathbf{P}}(s_{n-1}, s_n).$$

If $|\varsigma| = 0$ then $\mathit{Prob}(\varsigma) := 1$. $\Pr\left(\{X(n) = s', \mathbf{Y}(t) \leqslant \mathbf{y} \mid \varsigma\}\right)$ denotes the *conditional probability* that given the path ς at step n the state is s' and the total accumulated reward until time t is less than $\mathbf{y}$. The above equation can also be written as:

$$F_s^{s'}(t, \mathbf{y}) = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \sum_{\substack{|\varsigma|=n,\\ \varsigma[0]=s,\\ \varsigma[n]=s'}} \mathit{Prob}(\varsigma) \cdot \Pr\left(\{\mathbf{Y}(t) \leqslant \mathbf{y} \mid \varsigma\}\right). \tag{4.27}$$

Note that

$$\mathit{Prob}(\varsigma) \cdot \Pr\left(\{\mathbf{Y}(t) \leqslant \mathbf{y} \mid \varsigma\}\right) = \Pr\left(\{\mathbf{Y}(t) \leqslant \mathbf{y} \wedge \varsigma\}\right). \tag{4.28}$$

Now the task is to compute $\Pr\left(\{\mathbf{Y}(t) \leqslant \mathbf{y} \wedge \varsigma\}\right)$, for which we resort to the computation of an integration over a convex polytope. The basic idea is to generate timed constraints over variables determining the residence time of each state along ς to make $\mathbf{Y}(t) \leqslant \mathbf{y}$ hold. The desired probability can thus be formulated as a multidimensional integral, which can be computed by the efficient algorithm given in [LZ01].

Given a *discrete* finite path ς of length k, an LDP φ, and a time bound T, we define the set of linear constraints $\mathcal{S}$ generated in Algorithm 6.

In Algorithm 6, line 3 generates the set of constraints from each conjunct in formula φ. In line 5 we add one more constraint to ensure that in the interval of time $[0, T]$ we will reach the last state of ς.

Algorithm 6 Generate a set of linear constraints $\mathcal{S}$ induced by φ, ς and T

Require: LDP φ, a path ς of length k and a time bound T

Ensure: A set of linear constraints $\mathcal{S}$

1: $\mathcal{S} = \varnothing$

2: **for** $j \in J$ **do**

3: $\quad \mathcal{S} = \mathcal{S} \cup \left\{ \sum_{i \in K_j} c_{ji} \cdot \sum_{\substack{0 \leq \ell \leq k, \\ \varsigma[\ell] \models \mathrm{sf}_{ji}}} x_\ell \leqslant M_j \right\}$

4: **end for**

5: $\mathcal{S} = \mathcal{S} \cup \left\{ \sum_{i=0}^{k-1} x_i \leq T \right\} \cup \left\{ \sum_{i=0}^{k} x_i \geq T \right\}$

6: $\mathcal{S} = \mathcal{S} \cup \{x_i > 0\}$ for all x_i

7: **return** $\mathcal{S}$

We show the functioning of Algorithm 6 with the following example.

Example 4.3.3 *Assume the LDP* $\varphi = \int \mathit{Idle} - \frac{1}{3} \int \mathit{Busy} \leqslant 0 \wedge \int \mathit{Idle} - \frac{1}{4} \int \mathit{Sleep} \leqslant 0$*, the discrete path* $\varsigma = s_0 \to s_1 \to s_2 \to s_1 \to s_3$ *and the time bound* $T = 6$*. The set of linear constraints* $\mathcal{S}$ *generated by Algorithm 6 induced by* ς*,* φ *and* T *is:*

$$\mathcal{S} = \begin{cases} -\frac{1}{3} \cdot x_0 + x_1 + 0 \cdot x_2 + x_3 \leqslant 0 \\ 0 \cdot x_0 + x_1 - \frac{1}{4} \cdot x_2 + x_3 \leqslant 0 \\ x_0 + x_1 + x_2 + x_3 \leq 6 \\ x_0, x_1, x_2, x_3 > 0 \end{cases}$$

Similarly to Section 4.1, we need to prove that Algorithm 6 is correct, i.e., that Algorithm 6 returns the right set of linear inequalities $\mathcal{S}$. In order for $\mathcal{S}$ to be correct we need that any solution of the system must satisfy the formula φ when plugged back into the discrete path ς. Moreover, if there is a sequence of time jumps for ς that satisfies φ then that sequence must be a solution to $\mathcal{S}$. Lemma 4.3.5 proves the correctness of the system of linear inequalities returned by Algorithm 6 .

Lemma 4.3.5 *Assume a discrete path* ς *of the* CTMC $\mathcal{C}$*, an LDP* φ *and a time bound* T*. Let* $\mathcal{S}$ *be the set of linear constraints obtained by Algorithm 6. Then*

$$\varsigma[x_0, \ldots, x_{k-1}] \models \left(\varphi \wedge \int 1 \leq T \right) \quad \textit{iff} \quad (x_0, \ldots, x_{k-1}) \textit{ satisfies the constraints in } \mathcal{S}.$$

Proof Let φ_j be the j-th conjunct of φ. It is easy to see that:

$$\varsigma[x_0, \ldots, x_{k-1}] \models \varphi_j \quad \text{iff} \quad (x_0, \ldots, x_{k-1}) \text{ satisfies the constraints in } \mathcal{S},$$

which follows from the definition of $\models$ (see Definition 3.2.7). Note that $\varsigma[x_0, \ldots, x_{k-1}] \models \int 1 \leq t$ iff $\sum_{i=0}^{k-1} x_i \leq t$ (see Definition 3.2.8), which proves the lemma. ■

We define

$$Prob(\varsigma[\mathcal{S}]) := \Pr^{\mathcal{C}(\varsigma[0])}(\{\varsigma[x_0,\ldots,x_{k-1}] \mid (x_0,\ldots,x_{k-1}) \text{ satisfies the constraints in } \mathcal{S}\}).$$

For future use, declare the function $Volume\_int(\alpha,\varsigma,\mathcal{S})$ which, given an initial distribution α, a finite discrete path $\varsigma = s_0 \to \cdots \to s_k$ of length k and a set of linear constraints $\mathcal{S}$ over $x_0,\cdots,x_{k-1}$, returns

$$\alpha(s_0)\cdot\prod_{i=0}^{k-1} E(s_i)\cdot P(s_i,s_{i+1})\cdot \underbrace{\int\cdots\int}_{k}_{\mathcal{S}} \prod_{i=0}^{k-1} e^{-E(s_i)\tau_i} d\tau_i. \tag{4.29}$$

Evidently, $Prob(\varsigma[\mathcal{S}])$ is equal to $Volume\_int(\alpha,\varsigma,\mathcal{S})$. In [LZ01] an algorithm based on the Laplace transform is proposed to compute certain multidimensional integrals over polytopes. In Equation 4.29, the integration is over $\mathcal{S}$, which is the intersection of hyperplanes (in terms of linear inequalities). Hence, the algorithm of [LZ01] can be applied directly. The time complexity of solving the multidimensional integral is $\mathcal{O}(|J|^k)$. Recall that $|J|$ is the number of constraints and k is the number of variables. Note that we omit the simple constraints from Algorithm 6, line 5 and 6, when computing the complexity of the algorithm. The simple constraints denote a constant term in the overall complexity and can safely be removed.

Remark 4.3.3 *The reader should note that we have again reduced the problem to computing certain multidimensional integrals over polytopes, as in Section 4.1 and Section 4.2.*

The following theorem concludes this section, showing that, in order to compute $\Pr(\{\mathbf{Y}(t) \leqslant \mathbf{y} \wedge \varsigma\})$, one only needs to compute $Prob(\varsigma[\mathcal{S}])$, where $\mathcal{S}$ is generated from Algorithm 6.

Theorem 4.3.6 *Let ς be a discrete path of the* CTMC $\mathcal{C}$ *ending in G, $\mathcal{C}[\varphi,G]$ be the* MRM *induced by $\mathcal{C}$ and LDP φ, and $\mathcal{S}$ the set of linear constraints generated by ς, φ and time bound t. We have that:*

$$\Pr^{\mathcal{C}(s)[\varphi,G]}(\{\mathbf{Y}(t) \leq \mathbf{y} \wedge \varsigma\}) = Prob(\varsigma[\mathcal{S}]),$$

where $\mathbf{y} = \mathbf{M} = (M_0,...,M_{|J|-1})$.

Proof Let $\mathcal{C}(s)$ be the CTMC $\mathcal{C}$ such that, for a given state $s \in S$, $\alpha(s) = 1$. From Theorem 4.3.2 we know that:

$$\Pr^{\mathcal{C}(s)[\varphi,G]}(\{\mathbf{Y}(t) \leq \mathbf{y}\}) = \Pr^{\mathcal{C}(s)}(\{\rho \in Paths^{\mathcal{C}}(s) \mid \rho \models_t^G \varphi\}).$$

Let ς be a discrete path of length k such that $\varsigma[0] = s$. We have that:

$$\begin{aligned}
&\Pr^{\mathcal{C}(s)[\varphi,G]}\left(\{\mathbf{Y}(t) \leq \mathbf{y} \wedge \varsigma\}\right) = \\
&\Pr^{\mathcal{C}(s)[\varphi,G]}\Bigg(\{X(t) = \varsigma[k], \mathbf{Y}(t) \leq \mathbf{y} \wedge \\
&\qquad\qquad \exists z_0, \ldots, z_{k-1}, 0 \leq z_0 < z_1 < \cdots < z_{k-1} < t, X(0) = s, \bigwedge_{i=0}^{k-1} X(z_i) = \varsigma[i]\}\Bigg) = \\
&\Pr^{\mathcal{C}(s)[\varphi,G]}\left(\{\rho \in Paths^{\mathcal{C}}(s) \mid \rho \models_t^{\varsigma[k]} \varphi, \bigwedge_{i=0}^{k-1} \rho[i] = \varsigma[i]\}\right).
\end{aligned}$$

From Lemma 4.3.5 we obtain:

$$Prob(\varsigma[\mathcal{S}]) = \Pr^{\mathcal{C}(\varsigma[0])}\left(\left\{\rho \in Paths^{\mathcal{C}} \mid \varsigma[\rho\langle 0\rangle, \ldots, \rho\langle k-1\rangle] \models \varphi \wedge \int 1 \leq t\right\}\right).$$

One can easily see that:

$$\begin{aligned}
&\Pr^{\mathcal{C}(s)[\varphi,G]}\left(\left\{\rho \in Paths^{\mathcal{C}}(s) \mid \rho \models_t^{\varsigma[k]} \varphi, \bigwedge_{i=0}^{k-1} \rho[i] = \varsigma[i]\right\}\right) = \\
&\Pr^{\mathcal{C}(\varsigma[0])}\left(\left\{\rho \in Paths^{\mathcal{C}} \mid \varsigma[\rho\langle 0\rangle, \ldots, \rho\langle k-1\rangle] \models \varphi \wedge \int 1 \leq t\right\}\right).
\end{aligned}$$

This completes the proof. ■

Complete algorithm for the time-bounded verification of EDP

In order to compute the joint probability distribution of states and rewards, $F_s^{s'}(t, \mathbf{y})$ (see Equation 4.21), we must pick a finite set $\mathcal{P}$ of paths from $Paths^{\mathcal{D}}$, where $\mathcal{D}$ is the embedded DTMC of the CTMC under consideration. Following [QS94], we introduce a threshold $w \in (0, 1)$ such that $\varsigma \in \mathcal{P}$ only if $Prob(\varsigma) > w$. This is mainly for efficiency considerations. We also fix a maximum length N for the paths in $\mathcal{P}$. Now we define

$$\mathcal{P}(s, s', w, N) := \{\varsigma \in Paths^{\mathcal{D}} \mid |\varsigma| = N, \varsigma[0] = s, \varsigma[N] = s', Prob(\varsigma) > w\}.$$

We can approximate $F_s^{s'}(t, \mathbf{y})$ as

$$\widetilde{F}_{Ns}^{ws'}(t, \mathbf{y}) = \sum_{n=0}^{N} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \sum_{\varsigma \in \mathcal{P}(s,s',w,n)} Prob(\varsigma) \Pr\left(\{\mathbf{Y}(t) \leqslant \mathbf{y} \mid \varsigma\}\right),$$

where w and N are chosen as stated in Theorem 4.3.8. The approximation algorithm to compute $Prob := \widetilde{F}_{Ns}^{ws'}(t, \mathbf{y})$ is given in Algorithm 7. In Algorithm 7 we define $\circ$ to be the concatenation operator and we indicate with $\varsigma[|\varsigma|]$ the last state of ς.

Algorithm 7 Compute $\widetilde{F}^{w\ s'}_{N\ s}(t, \mathbf{y})$

```
1: Prob = 0
2: Paths = {s}
3: while Paths ≠ ∅ do
4:    choose ς ∈ Paths
5:    Paths = Paths \ {ς}
6:    if Prob(ς) > w and |ς| ≤ N then
7:       if ς[|ς|] = s' then
8:          Prob+ = e^{-Λt} ((Λt)^{|ς|}/|ς|!) Prob(ς) Pr({Y(t) ⩽ y | ς})
9:       else
10:         for s'' ∈ S do
11:            insert (ς ∘ s'') into Paths
12:         end for
13:      end if
14:   end if
15: end while
16: return Prob
```

A bound on the approximation error of Algorithm 7

We give a bound for the truncation of the infinite sum to a finite one, considering only the discrete paths whose probability is greater than w. We start with the following technical lemma.

Lemma 4.3.7 shows how to truncate the Poisson series $\sum_{i=0}^{\infty} \frac{e^{-\Lambda T}(\Lambda T)^i}{i!}$ for a sufficiently large integer N, such that the tail of the series, i.e., $\sum_{i=N+1}^{\infty} \frac{e^{-\Lambda T}(\Lambda T)^i}{i!}$, is smaller than a given ε, where ε is the maximum error that we are willing to tolerate.

Lemma 4.3.7 *Let $\varepsilon \in \mathbb{R}_{>0}$ and $T \in \mathbb{R}_{\geq 0}$. For any $N > \Lambda T e^2 + \ln(\frac{1}{\varepsilon})$, we have that*

$$\sum_{i=N+1}^{\infty} \frac{e^{-\Lambda T}(\Lambda T)^i}{i!} \leqslant \varepsilon.$$

Proof We have that

$$
\begin{aligned}
&\sum_{i=N+1}^{\infty} \frac{e^{-\Lambda T}(\Lambda T)^i}{i!} = e^{-\Lambda T} \cdot \left(\sum_{i=N+1}^{\infty} \frac{(\Lambda T)^i}{i!} \right) \\
&\leqslant e^{-\Lambda T} \cdot e^{\Lambda T} \cdot \frac{(\Lambda T)^N}{N!} && \text{(Taylor expansion)} \\
&\leqslant \frac{(\Lambda T)^N}{(N/e)^N} = \left(\frac{\Lambda T e}{N} \right)^N && \text{(Stirling's approximation)} \\
&\leqslant \left(\frac{1}{e} \right)^N && (N > \Lambda T e^2) \\
&\leqslant \left(\frac{1}{e} \right)^{\ln(1/\varepsilon)} = \varepsilon. && (N > \ln(\frac{1}{\varepsilon}))
\end{aligned}
$$

The following theorem states the error bound, which also suggests how to choose N and w for Algorithm 7 for a given ε.

Theorem 4.3.8 *Given $\varepsilon > 0$, for $N > \Lambda t e^2 + \ln\left(\frac{1}{\varepsilon}\right)$, and $w < \frac{\varepsilon}{\sum_{n=0}^{N} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!}}$, we have that*

$$\left| F_s^{s'}(t, \mathbf{y}) - \widetilde{F_{N}^{w}}_{s}^{s'}(t, \mathbf{y}) \right| \leqslant 2\varepsilon.$$

Proof

$$
\begin{aligned}
&\left| F_s^{s'}(t, y) - \widetilde{F_{N}^{w}}_{s}^{s'}(t, \mathbf{y}) \right| \\
&= \left| \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \sum_{\substack{|\varsigma|=n, \\ \varsigma[0]=s, \\ \varsigma[n]=s'}} \Pr(\{\varsigma\}) \cdot \Pr\left(\{\mathbf{Y}(t) \leqslant \mathbf{y} \mid \varsigma\}\right) - \right. \\
&\qquad \left. \sum_{n=0}^{N} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \cdot \sum_{\varsigma \in \mathcal{P}(s,s',w,n)} \Pr(\{\varsigma\}) \Pr\left(\{Y(t) \leqslant \mathbf{y} \mid \varsigma\}\right) \right|
\end{aligned}
$$

$$
= \underbrace{\left| \sum_{n=N+1}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \sum_{\substack{|\varsigma|=n, \\ \varsigma[0]=s, \\ \varsigma[n]=s'}} \Pr(\{\varsigma\}) \cdot \Pr\left(\{\mathbf{Y}(t) \leqslant \mathbf{y} \mid \varsigma\}\right) \right|}_{(\star)}
$$

$$+\underbrace{\sum_{n=0}^{N} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!}\cdot \sum_{\substack{|\varsigma|=n,\\ \varsigma[0]=s,\\ \varsigma[n]=s'}} \Pr(\{\varsigma\})\cdot \Pr\left(\{\mathbf{Y}(t)\leqslant \mathbf{y}\mid \varsigma\}\right) -}_{(\star\star)}$$

$$\underbrace{\sum_{n=0}^{N} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!}\cdot \sum_{\varsigma\in\mathcal{P}(s,s',w,n)} \Pr(\{\varsigma\})\Pr\left(\{Y(t)\leqslant \mathbf{y}\mid \varsigma\}\right)}_{(\star\star)}\Bigg|$$

We bound term $(\star)$ and term $(\star\star)$ separately.

- First, for $N > \Lambda t e^2 + ln\left(\frac{1}{\epsilon}\right)$ and by Lemma 4.3.7:

$$(\star) \leqslant \sum_{n=N+1}^{\infty} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!} \leqslant \epsilon$$

- Second:

$$\begin{aligned}
(\star\star) &= \sum_{n=0}^{N} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!}\cdot \sum_{\varsigma\notin\mathcal{P}(s,s',w,n)} \Pr(\{\varsigma\})\Pr\left(\{Y(t)\leqslant \mathbf{y}\mid \varsigma\}\right)\\
&\leqslant \sum_{n=0}^{N} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!}\cdot w\cdot \sum_{\varsigma\notin\mathcal{P}(s,s',w,n)} \Pr\left(\{Y(t)\leqslant \mathbf{y}\mid \varsigma\}\right)\\
&\leqslant w\cdot \sum_{n=0}^{N} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!}.
\end{aligned}$$

It follows that:

$$\left|F_s^{s'}(t,y) - \widetilde{F_{Ns}^{w}}^{s'}(t,\mathbf{y})\right| \leqslant \left|\epsilon + w\cdot \sum_{n=0}^{N} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!}\right|.$$

Taking $w \leqslant \frac{\epsilon}{\sum_{n=0}^{N} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!}}$, we obtain:

$$\left|F_s^{s'}(t,y) - \widetilde{F_{Ns}^{w}}^{s'}(t,\mathbf{y})\right| \leqslant 2\epsilon.$$

This completes the proof. ■

Complexity

We analyse the complexity of Algorithm 7. Recall that $|S|$ is the number of states of $\mathcal{C}$. Algorithm 7 is composed of two main steps: (1) find all paths of length at most N; and (2) for each of those paths, ς, compute $\Pr\left(\{\mathbf{Y}(t)\leqslant \mathbf{y}\mid \varsigma\}\right)$.

Theorem 4.3.9 *The complexity of Algorithm 7 is* $\mathcal{O}(|S|^N\cdot(|J|+|J|^N))$.

Proof The number of paths of length less than $N-1$, from standard graph theory, is at most $|S|^N$ (in case of fully connected CTMCs). Subsequently, for each of those $|S|^N$ paths, say ς, we have to compute $\Pr(\{\mathbf{Y}(t) \leqslant \mathbf{y} \mid \varsigma\})$. Using the approach that generates the set of linear constraints we have that the complexity to compute the volume of convex polytopes defined over N variables is $|J|^N$ (see [LZ01]), whereas the complexity to generate the set of linear constraints is linear in the cardinality of J. Therefore, the total complexity of Algorithm 7 is $\mathcal{O}(|S|^N \cdot (|J| + |J|^N))$. ■

4.3.2.2 Time-unbounded verification of EDP

In this section we show how to compute $Prob(\mathcal{C} \models^G \varphi)$. The main idea is that we approximate $Prob(\mathcal{C} \models^G \varphi)$ by $Prob(\mathcal{C} \models^G_T \varphi)$ for a sufficiently large $T \in \mathbb{R}_{\geq 0}$. Hence, we reduce the problem to time-bounded verification of EDP, which has been solved in Section 4.3.2.1.

For this purpose, we first introduce some background from linear algebra and matrix theory. We write $\mathbf{A}$ for a square matrix, with $a_{ij} \in \mathbb{R}$ the element of the i'th row and j'th column of $\mathbf{A}$. $\mathbf{A}$ is a *nonnegative matrix* if for any i, j, $a_{ij} \geq 0$. We write $eig(\mathbf{A})$ to be the set of all eigenvalues of matrix $\mathbf{A}$, and $\rho(\mathbf{A}) = \max\{|\lambda| \mid \lambda \in eig(\mathbf{A})\}$ be the *spectral radius* of $\mathbf{A}$, i.e., the maximum module of the eigenvalues of $\mathbf{A}$.

Definition 4.3.2 *Let* $\mathbf{A}$ *be a square matrix. The* logarithmic norm *of* $\mathbf{A}$*, denoted by* $\mu(\mathbf{A})$*, is defined as*

$$\mu(\mathbf{A}) = \max\left\{\lambda \middle| \lambda \in eig\left(\frac{\mathbf{A} + \mathbf{A}^\top}{2}\right)\right\}.$$

Note that this is well defined; as $\frac{\mathbf{A}+\mathbf{A}^\top}{2}$ *is a symmetric matrix, all the eigenvalues are reals.*

Note that $\mu(\mathbf{A}) \leq \rho\left(\frac{\mathbf{A}+\mathbf{A}^\top}{2}\right)$ and $\rho(\mathbf{A}) = \rho(\mathbf{A}^\top)$.

Definition 4.3.3 *Let* $\mathbf{A}$ *be a square matrix of dimension* m*. We call the graph* $\mathcal{G}_\mathbf{A}$ *of* $\mathbf{A}$ *the dependency graph where:*

- *the nodes of the graph are* $\{1, \cdots, m\}$*, and*
- *there is an edge from* i *to* j *iff* $a_{ij} > 0$*.*

Let $\mathcal{G}_\mathbf{A}$ *be a dependency graph. Recall from Section 3.1.2 that* $\mathcal{G}_\mathbf{A}$ *is called* strongly connected *if there is a path from each vertex in* $\mathcal{G}_\mathbf{A}$ *to every other vertex. The* strongly connected components *(SCCs) of* $\mathcal{G}_\mathbf{A}$ *are its maximal strongly connected subgraphs. Moreover, a matrix* $\mathbf{A}$ *is* irreducible *iff* $\mathcal{G}_\mathbf{A}$ *is strongly connected.*

Next we introduce a series of lemmas and proposition that will help to prove the main result of this section, i.e., Theorem 4.3.16. The end goal is to find a suitable time bound T such that we can approximate $Prob(\mathcal{C} \models^G \varphi)$ with $Prob(\mathcal{C} \models^G_T \varphi)$.

Proposition 4.3.10 ([Dah58]) *Let $||\cdot||$ be the spectral matrix norm, α be a vector with its associated Euclidean vector norm, and $T \geq 0$. It holds that:*

$$||\alpha \cdot e^{\mathbf{Q}T}|| \leq ||\alpha|| \cdot e^{\mu(\mathbf{Q})T}.$$

Proposition 4.3.11 ([HJ90]) *Let $\mathbf{A}$ be an arbitrary matrix and $\epsilon > 0$, then there exists some induced matrix norm $||\cdot||$ such that:*

$$||\mathbf{A}|| \leq \rho(\mathbf{A}) + \epsilon.$$

Definition 4.3.4 *An $m \times m$* substochastic matrix $\mathbf{A}$ *is a nonnegative matrix with the following properties:*

- *for any $0 \leqslant i \leqslant m$, $\sum_{1 \leqslant j \leqslant m} a_{ij} \leqslant 1$; and*
- *there exists some $0 \leqslant i \leqslant m$, such that $\sum_{1 \leqslant j \leqslant m} a_{ij} < 1$.*

Lemma 4.3.12 *Let $\mathbf{A}$ be an $m \times m$ irreducible substochastic matrix. It holds that $\rho(\mathbf{A}) < 1$.*

Proof For any $1 \leqslant i \leqslant m$ let $r_i^{(n)} = \sum_{k=1}^{m} \mathbf{A}_{ik}^n$ be the i-th row sum of $\mathbf{A}^n$. For $n = 1$ we write r_i instead of $r_i^{(1)}$. Since $\mathbf{A}$ is substochastic we have that $0 \leqslant r_i \leqslant 1$ for any $1 \leqslant i \leqslant m$ and $r_j < 1$ for at least one $1 \leqslant j \leqslant m$. Note that for $n \geq 1$:

$$r_j^{(n)} = \sum_{k=1}^{m} \mathbf{A}_{jk}^n = \sum_{k=1}^{m} \mathbf{A}_{jk} r_k^{(n-1)} \leqslant \sum_{k=1}^{m} \mathbf{A}_{jk} = r_j < 1.$$

By irreducibility of $\mathbf{A}$, for any i there is l such that $\mathbf{A}_{ij}^l > 0$. In fact, given that $\mathbf{A}$ is a $m \times m$ matrix and $i \neq j$ then we can assume $l < m$. Thus, we have that:

$$r_i^{(m)} = \sum_{k=1}^{m} \mathbf{A}_{ik}^l r_k^{(m-l)} < r_i^{(l)} \leqslant 1.$$

By the Gershgorin circle theorem [HJ90], we have that $\rho(A^m) < 1$. Hence $\rho(A) < 1$. ■

Lemma 4.3.13 *Suppose that $\rho(\mathbf{A}) < 1$, then $\mu(\mathbf{A}) < 1$.*

Proof We know that $\mu(\mathbf{A}) \leq \rho\left(\frac{\mathbf{A}+\mathbf{A}^\top}{2}\right)$. For *any* induced matrix norm $||\cdot||$, it holds that:

$$\rho\left(\frac{\mathbf{A}+\mathbf{A}^\top}{2}\right) \leq \frac{1}{2}(||\mathbf{A}+\mathbf{A}^\top||) \leq \frac{1}{2}||\mathbf{A}|| + \frac{1}{2}||\mathbf{A}^\top||.$$

Let $\epsilon > 0$ then from Proposition 4.3.11 it holds that for some matrix norm $||\cdot||$:

$$\begin{aligned}\mu(\mathbf{A}) &\leq \rho\left(\frac{\mathbf{A}+\mathbf{A}^\top}{2}\right)\\ &\leq \frac{1}{2}||\mathbf{A}|| + \frac{1}{2}||\mathbf{A}^\top||\\ &\leq \frac{1}{2}\rho(\mathbf{A}) + \frac{1}{2}\epsilon + \frac{1}{2}\rho(\mathbf{A}^\top) + \frac{1}{2}\epsilon\\ &= \rho(\mathbf{A}) + \epsilon.\end{aligned}$$

From Lemma 4.3.12 we know that $\rho(\mathbf{A}) < 1$ and so we can pick an ϵ such that $\rho(\mathbf{A}) + \epsilon < 1$. It follows that $\mu(\mathbf{A}) < 1$. ■

Now fix the CTMC $\mathcal{C}$ and the set of goal states $G \subseteq S$ with $|G| = m$. Recall that $\mathbf{Q}$ is the infinitesimal generator of $\mathcal{C}$. As the first step, we identify the set of states $S_{>0} \subseteq S$ starting from which there is positive probability to reach G. This can be done through a graph analysis in a standard way, see [BK08, Ch. 10]. We still write $\mathbf{Q}_{>0}$ for the principal submatrix of the infinitesimal generator $\mathbf{Q}$ corresponding to $S_{>0}$. We partition $\mathbf{Q}_{>0}$ as follows

$$\mathbf{Q}_{>0} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \tag{4.30}$$

where $\mathbf{Q}_1$ is the square matrix of size $(|S_{>0}| - m) \times (|S_{>0}| - m)$ denoting transitions between the set of transient states $s \in S_{>0} \setminus G$, $\mathbf{Q}_2$ is the matrix of size $(|S_{>0}| - m) \times m$ denoting transitions from the transient states to the set of goal states G and $\mathbf{0}$ is a matrix composed of zeros. The reader should note that, given any infinitesimal generator $\mathbf{Q}$, it is always possible to express $\mathbf{Q} = \Lambda(\mathbf{P} - \mathbf{I})$, where Λ is the maximal exit rate of $\mathcal{C}$, $\mathbf{I}$ is the identity matrix and $\mathbf{P} = (\mathbf{I} + \frac{\mathbf{Q}}{\lambda})$ is a stochastic matrix. In the sequel we indicate with $\mathbf{P}_1$ the principal submatrix of $\mathbf{P}$ corresponding to $\mathbf{Q}_1$. Abusing notation we indicate with $\mathbf{I}_1$ the identity matrix of the same size as $\mathbf{P}_1$.

We define a random variable $T_G : \mathit{Paths}^{\mathcal{C}} \to \mathbb{R}_{\geq 0}$ that will denote the first entrance time of G. More specifically, given a path ρ:

$$T_G(\rho) = \begin{cases} \infty & \forall j \in \mathbb{N}.\ \rho[j] \notin G \\ \sum_{j=0}^{k-1} \rho\langle j\rangle & \text{o/w, where } k = \min\{l \mid \rho[l] \in G\} \end{cases}.$$

The following proposition states a helpful property of the "transient part" of the infinitesimal generator of $\mathcal{C}$, relying on Lemma 4.3.12 and Lemma 4.3.13. Note that [ESY12] contains a similar argument showing essentially the same result, although in a different context.

Proposition 4.3.14

$$\mu(\mathbf{Q}_1) < 0.$$

Proof We first focus our attention on $\mathbf{P}_1$, which is a substochastic matrix. Let $\mathcal{G}_{\mathbf{P_1}}$ be the dependency graph of $\mathbf{P}_1$. We consider the SCC-decomposition of $\mathcal{G}_{\mathbf{P_1}}$, and assume a topological ordering among SCCs $\{B_1, \cdots, B_k\}$ such that, for $i \in B_m$ and $i' \in B_{m'}$, the existence of an edge from i to i' implies that $m < m'$. By Lemma 4.3.12, we have the following property: for any $\ell \in \{1, \cdots, k\}$ and the principal submatrix corresponding to B_ℓ, written as $\mathbf{P}_1[B_\ell]$,

$$\rho(\mathbf{P}_1[B_\ell]) < 1. \tag{4.31}$$

Since $\mathbf{P}_1$ is a nonnegative matrix, we have that there exists a nonnegative eigenvector v associated with $\rho(\mathbf{P}_1)$, i.e.,

$$\mathbf{P}_1 v = \rho(\mathbf{P}_1) v$$

We observe that, for any index $1 \leqslant i \leqslant n$, if $v_i > 0$ then, for any j such that there is an edge from j to i, we have that:

$$\begin{aligned}
(\mathbf{P}_1 v)_j &= \sum_{1 \leqslant k \leqslant n} p_{jk} v_k \\
&= \sum_{\substack{1 \leqslant k \leqslant n, \\ k \neq i}} p_{jk} v_k + p_{ji} v_i \\
&\geq p_{ji} v_i \\
&> 0.
\end{aligned}$$

Since $(\mathbf{P}_1 v)_j = \rho(\mathbf{P}_1) v_j$, we obtain that $v_j > 0$. Repeating the same argument, we have that, for each SCC, if for *some* index i we have $v_i > 0$, then for *any* index i in this SCC, $v_i > 0$.

It follows that there must exist some SCC such that, for any index i in this SCC, we have $v_i > 0$. Let $\hbar$ be the maximum index for such an SCC. Consider the principal submatrix corresponding to $B_\hbar$. For each index $i \in B_\hbar$, we have that:

$$\begin{aligned}
(\mathbf{P}_1 v)_i &= \sum_{1 \leqslant j \leqslant n} p_{ij} v_j \\
&= \sum_{\substack{1 \leqslant j \leqslant n, \\ j \in B_\hbar}} p_{ij} v_j + \sum_{\substack{1 \leqslant j \leqslant n, \\ j \notin B_\hbar}} p_{ij} v_j \\
&= \sum_{\substack{1 \leqslant j \leqslant n, \\ j \in B_\hbar}} p_{ij} v_j \\
&= \rho(\mathbf{P}_1) v_i.
\end{aligned}$$

It follows that $\rho(\mathbf{P}_1[B_\hbar]) \geq \rho(\mathbf{P}_1)$. However, we also have $\rho(\mathbf{P}_1[B_\hbar]) \leqslant \rho(\mathbf{P}_1)$ as $\mathbf{P}_1[B_\hbar]$ is a principal submatrix. Hence $\rho(\mathbf{P}_1[B_\hbar]) = \rho(\mathbf{P}_1)$. Therefore, $\rho(\mathbf{P}_1) < 1$ by Equation (4.31).

Now note that by Lemma 4.3.13 if $\rho(\mathbf{P}_1) < 1$ then $\mu(\mathbf{P}_1) < 1$. Moreover, $\mu(\mathbf{Q}_1) = \mu(\Lambda(\mathbf{P}_1 - \mathbf{I}_1))$ which in turn yields that $\mu(\mathbf{Q}_1) \leq \Lambda(\mu(\mathbf{P}_1) - 1)$ since $\mu(\mathbf{I}_1) = 1$. Thus, $\mu(\mathbf{P}_1) < 1$ implies that $\mu(\mathbf{Q}_1) < 0$, which concludes the proof. ■

Next we present Proposition 4.3.15 which relates the probability of all the timed paths ρ of $\mathcal{C}$ that reach the set of goal states G for the first time in more than T time units to an exponential function over the infinitesimal generator $\mathbf{Q}$ of $\mathcal{C}$.

Proposition 4.3.15 *For any $T \in \mathbb{R}_{\geq 0}$ it holds that:*

$$\Pr^{\mathcal{C}}(\{\rho \in Paths^{\mathcal{C}} \mid \rho \models \Diamond G \wedge T_G(\rho) > T\}) = \hat{\alpha} \cdot e^{\mathbf{Q}_1 T}\mathbf{e},$$

where $\hat{\alpha} = \alpha[1, \ldots |S_{>0}| - m]$ and $\mathbf{e}$ is a vector assigning 1's to the goal states and 0's to all the other states.

Proof Proof can be found in [NNN10]. ■

Now we are in a position to state the main result of this section. Theorem 4.3.16 shows how to approximate time-unbounded EDPs with time-bounded EDPs. More precisely, Theorem 4.3.16 states that, given a maximum error tolerance ε, we can choose a time bound T $\left(T > \frac{\ln(\varepsilon/\sqrt{|G|})}{\mu(\mathbf{Q}_1)}\right)$ such that the probability of all the timed paths that satisfy the formula φ $\left(Prob(\mathcal{C} \models^{G} \varphi)\right)$ is ε-close to the probability of all the timed paths that satisfy the formula φ in T time units $\left(Prob(\mathcal{C} \models^{G}_{T} \varphi)\right)$.

Theorem 4.3.16 *Given $0 < \varepsilon < 1$ and $T > \frac{\ln(\varepsilon/\sqrt{|G|})}{\mu(\mathbf{Q}_1)}$:*

$$Prob(\mathcal{C} \models^{G} \varphi) - Prob(\mathcal{C} \models^{G}_{T} \varphi) \leq \varepsilon.$$

Proof We have

$$\begin{aligned}
&Prob(\mathcal{C} \models^{G} \varphi) - Prob(\mathcal{C} \models^{G}_{T} \varphi) \\
\leqslant &\Pr^{\mathcal{C}}(\{\rho \in Paths^{\mathcal{C}} \mid \rho \models \Diamond G \wedge T_G(\rho) \geq T\}) \\
= &\hat{\alpha} \cdot e^{\mathbf{Q}_1 T}\mathbf{e} = ||\hat{\alpha} \cdot e^{\mathbf{Q}_1 T}\mathbf{e}|| && \text{(by Proposition 4.3.15)} \\
\leqslant &||\hat{\alpha}|| \cdot e^{\mu(\mathbf{Q}_1)T} \cdot ||\mathbf{e}|| && \text{(by Proposition 4.3.10)} \\
\leq &\varepsilon
\end{aligned}$$

The correctness of the bound is guaranteed by Proposition 4.3.14. ■

Due to this theorem, given an error bound ε and a set of goal states G, we can pick a time bound T such that $T \geq \frac{\ln(\varepsilon/\sqrt{|G|})}{\mu(\mathbf{Q}_1)}$ and compute $Prob(\mathcal{C} \models_T^G \varphi)$. For computing $\mu(\mathbf{Q}_1)$, we note that it only requires computing eigenvalues of the symmetrisation of $\mathbf{Q}_1$ for which efficient numerical algorithms exist.

4.3.3 Verification of IDP

In this section, we tackle the problem of verification of IDPs. Recall that IDPs are different from EDPs in the sense that they require the formula to hold at any instant of time along the path under consideration. Again, we fix a CTMC $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$ and an LDP

$$\Phi = \int 1 \leqslant T \to \underbrace{\bigwedge_{j \in J} \left(\sum_{k \in K_j} c_{jk} \int \mathrm{sf}_{jk} \leqslant M_j \right)}_{\varphi}.$$

As highlighted in Section 3.1.2, we shall distinguish two cases according to whether T is finite or infinite. We firstly give some definitions and algorithms that are common to both cases.

Given φ, a *discrete* finite path ς of length k and a time bound $T < \infty$, we define the set of linear constraints $\mathcal{S}$ as generated in Algorithm 8. Note that this is different from the constraints obtained from Algorithm 6 in the previous section.

Algorithm 8 Generate a set of linear constraints $\mathcal{S}$ induced by φ, ς and T

Require: LDP φ, a path ς of length k and a time bound T

Ensure: A set of linear constraints $\mathcal{S}$

1: $\mathcal{S} = \varnothing$
2: **for** $z = 0;\ \ z < k;\ \ z{+}{+}$ **do**
3: **for** $j \in J$ **do**
4: $\mathcal{S} = \mathcal{S} \cup \left\{ \sum_{i \in K_j} c_{ji} \cdot \sum_{\substack{0 \leq \ell \leq z, \\ \varsigma[\ell] \models \mathrm{sf}_{ji}}} x_\ell \leqslant M_j \right\}$
5: **end for**
6: **end for**
7: **return** $\mathcal{S}$

We show in Example 4.3.4 a concrete example of the set of constraints generated by Algorithm 8 for a given discrete path ς.

Example 4.3.4 *Let* $\varphi = \int Busy - \int Idle \leqslant 0$ *be an LDP and* $\varsigma = s_0 \to s_1 \to s_0 \to s_1 \to$

s_3. The set of linear constraints $\mathcal{S}$ induced by ς and φ is:

$$\mathcal{S} = \begin{cases} x_{00} \leqslant 0 \\ x_{00} - x_{01} \leqslant 0 \\ x_{00} - x_{01} + x_{02} \leqslant 0 \\ x_{00} - x_{01} + x_{02} - x_{03} \leqslant 0 \\ x_{00}, x_{01}, x_{02}, x_{03} > 0 \end{cases}$$

In order to show the correctness of Algorithm 8 we need to prove that, given a discrete path ς of length k of $\mathcal{C}$, then any solution of the set of linear constraints $\mathcal{S}$ returned by Algorithm 8, say $(x_0, \ldots, x_{n-1})$, must satisfy the following condition: the timed path obtained by plugging the solution $(x_0, \ldots, x_{n-1})$ of $\mathcal{S}$ into ς, namely $\varsigma[x_0, \ldots, x_{n-1}]$, must verify φ. This is proved in Lemma 4.3.17.

Lemma 4.3.17 *Let ς be a finite path of the* CTMC *$\mathcal{C}$, φ be an LDP and T be a time bound. Moreover, let $\mathcal{S}$ be the set of linear constraints obtained by Algorithm 8. Then*

$$\varsigma[x_0, \ldots, x_{n-1}] \models^\star \left(\varphi \wedge \int 1 \leq T\right) \quad \textit{iff} \quad (x_0, \ldots, x_{n-1}) \in \mathcal{S}.$$

Proof Let φ_j be the j-th conjunct of φ. From Definition 3.2.8 we have that

$$\varsigma[x_0, \ldots, x_{n-1}] \models^\star \varphi_j \quad \text{iff} \quad (x_0, \ldots, x_{n-1}) \in \mathcal{S} = \bigcup_{z=0}^{n-1} \left\{ \sum_{i \in K_j} c_{ji} \cdot \sum_{\substack{0 \leq \ell \leq z, \\ \varsigma[\ell] \models \mathrm{sf}_{ji}}} x_\ell \leqslant M_j \right\}.$$

Note that $\varsigma[x_0, \ldots, x_{n-1}] \models \int 1 \leq t$ iff $\sum_{i=0}^{n-1} x_i \leq t$ (see Definition 3.2.8), which proves the lemma. ■

We define $Prob^\star(\varsigma[\mathcal{S}])$ to be

$$\Pr^{\mathcal{C}}(\{\rho \in Paths^{\mathcal{C}} \mid \exists\, (x_0, \ldots, x_{n-1}) \in \mathcal{S}.\ \rho[0..n] \in \varsigma[x_0, \ldots, x_{n-1}] \wedge \rho[0..n] \models^\star \varphi\}),$$

which can be computed by the function $Volume\_int(\alpha, \varsigma, \mathcal{S})$ (cf. Equation (4.29)), where $\mathcal{S}$ is the set of constraints generated from Algorithm 8. We now introduce an auxiliary definition for paths of CTMCs.

Definition 4.3.5 *Given an infinite timed path ρ, an* absorbing *set of states G of the* CTMC *$\mathcal{C}$, and a time bound $T < \infty$, we write $\rho \models^\star_{G,T} \varphi$ if there exists some $n \in \mathbb{N}$ such that:*

- *$\rho[n] \in G$ and $\sum_{i=0}^{n} \rho\langle i \rangle \leqslant T$, and*
- *for each $0 \leqslant i \leqslant n$, $\rho[0..i] \models \varphi$.*

Remark 4.3.4 *Note that, as we assume that G is absorbing, the only difference between $\rho \models^{\star}_{G,T} \varphi$ and $\rho \models^{G}_{T} \varphi$ given in Definition 3.2.8 lies in that, here, we require that, for each $0 \leqslant i \leqslant n$, $\rho[0..i] \models \varphi$, whereas in Definition 3.2.8 we require that $\rho[0..n] \models \varphi$. This reflects the distinction between EDP and IDP.*

Our task now is to approximate the probability $Prob(\mathcal{C} \models^{\star}_{G,T} \varphi)$. For this purpose, we present Algorithm 9 which computes an approximation $\widetilde{Prob}_N(\mathcal{C} \models^{\star}_{G,T} \varphi)$ of $Prob(\mathcal{C} \models^{\star}_{G,T} \varphi)$ for a given N.

Algorithm 9 Compute $\widetilde{Prob}_N(\mathcal{C} \models^{\star}_{G,T} \varphi)$

Require: A CTMC $\mathcal{C}$, an LDP formula φ, set of goal states G, time bound T, and N

1: $Prob = 0$
2: **for** $\varsigma \in Paths^{\mathcal{D}}$ s.t. $\exists i.\ \varsigma[i] \in G$ and $|\varsigma| \leq N$ **do**
3: Generate $\mathcal{S}$ from φ, ς and T, with Algorithm 8
4: $Prob = Prob + Volume\_int(\alpha, \varsigma, \mathcal{S})$
5: **end for**
6: **return** $Prob$

4.3.3.1 Verification of unbounded IDP

This section is devoted to computing $Prob(\mathcal{C} \models^{\star} \varphi)$. For this purpose, we need to perform a graph analysis of $\mathcal{C}$. We start with some standard definitions.

Definition 4.3.6 *Given a BSCC B of the* CTMC $\mathcal{C}$ *and an LDP φ, we say*

- *B is* bad *w.r.t. the j-th conjunct in φ, φ_j, if*

$$\exists s \in B.\ \exists i \in K_j.\ s \models \mathrm{sf}_{ji} \wedge c_{ji} > 0$$

  and otherwise B is good *w.r.t. φ_j.*

- *B is* good *w.r.t. φ (written $B \models \varphi$) if B is good for each conjunct of φ; otherwise B is* bad *(written $B \not\models \varphi$).*

Our next step is to prove that, given a CTMC $\mathcal{C}$, an LDP φ and a bad BSCC B, then the probability of all the timed paths ρ that reach B and satisfy φ is equal to zero. This is formally proved in Lemma 4.3.18. Intuitively, this is due to one of the following two facts:

- The formula was false before reaching B; or
- The formula was true before reaching B but then it will become false once B is reached. Moreover, since B is a bad BSCC, it will stay there forever and the formula will stay false forever.

Lemma 4.3.18 *Given a* CTMC $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$*, an LDP* φ *and a BSCC* B*, we have that, if* B *is bad, then* $\Pr^{\mathcal{C}}(\{\rho \mid \rho \models^{\star} \varphi\} \mid \Diamond B) = 0$.

Proof We have the following basic facts, which follow from ergodicity theorems related to stochastic processes (see [MT09]):

1. Given a BSCC B, every state $s \in B$ is visited infinitely often with probability 1.
2. Any path $\rho \in \mathit{Paths}^{\mathcal{C}}$ eventually reaches one of the BSCCs of $\mathcal{C}$.

Given the second fact we only need to prove that for a bad BSCC B it holds that $\Pr^{\mathcal{C}}(\{\rho \mid \rho \models^{\star} \varphi\} \mid \Diamond B) = 0$. We note that:

$$\Pr^{\mathcal{C}}(\{\rho \mid \rho \models^{\star} \varphi\} \mid \Diamond B) = \frac{\Pr^{\mathcal{C}}(\{\rho \mid \rho \models^{\star} \varphi\} \cap \Diamond B)}{\Pr^{\mathcal{C}}(\Diamond B)}.$$

Therefore, in order to prove that $\Pr^{\mathcal{C}}(\{\rho \mid \rho \models^{\star} \varphi\} \mid \Diamond B) = 0$, it is enough to show that $\{\rho \mid \rho \models^{\star} \varphi\} \cap \Diamond B = \varnothing$. We prove it by contradiction. First, observe that

$$\{\rho \mid \rho \models^{\star} \varphi\} \cap \Diamond B = \bigcap_{j \in J} \left(\{\rho \mid \rho \models^{\star} \varphi_j\} \cap \Diamond B\right),$$

where φ_j is the j-th conjunct of φ. Therefore, we will only show that $\{\rho \mid \rho \models^{\star} \varphi_j\} \cap \Diamond B = \varnothing$ for some $j \in J$. Let $\rho \in \{\rho \mid \rho \models^{\star} \varphi_j\} \cap \Diamond B$. Then $\rho \in \Diamond B$. Given that B is bad it holds that $\exists s \in B.\, \exists i \in K_j.\, \mathrm{sf}_{ji} \in L(s) \wedge c_{ji} > 0$. From the first fact we know that there exist infinitely many n such that $\rho[n] = s$. Therefore, we have that $c_{ji} \int \mathrm{sf}_{ji} \to \infty$. We also know that $\rho \models^{\star} \varphi_j$ iff $\forall n. \rho[0 \ldots n] \models \varphi$ or

$$\forall n. \sum_{k \in K_j} c_{jk} \sum_{\substack{0 \le i' < n, \\ \rho[0 \ldots n] \models \mathrm{sf}_{jk}}} \rho[0 \ldots n]\langle i' \rangle \le M_j. \tag{4.32}$$

Given that $i \in K_j$ and $c_{ji} \int \mathrm{sf}_{ji} \to \infty$, Equation (4.32) does not hold. Therefore, we have that $\rho \not\models^{\star} \varphi_j$, which is a contradiction. ■

Let **BSCC** be the set of all BSCCs in $\mathcal{C}$ and $\widetilde{\mathrm{BSCC}}$ be the set of all good BSCCs.

Definition 4.3.7 *Given a* CTMC $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$ *and an LDP* φ*, we define a new* CTMC $\mathcal{C}^a = (S, \mathrm{AP}^a, L^a, \alpha, \mathbf{P}^a, E)$ *as follows:*

- $\mathrm{AP}^a = \mathrm{AP} \cup \{\bot\}$*, where* $\bot$ *is fresh;*
- *for all* $s \in B$ *and* $B \in \widetilde{\mathrm{BSCC}}$ *make* s *absorbing and let* $L^a(s) = L(s) \cup \{\bot\}$*; and*
- *for all other states* $s \notin B$*,* $B \in \widetilde{\mathrm{BSCC}}$ *and* $s' \in S$*, let* $\mathbf{P}^a(s, s') = \mathbf{P}(s, s')$*,* $L^a(s) = L(s)$.

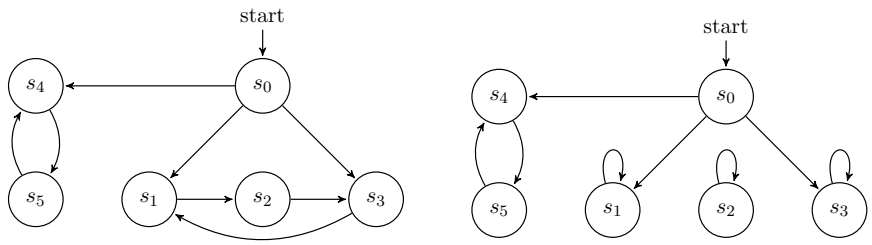


Figure 4.7: Example of BSCC decomposition to demonstrate CTMC conversion in Definition 4.3.7

Example 4.3.5 *Consider the left* CTMC $\mathcal{C}$ *from Figure 4.7, in which there are two BSCCs* $B_1 = \{s_4, s_5\}$ *and* $B_2 = \{s_1, s_2, s_3\}$*. Moreover, assume that* $B_1 \not\models \varphi$ *and* $B_2 \models \varphi$ *for a given LDP* φ*. After applying Definition 4.3.7 to* $\mathcal{C}$ *we get* $\mathcal{C}^a$ *shown on the right, where the labels of the states* s_1*,* s_2 *and* s_3 *are augmented with the label* $\{\bot\}$ *and all the other labels are left unchanged.*

We now introduce an auxiliary definition, which, roughly, is the counterpart of (the unbounded version of) Definition 4.3.5.

Definition 4.3.8 *Given an infinite timed path* ρ *and* $G \subset S$*, we write* $\rho \models^\star_G \varphi$ *if there exists some* $n \in \mathbb{N}$ *such that:*

- $\rho[n] \in G$*, and*
- *for each* $0 \leqslant i \leqslant n$*,* $\rho[0..i] \models \varphi$*.*

The following proposition, Proposition 4.3.19, states that in order to compute $Prob(\mathcal{C} \models^\star \varphi)$, one can first make good BSCCs absorbing while removing bad BSCCs, and then reduce to computing $Prob(\mathcal{C} \models^\star_G \varphi)$ for a suitable G, which, in turn, uses Algorithm 9.

Proposition 4.3.19 *Given a* CTMC $\mathcal{C} = (S, \mathrm{AP}, L, \alpha, \mathbf{P}, E)$ *and an LDP* φ*, we have that*

$$Prob(\mathcal{C} \models^\star \varphi) = \Pr^{\mathcal{C}^a}(\{\rho \mid \rho \models^\star_G \varphi\}),$$

where $G = \{s \in S \mid \bot \in L(s)\}$*.*

Proof Applying the law of total probability we have that

$$\begin{aligned}
&\Pr^{\mathcal{C}}(\{\rho \mid \rho \models^{\star} \varphi\}) \\
= &\sum_{B \in \mathbf{BSCC}} \Pr^{\mathcal{C}}\left(\{\rho \mid \rho \models^{\star} \varphi\} \mid \Diamond B\right) \cdot \Pr^{\mathcal{C}}(\Diamond B) \\
= &\sum_{B \in \widetilde{\mathrm{BSCC}}} \Pr^{\mathcal{C}}\left(\{\rho \mid \rho \models^{\star} \varphi\} \mid \Diamond B\right) \cdot \Pr^{\mathcal{C}}(\Diamond B) \quad \text{(by Lemma 4.3.18)} \\
= &\sum_{B \in \widetilde{\mathrm{BSCC}}} \Pr^{\mathcal{C}}\left(\{\rho \mid \rho \models^{\star} \varphi \wedge (\{\rho[0 \ldots n-1] \not\models \varphi\} \cup \{\rho[0 \ldots n-1] \models \varphi\})\} \mid \Diamond B\right) \\
&\cdot \Pr^{\mathcal{C}}(\Diamond B),
\end{aligned}$$

where for all $i < n$, $\rho[i] \notin B$. We have

$$\begin{aligned}
&\Pr^{\mathcal{C}}(\{\rho \mid \rho \models^{\star} \varphi\}) \\
= &\sum_{B \in \widetilde{\mathrm{BSCC}}} \Pr^{\mathcal{C}}\left(\{\rho \mid \rho \models^{\star} \varphi \wedge \rho[0 \ldots n-1] \not\models \varphi\} \mid \Diamond B\right) \cdot \Pr^{\mathcal{C}}(\Diamond B) \\
&+ \sum_{B \in \widetilde{\mathrm{BSCC}}} \Pr^{\mathcal{C}}\left(\{\rho \mid \rho \models^{\star} \varphi \wedge \rho[0 \ldots n-1] \models \varphi\} \mid \Diamond B\right) \cdot \Pr^{\mathcal{C}}(\Diamond B).
\end{aligned}$$

By definition of $\models^{*}$, $\Pr^{\mathcal{C}}\left(\{\rho \mid \rho \models^{\star} \varphi \wedge \rho[0 \ldots n-1] \not\models \varphi\} \mid \Diamond B\right) = 0$. Using similar reasoning as in Lemma 4.3.18, one can show that

$$\Pr^{\mathcal{C}}\left(\{\rho \mid \rho \models^{\star} \varphi \wedge \rho[0 \ldots n-1] \models \varphi\} \mid \Diamond B\right) = 1,$$

for any $B \in \widetilde{\mathrm{BSCC}}$. Therefore, we obtain that

$$\begin{aligned}
&\Pr^{\mathcal{C}}(\{\rho \mid \rho \models^{\star} \varphi\}) \\
= &\sum_{B \in \widetilde{\mathrm{BSCC}}} \Pr^{\mathcal{C}}(\Diamond B) = \sum_{B \in \widetilde{\mathrm{BSCC}}} \Pr^{\mathcal{C}}(\{\rho \mid \rho \models^{\star}_{B} \varphi\}) \\
= &\Pr^{\mathcal{C}}\left(\bigcup_{B \in \widetilde{\mathrm{BSCC}}} \{\rho \mid \rho \models^{\star}_{B} \varphi\}\right) \\
= &\Pr^{\mathcal{C}^a}(\{\rho \mid \rho \models^{\star}_{G} \varphi\}),
\end{aligned}$$

where $G = \bigcup_{B \in \widetilde{\mathrm{BSCC}}} \{s \in B\} = \{s \in S \mid \perp \in L(s)\}$ by Definition 4.3.7. ■

Complete algorithm for the time-unbounded verification of IDP

The general algorithm for computing time-unbounded IDP verification is given in Algorithm 10.

Algorithm 10 computes $\widetilde{Prob}(\mathcal{C} \models^{\star} \varphi)$, which is an approximation of $Prob(\mathcal{C} \models^{\star} \varphi)$. Lines 4-9 obtain $\mathcal{C}^a$ and the goal states G, according to Definition 4.3.7, and then the algorithm calls the function $\widetilde{Prob}_N(\mathcal{C} \models^{\star}_{G,T} \varphi)$, given by Algorithm 9 (on page 84), by choosing T and N according to the specified error bounds ε_1 and ε_2 respectively.

Algorithm 10 Compute $\widetilde{Prob}(\mathcal{C} \models^{\star} \varphi)$

Require: A CTMC $\mathcal{C}$, an LDP formula φ, ε_1 and ε_2

1: Identify all BSCCs B in $\mathcal{C}$
2: $G = \varnothing$
3: $Prob = 0$
4: **for each** BSCC B **do**
5: **if** $B \models \varphi$ **then**
6: Make every state in B absorbing
7: $G = G \cup B$
8: **end if**
9: **end for**
10: Choose $T = \frac{\ln\left(\frac{\varepsilon}{2\sqrt{|G|}}\right)}{\mu(\mathbf{Q}_1)}$ and $N = \frac{\Lambda e^2 \ln\left(\frac{\epsilon}{2\sqrt{|G|}}\right)}{\mu(\mathbf{Q}_1)} + \ln(\frac{2\sqrt{|G|}}{\varepsilon})$ (see Remark 4.3.5)
11: $Prob = \widetilde{Prob}_N(\mathcal{C} \models^{\star}_{G,T} \varphi)$
12: **return** *Prob*
13: Recall that $\mu(\mathbf{Q}_1)$ denotes the logarithmic norm of $\mathbf{Q}_1$ (cf. Definition 4.3.2)

A bound on the approximation error of Algorithm 10

Intuitively, there are two factors that contribute to the error introduced by Algorithm 10:

- the error introduced by approximating $\Pr^{\mathcal{C}^a}(\{\rho \mid \rho \models^{\star}_{G} \varphi\})$ with $Prob(\mathcal{C}^a \models^{\star}_{G,T} \varphi)$, which can be obtained in a similar way as for Theorem 4.3.16, denoted by ε_1; and
- the error introduced by approximating $Prob(\mathcal{C}^a \models^{\star}_{G,T} \varphi)$ with $\widetilde{Prob}_N(\mathcal{C}^a \models^{\star}_{G,T} \varphi)$, denoted by ε_2.

Theorem 4.3.20 *Given ε_1 and ε_2, we have that:*

$$Prob(\mathcal{C} \models^{\star} \varphi) - \widetilde{Prob}(\mathcal{C} \models^{\star} \varphi) \leq \varepsilon_1 + \varepsilon_2.$$

where $\widetilde{Prob}(\mathcal{C} \models^{\star} \varphi)$ can be computed by Algorithm 10.

Proof The claim follows from Theorem 4.3.8, 4.3.16, and Proposition 4.3.19. ■

Remark 4.3.5 *Given ε* a priori*, one possibility is to let $\varepsilon_1 = \varepsilon_2 = \frac{\varepsilon}{2\sqrt{|G|}}$, and hence $T = \frac{\ln\left(\frac{\varepsilon}{2\sqrt{|G|}}\right)}{\mu(\mathbf{Q}_1)}$ and $N = \frac{\Lambda e^2 \ln\left(\frac{\epsilon}{2\sqrt{|G|}}\right)}{\mu(\mathbf{Q}_1)} + \ln(\frac{2\sqrt{|G|}}{\varepsilon})$ suffice.*

4.3.3.2 Verification of time-bounded IDP

In this section we show how to deal with the time-bounded variant of IDP.

A well-known fact regarding CTMCs is that the set of Zeno paths is of probability 0.

Lemma 4.3.21 *Given a* CTMC $\mathcal{C}$ *and a time bound* $T < \infty$*, we have that:*

$$\Pr^{\mathcal{C}}\left(\left\{\rho \mid \rho \models^{\star} \int 1 \leqslant T\right\}\right) = 0.$$

We refer the readers to [BHHK03] for a proof.

For a CTMC $\mathcal{C}$, we write $\mathcal{C}[s]$ for the CTMC obtained from $\mathcal{C}$ by making the state s absorbing. The following theorem plays a pivotal role.

Theorem 4.3.22 *Given a* CTMC $\mathcal{C}$ *and an LDP* Φ *it holds that:*

$$\mathit{Prob}(\mathcal{C} \models^{\star} \Phi) = \sum_{s \in S} \mathit{Prob}(\mathcal{C}[s] \models^{\star}_{\{s\},T} \varphi).$$

Proof By the law of total probability we have that:

$$\Pr^{\mathcal{C}}(\{\rho \mid \rho \models^{\star} \Phi\}) = \sum_{s \in S} \Pr^{\mathcal{C}}(\{\rho \mid \rho \models^{\star} \Phi\} \mid \{\rho \mid \rho@T = s\}) \cdot \Pr^{\mathcal{C}}(\{\rho \mid \rho@T = s\}),$$

since $\sum_{s \in S} \Pr^{\mathcal{C}}(\{\rho \mid \rho@T = s\}) = 1$. Observe that:

$$\begin{aligned}
\Pr^{\mathcal{C}}(\{\rho \mid \rho \models^{\star} \Phi\} \mid \{\rho \mid \rho@T = s\}) &= \\
&= \frac{\Pr^{\mathcal{C}}(\{\rho \mid \rho \models^{\star} \Phi\} \bigcap \{\rho \mid \rho@T = s\})}{\Pr^{\mathcal{C}}(\{\rho \mid \rho@T = s\})} \\
&= \frac{\Pr^{\mathcal{C}}\{\{\rho \mid \forall i.\rho[0..i] \models \int 1 \leqslant T \to \varphi \text{ and } \rho@T = s\}\}}{\Pr^{\mathcal{C}}(\{\rho \mid \rho@T = s\})} \\
&= \frac{\Pr^{\mathcal{C}[s]}(\{\rho \mid \rho \models^{\star}_{\{s\},T} \varphi)}{\Pr^{\mathcal{C}}(\{\rho \mid \rho@T = s\})}.
\end{aligned}$$

Note that, for the last step, we use Lemma 4.3.21 and Definition 4.3.5. It follows that:

$$\begin{aligned}
&\Pr^{\mathcal{C}}(\{\rho \mid \rho \models^{\star} \Phi\}) \\
= &\sum_{s \in S} \frac{\Pr^{\mathcal{C}[s]}(\{\rho \mid \rho \models^{\star}_{\{s\},T} \varphi)}{\Pr^{\mathcal{C}}(\{\rho \mid \rho@T = s\})} \cdot \Pr^{\mathcal{C}}(\{\rho \mid \rho@T = s\}) \\
= &\sum_{s \in S} \mathit{Prob}(\mathcal{C}[s] \models^{\star}_{\{s\},T} \varphi).
\end{aligned}$$

This completes the proof. ■

The solution boils down to the computation of $\mathit{Prob}(\mathcal{C}[s] \models^{\star}_{\{s\},T} \varphi)$ for each state s, for which we can apply Algorithm 9 for approximations. A detailed description is given in Algorithm 11.

Complete algorithm for the time-bounded verification of IDP

The general algorithm for computing time-bounded IDP verification is given in Algorithm 11.

Algorithm 11 Compute $\widetilde{Prob}(\mathcal{C} \models^{\star} \Phi)$

Require: A CTMC $\mathcal{C}$, an LDP Φ and ε

1: $Prob = 0$
2: Chose $N \geq \Lambda T e^2 + \ln\left(\frac{|S|\cdot|\sqrt{G}|}{\varepsilon}\right)$
3: **for** $s \in S$ **do**
4: $\quad Prob+ = \widetilde{Prob}_N(\mathcal{C}[s] \models^{\star}_{\{s\},T} \varphi)$
5: **end for**
6: **return** *Prob*

A bound on the approximation error of Algorithm 11

We also have the following error bound.

Theorem 4.3.23 *Given ε and $N \in \mathbb{N}$, it holds that*

$$Prob(\mathcal{C} \models^{\star} \Phi) - \widetilde{Prob}(\mathcal{C} \models^{\star} \Phi) < \varepsilon.$$

Proof For each s, we compute $Prob(\mathcal{C}[s] \models^{\star}_{\{s\},T} \varphi)$ up to $\frac{\varepsilon}{|S|\cdot\sqrt{|G|}}$. Namely, we choose N such that $N \geq \Lambda T e^2 + \ln\left(\frac{|S|\cdot\sqrt{|G|}}{\varepsilon}\right)$. It follows that

$$Prob(\mathcal{C} \models^{\star} \Phi) - \widetilde{Prob}(\mathcal{C} \models^{\star} \Phi) \leqslant |S| \cdot \frac{\varepsilon}{|S|} \leqslant \varepsilon.$$

This completes the proof. ■

4.4 Summary

In this chapter we have introduced model checking algorithms for CTMCs against multiple property specification formalisms, such as MTL (see Section 4.1), Timed Automata (see Section 4.2) and LDPs (see Section 4.3). Note that all the above mentioned problems can be solved using variants of the algorithm described in Section 4.1. In all the cases, the key idea is to generate an appropriate set of linear constraints over the variables that determine the residence time of each state along the discrete path under consideration. Then, the problem reduces to computing multidimensional integrals over those sets of linear constraints. We reuse a Matlab [MAT13] implementation, mentioned at the end of Section 4.1, of the algorithm of [LZ01] to compute multidimensional integrals over sets of linear constraints.

The verification of LDPs and TAs over CTMC share the same drawbacks as the verification of MTL formulas over CTMCs, namely

1. We need to enumerate all the possible paths of length up to N and this is exponential in N;

2. The Matlab implementation of the algorithm of [LZ01] runs quite slowly due to the use of the symbolic toolbox.

Thus, improving the two bottlenecks above would speed up the verification time for all the classes of problems analysed in this chapter, i.e. MTL, TAs and LDPs. As described at the end of Section 4.1, one possible solution would be to eliminate the use of symbolic variables from the Matlab implementation and to avoid having to enumerate all the paths of length up to N via random algorithms.

Generating sets of linear constraints and solving multidimensional integrals over those sets is a powerful technique. In fact, we can always use sets of linear constraints to characterise the validity of a property in a timed path, independently from the formalism chosen or the model considered, as long as the property under consideration can be related to the residence time in system states. This technique is a central focus of this thesis, since it becomes a valuable tool to solve a vast class of CTMC-related problems, and beyond. In fact, in the next chapter, Chapter 5, we show how similar techniques can be applied to synthesising model parameters for networks of HIOAs (see Section 5.2.2 for the definition of HIOAs and network of HIOAs). Although the problem that we tackle is different from model checking, i.e., parameter synthesis, and the systems that we analyse seem to be incomparable with CTMCs, HIOAs also allow for continuous dynamics and the solution techniques will share many similarities with the algorithms presented in this chapter.

Chapter 5

A framework for the verification of real-time properties of medical devices

The general aim of this chapter is to provide a model-based framework supporting the formal verification and validation of medical devices, with particular emphasis on cardiac pacemakers. The components of the framework are: a model of the human heart, a model of the pacemaker and a property specification to check. The pacemaker is modelled as network of Timed Automata (see Section 5.2.1), whereas the human heart is modelled either as network of Timed Automata or as network of Hybrid Automata (see Section 5.2.2). The framework can be instantiated with personalised heart models in which parameters can be learnt from real data. We use specific real-time properties which we call property patterns (see Section 5.5.1) and the *Counting Metric Temporal Logic* (CMTL, see Section 5.5.2) in order to specify the properties of interest, such as average beat rate of the human heart and energy consumption of the pacemaker.

We show examples of how the framework can be instantiated with hybrid human heart models and a TA pacemaker model in order to perform formal verification. However, the long term goal is to use the framework for different medical devices such as neurostimulators and defibrillators.

This chapter tackles two main verification problems. Section 5.6 addresses the model checking problem for real-time properties over two different kinds of human heart models, whereas Section 5.7 gives an algorithm to solve the parameter synthesis problem for pacemakers.

More specifically, Section 5.6 answers the following question.

Model checking problem

Input: A network of *Hybrid I/O Automata* (HIOAs) $\mathcal{N}$ and a real-time property pattern

Problem: Find the probability that the property pattern is satisfied in $\mathcal{N}$.

On the other hand, Section 5.7 addresses the question below.

Optimal parameter synthesis problem

Input: A parametric network of *timed I/O automata* TIOAs $\mathcal{N}$, a set of parameters $\Gamma = \Gamma_u \cup \Gamma_c$ composed of controllable (Γ_c) and uncontrollable (Γ_u) parameters, a *Counting Metric Temporal Logic* (CMTL) formula φ and a path length n.

Problem: Find the optimal parameter values for Γ_c for any values of parameters Γ_u with respect to an objective function $\mathcal{O}$ such that φ is satisfied on $\mathcal{N}$, if such values exist.

As mentioned in the introduction of this thesis, one feature shared across most of medical devices is that a fault in the system or in the embedded software could be dangerous. Thus, the benefits of developing a formal framework for the verification of medical devices would include: increased usability and reliability; decreased failure rate and recalls; and reduced risks to patients and users.

The chapter is based on two published conference papers [CDKM12a, CDKM13c], one unpublished report [DKM13], one journal publication [CDKM13a] and is organised as follows. We start by discussing in Section 5.1 the functioning of the human heart and its main features. We continue in Section 5.2, where we describe the formalisms used to model the human heart and the pacemaker. In Section 5.3 we introduce two formal ways of modelling the human heart as a network of HIOAs. The first method, in Section 5.3.1, involves the modelling of single heart cells as HIOAs and then connecting them to form a network. The second method, in Section 5.3.2, is instead a mapping from the electrocardiogram signal (ECG), an electric signal read from the torso of the human body, to a network of HIOAs. Section 5.4 presents a detailed Timed Automata (TA) model of the pacemaker taken from [JPM+12b]. More specifically, the pacemaker is modelled as network of *Timed I/O Automata* (TIOAs, see Section 5.2.1). Next, Section 5.5 introduces the real-time properties that we want to study. We conclude the chapter with Section 5.6, largely based on [CDKM13a], and Section 5.7 whose results are awaiting submission [DKM13], where we give algorithms for the model checking and synthesis problems, respectively.

5.1 The heart and its electrical activity

In this section we describe the working of the human heart, focusing on the *electrical conduction system* (ECS)[Nat07]. The main function of the human heart is to maintain the blood circulation of the body. This rhythmic, pump-like function is driven by muscle contractions, and in particular the contraction of the atria and ventricles which are triggered by electrical signals.

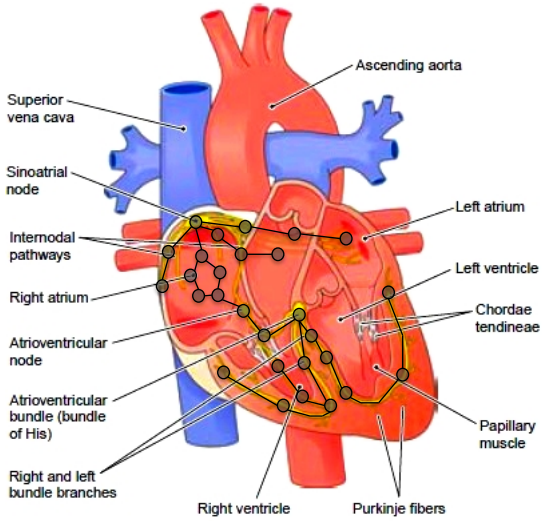


Figure 5.1: Electrical conduction system of the heart.

5.1.1 SA node

The *sinoatrial* (SA) node (a special tissue in the heart, see Figure 5.1) spontaneously produces an electrical signal, which is the *natural pacemaker* of the heart. On each heart beat, it generates the control electrical signal which is conducted through prescribed *internodal pathways* into the atrium causing its contraction. The signal then passes through the slow conducting *atrioventricular (AV) node*, allowing the blood to empty out the atria and fill the ventricles. The fast conducting *Purkinje* system spreads the electricity through the ventricles, causing all tissues in both ventricles to contract simultaneously and to force blood out of the heart. At the cellular level, the electrical signal is a change in the potential across the cell membrane, which is caused by the flow of ions between the inside and outside of the cell. It is known that sodium, potassium and calcium are the major ion species involved in this process; they flow through multiple voltage-gated ion channels. Excitation disturbances can occur in the behaviour of these ion channels at the cellular level, or in the propagation of the electrical waves at the cell network level [YEGS05].

Abnormalities in the electrical signal generation and fast or slow propagation can cause different types of arrhythmias, such as *Tachycardia* and *Bradycardia*, which require

medical intervention in the form of medication, surgery or implantable pacemakers.

5.1.2 Action potential

At the cellular level, the heart tissue is activated by an external voltage applied to the cell

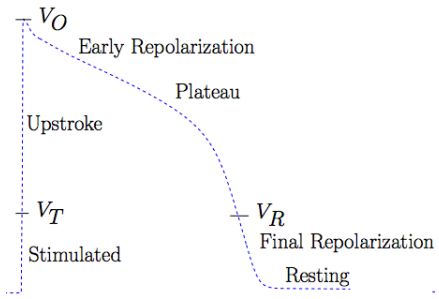


Figure 5.2: Action potential [YEGS08].

or the SA node. After the activation, a transmembrane voltage change over time can be sensed due to ion channel activities, which is referred to as an *action potential* (AP). This is also the signal that an implantable pacemaker will receive or generate. A simplified version of the *ventricular* AP (generated according to the dynamic Luo-Rudy model [YEGS08] from the Simulink implementation via Matlab [MAT13]) is shown in Figure 5.2. The AP is fired as an all-or-nothing response to a supra-threshold electrical signal, and each AP follows roughly the same sequence of events and has the same magnitude regardless of the applied stimulus. In general, APs exhibit the following major phases:

- *Stimulated.* This is the phase where the cell is triggered by a voltage spike from the AP of its neighbouring tissue or from an artificial pacing signal (pacemaker signal). However, if the current does not reach the threshold, then the cell cannot get stimulated, and consequently it goes to the resting phase.
- *Rapid upstroke.* If the received voltage spike is high enough, the upstroke indicates the depolarisation of the cell and the time when the muscle contracts.
- *Plateau and ER* (early repolarisation). This is a plateau phase during which calcium influx facilitates the muscle contraction.
- *Resting and FR* (final repolarisation). This is the last phase which features faster repolarisation that brings the potential back to the resting phase.

After the initial increase in the membrane potential, an AP lasts for a couple of hundred milliseconds (for most mammals including human beings). The early portion of an AP is known as the *effective refractory period* (ERP), due to its non-responsiveness to further stimulation, and the latter portion is known as the *relative refractory period* (RRP), during

which an altered secondary excitation event is possible if the stimulation threshold is raised. Excitations during the RRP period will produce a slightly shorter subsequent ERP period.

5.2 Models

In this section we present additional background material that is needed to understand the chapter. First, we will introduce the models used to model the human heart and the pacemaker, i.e., *Timed I/O Automata* (TIOAs) and *Hybrid I/O Automata* (HIOAs). Both models can be expressed in Matlab [MAT13] and Simulink [SIM13] (see Section 5.6.1 for details).

Before describing the model of TIOAs and the model HIOAs we need to enrich some of the definitions already introduced for TAs in Section 3.2.4 of Chapter 3.

Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of *nonnegative* real-valued variables, called *clocks*. An $\mathcal{X}$-valuation is a function $\eta : \mathcal{X} \to \mathbb{R}_{\geqslant 0}$ assigning to each variable x a nonnegative real value $\eta(x)$. Let $\Gamma = \{v_1, \ldots, v_n\}$ be a set of *nonnegative* real-valued parameters taking values respectively in the domains $\mathbf{D}_{v_1} \ldots \mathbf{D}_{v_n}$.

Definition 5.2.1 *Given a real domain* $\mathbf{D} = [l, u]$*, where* $l, u \in \mathbb{R}_{\geqslant 0}$*, we define its* δ*-discretisation to be the discrete domain of points* $\bar{\mathbf{D}} = [l, l + \delta, l + 2\delta, \ldots, l + k\delta]$ *where* $l + k\delta \leqslant u$ *and* $l + (k + 1)\delta > u$*.*

A $\Gamma-$valuation is a function $\vartheta : \Gamma \to \mathbb{R}_{\geqslant 0}$ assigning to each parameter $v \in \Gamma$ a nonnegative real value $\vartheta(v)$. Let $\mathcal{Y}$ be a set and $\mathcal{V}(\mathcal{Y})$ denote the set of all valuations over $\mathcal{Y}$. A *clock constraint* on $\mathcal{X}$, denoted by g, is a conjunction of expressions of the form $x \bowtie y$ for clock $x \in \mathcal{X}$, comparison operator $\bowtie \in \{<, \leqslant, >, \geqslant\}$ and $y \in \{\mathbb{N} \cup \Gamma\}$. We write $x \in g$, for $x \in \mathcal{X}$, if the guard g contains a constraint on clock x and $g.x := (\bowtie, y)$ with $g.x(1) = \bowtie$ and $g.x(2) = y$ if $x \bowtie y$ is a constraint of g. Let $\mathcal{B}(\mathcal{X}, \Gamma)$ denote the set of clock constraints over $\mathcal{X}$ and Γ. An $(\mathcal{X}, \Gamma)$-valuation (η, ϑ) *satisfies* a constraint $x \bowtie y$, denoted $(\eta, \vartheta) \models x \bowtie y$, if and only if $\eta(x) \bowtie y$ and $y \in \mathbb{N}$, or $\eta(x) \bowtie \vartheta(y)$ and $y \in \Gamma$; it satisfies a conjunction of such expressions if and only if η satisfies all of them.

Let $\mathbf{0}$ denote the valuation that assigns 0 to all clocks. For a subset $X \subseteq \mathcal{X}$, the reset of X, denoted $\eta[X := 0]$, is the valuation η' such that $\forall x \in X.\ \eta'(x) := 0$ and $\forall x \notin X.\ \eta'(x) := \eta(x)$. For $\delta \in \mathbb{R}_{>0}$ and $\mathcal{X}$-valuation η, $\eta + \delta$ is the $\mathcal{X}$-valuation η'' such that $\forall x \in \mathcal{X}.\ \eta''(x) := \eta(x) + \delta$, meaning that all clocks proceed at the same speed.

5.2.1 Timed I/O automata

In this section we introduce the *Timed I/O Automata* (TIOAs) model defined in [KLSV10], which we augment with parametric guards and priority on the transitions in order to impose determinism. In words, TIOAs are TAs augmented with input/output actions and

priorities. The reason why we impose deterministic behaviours in our model is because we will use TIOAs to model medical devices (see Section 5.4). Non-determinism is often viewed as an undesirable feature of medical devices, since it could lead to uncontrollable dangerous behaviours of the system. For such a reason, we tailor our model to the specific domain in which we operate and exclude non-determinism by means of prioritised transitions. We also use parametric guards on the transitions. The reason for introducing parameters is again related to the specific application domain in which we operate. Pacemakers are electronic devices. Their functional behaviour is controlled by hardware and cannot be fully specified. For instance, the hardware manufacturers, instead of giving the exact response time of the device to a given event, will specify a possible time interval in which a response from the device should be expected. Hence, there is a need for parameters that can take values in a given domain in order to cover various timing behaviours.

Definition 5.2.2 (Deterministic Timed I/O Automaton with Priority) *A deterministic timed I/O automaton (TIOA) with priority* $\mathcal{A} = (\mathcal{X}, \Gamma, Q, q_0, \Sigma_{\text{in}}, \Sigma_{\text{out}}, \rightarrow, \gamma)$ *consists of:*

- *A finite set of clocks* $\mathcal{X}$*.*
- *A finite set of real-valued parameters* $\Gamma = \Gamma_c \cup \Gamma_u$*, where* Γ_c *and* Γ_u *are respectively the set of controllable and uncontrollable parameters.*
- *A finite set of modes* Q*, with the initial mode* $q_0 \in Q$*.*
- *A finite set of input actions* Σ_{in} *and a finite set of output actions* Σ_{out}*.*
- *A transition relation* $\rightarrow \subseteq Q \times (\Sigma_{\text{in}} \cup \Sigma_{\text{out}}) \times \mathcal{B}(\mathcal{X}, \Gamma) \times 2^{\mathcal{X}} \times Q$*. For any* $q, q' \in Q$ *and clocks to reset* $X \subseteq \mathcal{X}$*, if* $a \in \Sigma_{\text{out}}$ *then* $e = (q, a, g, X, q')$ *has* $g \neq \textbf{true}$*. Also, for any* $q \in Q$ *and any two outgoing transitions of* q *with guards* $g_1, g_2 \neq \textbf{true}$*, we require that* $g_1 \cap g_2 = \varnothing$*.*
- *A priority function* $\gamma : Q \times (\Sigma_{\text{in}} \cup \Sigma_{\text{out}}) \rightarrow \mathbb{N}$ *that assigns a priority to an action in a given mode. For any* $q \in Q$*,* $a_{\text{in}} \in \Sigma_{\text{in}}$*,* $a_{\text{out}} \in \Sigma_{\text{out}}$ *and* $a_1, a_2 \in (\Sigma_{\text{in}} \cup \Sigma_{\text{out}})$ *we require* $\gamma(q, a_{\text{in}}) < \gamma(q, a_{\text{out}})$ *and* $\gamma(q, a_1) \neq \gamma(q, a_2)$*.*

Let $e = (q, a, g, X, q')$ be a transition of TIOA $\mathcal{A}$ and η a clock valuation. We say that an action a is *enabled* if either $a \in \Sigma_{\text{in}}$ or $a \in \Sigma_{\text{out}}$ and $\eta \models g$. Observe that every transition of the TIOA $\mathcal{A}$ that has an output action is *urgent*, i.e., it is taken when the guard becomes true. The TIOA in the above definition can still exhibit Zeno behaviour, but one can use the sufficient criteria in ([BK08], Lemma 9.24) to check for absence of Zenoness.

The TIOAs as defined above are able to synchronise on matching input and output actions, thus forming networks $\mathcal{N}$ of communicating automata. An example of network of TIOAs is shown in Example 5.2.1.

Similarly to Section 3.2.4, we will now define, first informally and after formally in Definition 5.2.3, timed and untimed paths of a network of TIOAs. We keep the notation introduced in Section 3.2.4 to refer to timed and untimed paths of the network.

Informally, the network $\mathcal{N}$ evolves as follows. Each component $\mathcal{A}_i$ of $\mathcal{N}$ can either:

I) have an output transition with maximum priority enabled, in which case the component fires the output transition and moves to the next mode accordingly, or

II) if no output transition is enabled then it either

(a) synchronises with an output transition fired by another component, which must have a matching input transition, or

(b) lets time pass.

Formally, keeping the labelling I), II)(a) and II)(b) introduced above, the composition is defined as follows.

Definition 5.2.3 (Network of TIOAs) *Let $\mathcal{N} = \{\mathcal{A}^{(i)} \mid i \in \{1,\dots,m\}\}$ with $m \in \mathbb{N}$ be a network of m TIOAs $\mathcal{A}^{(i)}$, $i \in \{1,\dots,m\}$. Define the set of modes of the network by $\vec{Q} = Q^{(1)} \times \cdots \times Q^{(m)}$. Let $\vartheta^{(i)}$ be a parameter instantiation for every $i \in \{1,\dots,m\}$. We say $\sigma = \vec{q_0} \xrightarrow{t_0} \vec{q_1} \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} \vec{q_n}$, $(t_j \leqslant 0, j \in \{0,\dots,n-1\})$ is the finite* timed path *of a network $\mathcal{N}$ of m TIOAs if for all $j \in \{0,\dots,n-1\}$ there exists an index set $\mathcal{I}_j \subseteq \{1,\dots,m\}$ such that:*

I) For all $i \in \mathcal{I}_j$, $(q_j^{(i)}, a_j^{(i)}, g_j^{(i)}, X_j^{(i)}, q_{j+1}^{(i)}) \in \rightarrow^{(i)}$, $\gamma(q_j^{(i)}, a_j^{(i)})$ is the maximum, with respect to the priority relation γ, over the set of enabled actions of $q_j^{(i)}$, $a_j^{(i)} \in \Sigma_{\text{out}}^{(i)}$, $(\eta_j^{(i)}+t_j, \vartheta^{(i)}) \models g_j^{(i)}$ and $\eta_{j+1}^{(i)} = (\eta_j^{(i)}+t_j)[X_j^{(i)} := 0]$, where $\eta_j^{(i)}$ is the clock valuation when entering *$q_j^{(i)}$. We define the set $\Sigma_{\text{out},j}$ to be the set of output actions $a_j^{(i)}$.*

II) For all $k \in \{1,\dots,m\} \backslash \mathcal{I}_j$:

(a) if there exists an $i \in \mathcal{I}_j$ such that $a_j^{(k)} = a_j^{(i)}$ and $a_j^{(k)} \in \Sigma_{\text{in}}^{(k)}$ then it must be the case that $(q_j^{(k)}, a_j^{(k)}, \textbf{true}, \varnothing, q_{j+1}^{(k)}) \in \rightarrow^{(k)}$, $\gamma(q_j^{(k)}, a_j^{(k)})$ is the maximum over the set of enabled actions of $q_j^{(k)}$ and $\eta_{j+1}^{(k)} := \eta_j^{(k)}+t_j$,

(b) otherwise $q_{j+1}^{(k)} := q_j^{(k)}$ and $\eta_{j+1}^{(k)} := \eta_j^{(k)}+t_j$.

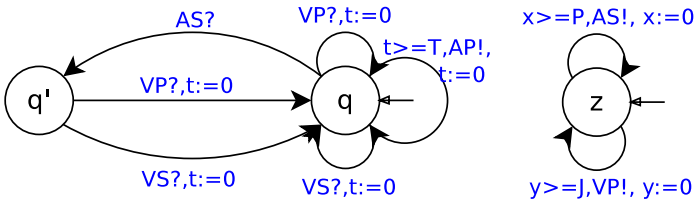
We define $\Sigma_{\text{in},j}$ to be the set of input actions $a_j^{(k)}$.

We define the set $Act_j = \Sigma_{\text{out},j} \cup \Sigma_{\text{in},j}$ of enabled actions at step j. We write $\sigma[j] := Act_j$ for $(j \in \{0,\dots,n-1\})$ and $\sigma\langle j\rangle := t_j$. Moreover, for $t \in \mathbb{R}_{\geqslant 0}$, $\sigma@t := o$, where o

is the smallest index such that $\sum_{k=0}^{o} \sigma\langle k\rangle > t$. *We can instantiate the timed path* σ *with a different sequence of times* $t'_0, \cdots, t'_{n-1}$ *by* $\sigma[t'_0, \cdots, t'_{n-1}]$.

Given a finite timed path $\sigma = \vec{q_0} \xrightarrow{t_0} \vec{q_1} \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} \vec{q_n}$ *we define* $\sigma[\![j]\!] := \vec{q_j} \to \vec{q_{j+1}} \to \cdots \to \vec{q_n}$ *to be the untimed suffix of* σ. *As for timed paths, we can instantiate an untimed path by* $\sigma[\![j]\!][t_j, \cdots, t_{n-1}] = \vec{q_j} \xrightarrow{t_j} \vec{q_{j+1}} \xrightarrow{t_{j+1}} \cdots \xrightarrow{t_{n-1}} \vec{q_n}$.

Example 5.2.1 *In Figure 5.3 we present an example of a network* $\mathcal{N}$ *composed of two TIOAs,* $\mathcal{A}_1$ *and* $\mathcal{A}_2$. *The TIOAs* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *represent an abstraction of two components of the pacemaker model described later in Section 5.4, namely, the Lower Rate Interval (LRI) component (see Figure 5.10(a)) and the Upper Rate Interval (URI) component (see Figure 5.11(a)).*



(a) $\mathcal{A}_1$ (b) $\mathcal{A}_2$

Figure 5.3: Example network $\mathcal{N}$ with two components.

Here $\mathcal{X}^{(1)} = \{t\}$, $\mathcal{X}^{(2)} = \{x, y\}$, $\Gamma^{(1)} = \{T\}$, $\Gamma^{(2)} = \{P, J\}$, $\Sigma^{(1)}_{\text{in}} = \{\mathsf{AS}, \mathsf{VS}, \mathsf{VP}\}$, $\Sigma^{(2)}_{\text{out}} = \{\mathsf{AP}\}$, $\Sigma^{(2)}_{\text{in}} = \varnothing$ *and* $\Sigma^{(2)}_{\text{out}} = \{\mathsf{AS}, \mathsf{VP}\}$. *A sample path of the network* $\mathcal{N}$ *is* $\sigma = (q, z) \to (q', z) \to (q, z) \to (q, z)$, *where for simplicity we have omitted time stamps and actions from the transitions. The initial state is* (q, z). *The automaton* $\mathcal{A}_2$ *triggers the first two transitions with the output actions* AS *and* VP, *moving the system respectively to* (q', z) *with the first action and to* (q, z) *with the second. Essentially, the automaton* $\mathcal{A}_2$ *is evolving following case I) in Definition 5.2.3. On these transitions, the automaton* $\mathcal{A}_1$ *will synchronise with* $\mathcal{A}_2$ *via matching inputs,* AS *and* VP. *Here the automaton* $\mathcal{A}_1$ *is synchronising, meaning that it is evolving following the first subcase of II) in Definition 5.2.3. The third transition of the path is instead triggered by* $\mathcal{A}_1$ *through the output action* AP. *Note that* σ *is a finite timed path, but in the example we have decided to abstract time from* σ *in order to simplify the notation.*

5.2.2 Hybrid I/O automata

We next introduce now the *Hybrid I/O Automata* (HIOAs) [LSVW95]. In short, HIOAs are TIOAs augmented with continuous variables.

We extend the definitions of clocks, valuation functions and constraints that we introduced at the beginning of this section. In particular, for HIOAs, we do not just consider only clocks but a more general set of variables. The idea is that those variables update

their values differently from clocks. Thus, valuation functions must be adapted to evaluate not only simple clocks but any general variable with its possible updates.

Let $\mathcal{X} = \{x_1, \ldots, x_d\}$ be a set of variables in $\mathbb{R}$. An $\mathcal{X}$-valuation is a function $\eta : \mathcal{X} \to \mathbb{R}$ assigning to each variable $x \in \mathcal{X}$ a real value $\eta(x)$. Let $\mathcal{V}(\mathcal{X})$ denote the set of all valuations over $\mathcal{X}$. A *constraint* on $\mathcal{X}$, denoted by grd, is a conjunction of expressions of the form $x \bowtie c$ for variable $x \in \mathcal{X}$, comparison operator $\bowtie \in \{<, \leq, >, \geq\}$ and $c \in \mathbb{R}$. Let $\mathcal{B}(\mathcal{X})$ denote the set of constraints over $\mathcal{X}$. An $\mathcal{X}$-valuation η satisfies constraint grd, denoted $\eta \models grd$, if and only if $(\eta(x_1), \cdots, \eta(x_d)) \in grd$. For $\delta \in \mathbb{R}$ and $\mathcal{X}$-valuation η, $\eta{+}\delta$ is the $\mathcal{X}$-valuation η' such that $\forall x \in \mathcal{X}.\ \eta'(x) := \eta(x){+}\delta$, which implies that all variables proceed at the same speed. Let $\mathcal{Y}(\mathcal{X})$ denote the set of all real-valued functions over $2^{\mathcal{X}}$. We define $\mathcal{L}(\mathcal{X}) := \{x := u \mid x \in \mathcal{X} \wedge u \in \mathcal{X} \cup \{0\}\}$ to be the set of *update* assignments over the set of variables $\mathcal{X}$. For an assignment $\lambda = \{x := u\} \in \mathcal{L}(\mathcal{X})$ we write $\eta[\lambda]$ to be the valuation η' such that $\eta'(x) = \eta(u)$ and $\eta'(y) = \eta(y)$ for all $y \in \mathcal{X}$ and $y \neq x$.

We impose some restrictions on the HIOAs of [LSVW95], which are described before introducing the network of HIOAs in Definition 5.2.6.

Definition 5.2.4 (Hybrid I/O Automaton) *A hybrid I/O automaton (HIOA)* $\mathcal{A} = (\mathcal{X}, Q, q_0, E_1, E_2, \mathrm{Inv}, \to, \mathrm{Diff})$ *consists of:*

- *a finite set of variables* $\mathcal{X}$*;*
- *a finite set of modes* Q*, with the initial mode* $q_0 \in Q$*;*
- *a finite set* E_1 *of* input *actions and a finite set* E_2 *of* output *actions with* $\mathcal{E} = E_1 \cup E_2$*;*
- *an invariant function* $\mathrm{Inv} : Q \to \mathcal{B}(\mathcal{X})$*;*
- *a transition relation* $\to \subseteq Q \times (\mathcal{E} \cup \{\varsigma\}) \times \mathcal{B}(\mathcal{X}) \times 2^{\mathcal{L}(\mathcal{X})} \times Q$*, where* ζ *is the internal action; and*
- *a derivative function* $\mathrm{Diff} : Q \times \mathcal{X} \to \mathcal{Y}(\mathcal{X})$ *that assigns a function to a variable* $x \in \mathcal{X}$*.*

The state space of an HIOA is $\mathcal{S} = Q \times \mathcal{V}(\mathcal{X})$. A state $s \in \mathcal{S}$ is thus a pair $s = (q, \eta)$, where $q \in Q$ is a mode and η is the continuous state denoting a valuation of all variables $\mathcal{X}$. The initial state is $s_0 = (q_0, \mathbf{0})$ where $\mathbf{0}$ is the valuation which assigns 0 to each variable. Notice that we only consider transitions with at most one input or output action. Let $\Phi : Q \times \mathcal{V}(\mathcal{X}) \times \mathbb{R}_{\geq 0} \to \mathcal{V}(\mathcal{X})$ be the flow function defined by Diff. The (unlabelled) transition relation of $\mathcal{A}$ is a set $\mathcal{J} \subseteq \mathcal{S} \times \mathcal{S}$ that defines transitions between states of $\mathcal{A}$.

Example 5.2.2 *As an example of a simple HIOA consider the HIOA* $\mathcal{A} = (\mathcal{X}, Q, q_0, E_1, E_2, \mathrm{Inv}, \to, \mathrm{Diff})$ *in Figure 5.4, where* $\mathcal{X} = \{x, \theta\}$*,* $Q = \{q_0\}$*,* $E_1 = \{\varnothing\}$*,* $E_2 = \{\mathsf{Aget}, \mathsf{Vget}\}$*,*

and Inv $= \{\varnothing\}$. *The transition relation* $\rightarrow$ *consists of two self-loops. The first self-loop is taken when the guard* $\theta = \theta_1$ *is satisfied and it outputs the action* Aget, *whereas the second self-loop is taken when* $\theta = \theta_2$ *and it outputs the action* Vget. *The variables* $\dot{x}, \dot{\theta}$ *denote the derivative function* Diff *of respectively* x *and* θ. *The system of ODEs in Figure 5.4 is used to map the electrocardiogram signal recorded from the torso of a human body to a human heart cell. The variable* x *represents the voltage amplitude of the signal and the variable* θ *the cardiac phase (see Section 5.3.2 for further details).*

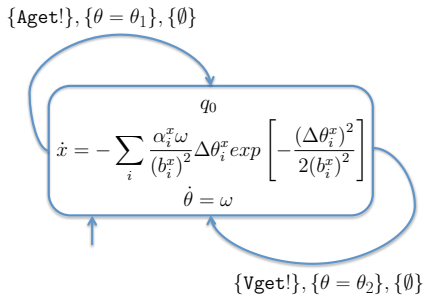


Figure 5.4: Example hybrid I/O automaton.

We now define the discrete-time simulation semantics for a hybrid automaton introduced in [AKRS08]. The main idea is to discretise the flow functions of the HIOA using an integration routine $\mathbb{S}$. There are several standard approaches that can be applied here, for instance the Runge-Kutta method which has a total accumulation error of $\mathcal{O}(h^4)$, where h is the time step. More specifically, we use $\Phi^{\mathbb{S}}$ to denote the flow function obtained by using the integration routine $\mathbb{S}$.

Definition 5.2.5 (Discrete-time semantics [AKRS08]) *Consider an HIOA* $\mathcal{A}$, *an integration routine* $\mathbb{S}$, *a time step* h *and a time bound* T. *Let* $k = \lfloor \frac{T}{h} \rfloor$. *The set of* k-step trajectories *of* $\mathcal{A}$ *consists of sequences (discrete paths) of the form* $\sigma = s_0, s'_0, \ldots, s_{k-1}, s'_{k-1}$, *where the states* $s_i = (q_i, \eta_i)$ *and* $s'_i = (q'_i, \eta'_i)$ *are defined as follows:*

- *the state* $s_0 = (q_0, \eta_0)$ *is the initial state;*
- *for each* $0 \leqslant i \leqslant k-1$, (q'_i, η'_i) *is the state after the continuous state evolution of the system from* (q_i, η_i) *with* $q'_i = q_i$ *and* $\eta'_i = \Phi^{\mathbb{S}}(q_i, \eta_i, h)$;
- *for each* $0 \leqslant i \leqslant k-2$, $((q'_i, \eta'_i), (q_{i+1}, \eta_{i+1})) \in \mathcal{J}$.

We define $\sigma[i]$ to be the i-th state in σ, and $|\sigma|$ to be the length of the discrete path, i.e., the number of states $s_i \in \sigma$.

We use a *network of HIOAs* for the composition of more than one HIOA. The (discrete-time simulation) semantics of a network of HIOAs is the same as for a single HIOA. In order to obtain a deterministic network we impose some restrictions on HIOAs as follows:

- they must be *input enabled*, meaning that, for each mode and each input action, there is an edge labelled by the input action;

- the output actions have the highest priority, meaning that they are always *urgent*, i.e., if at any state the output action is enabled, the system must execute that action;

- the input actions are never enabled unless the corresponding output actions from the environment synchronise with them: once they can be synchronised, they are urgent;

- for each mode, there is a self-loop labelled by the internal action.

Definition 5.2.6 (Network of Hybrid Automata) *Let m be the number of HIOAs in the network. A state of the network is $\left((q^{(1)}, \eta^{(1)}), \cdots, (q^{(m)}, \eta^{(m)})\right)$. There is a transition*

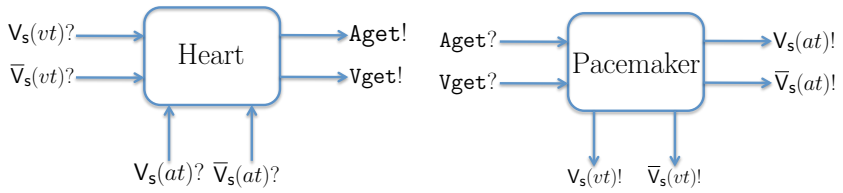
$$\left((q_i^{(1)}, \eta_i^{(1)}), \ldots, (q_i^{(m)}, \eta_i^{(m)})\right) \to \left((q_{i+1}^{(1)}, \eta_{i+1}^{(1)}), \ldots, (q_{i+1}^{(m)}, \eta_{i+1}^{(m)})\right),$$

where

- *either, for each $1 \leqslant k \leqslant m$, $(q_i^{(k)}, \eta_i^{(k)})$ has a continuous evolution;*

- *or, for each $1 \leqslant k \leqslant m$, $(q_i^{(k)}, \eta_i^{(k)})$ has a discrete transition. If, for some k, $(q_i^{(k)}, \eta_i^{(k)})$ enables an output action $a \in E_2^{(k)}$, then all of the other $(q_i^{(k')}, \eta_i^{(k')})$ must take a corresponding input action $a \in E_1^{(k')}$ (notice that this is guaranteed by input enabledness, the first of the restrictions that we have previously introduced); otherwise, each state evolves by taking the internal action.*

Example 5.2.3 *Figure 5.5 shows an example network of HIOAs. The network is composed of two HIOAs, $\mathcal{A}_1$ and $\mathcal{A}_2$, which have been labelled respectively as* Heart *and* Pacemaker*. The HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ represent respectively an abstract model of the human heart and a model of a pacemaker. $\mathcal{A}_1$ and $\mathcal{A}_2$ communicate via input and output actions which are marked by ? and !, respectively. $\mathcal{A}_2$ communicates with $\mathcal{A}_1$ through four output actions, $\mathsf{V_s}(at)!$, $\overline{\mathsf{V}}_\mathsf{s}(at)!$, $\mathsf{V_s}(vt)!$ and $\overline{\mathsf{V}}_\mathsf{s}(vt)!$. $\mathcal{A}_1$ communicates with $\mathcal{A}_2$ using two output actions* `Aget`*! and* `Vget`*!. An example of execution of the network might be the following. First the heart component, $\mathcal{A}_1$, outputs the action* `Aget`*!. The pacemaker component, $\mathcal{A}_2$, synchronises with that action via the input action* `Aget`*?. After some time interval of not receiving any event from the heart, the pacemaker decides to deliver a stimulus and outputs the beginning of the stimulus action $\mathsf{V_s}(vt)!$. The heart then synchronises with the action $\mathsf{V_s}(vt)?$. When the stimulus terminates, the pacemaker outputs the action $\overline{\mathsf{V}}_\mathsf{s}(vt)!$ and the heart synchronises with its respective action $\overline{\mathsf{V}}_\mathsf{s}(vt)?$.*



(a) $\mathcal{A}_1$. (b) $\mathcal{A}_2$.

Figure 5.5: Example of a network of HIOAs.

5.3 Hybrid heart models

In this section we present two hybrid human heart models that we will use to validate our framework. The two hybrid heart models in Section 5.3.1 and Section 5.3.2 share multiple similarities: they both model the ECS (see Section 5.1) of the heart as a network of HIOAs. However, there are some advantages and disadvantages that users should consider when choosing one model over the other. The cardiac cell heart model in Section 5.3.1 can be very accurate according to the number of cells that one decides to model. Of course, such accuracy comes at a price: the system becomes substantially more complex when one increases the number of cells. Moreover, each of those cells contains a number of parameters which characterise the cell behaviour. It is quite hard to estimate the cell parameters directly. On the other hand, the ECG model in Section 5.3.2 is simpler and more tractable than the cellular heart model since it uses a smaller number of cells to model the human heart. Section 5.3.2 presents also a mapping from ECG signals to action potential which allows one to switch from an ECG model to a cardiac cellular model. Although the ECG model is a less precise abstraction of the human heart, it has some advantages. ECG signals are easy to record and monitor. Thus, it is possible to collect patient-specific ECG data and map those onto a cellular heart model. In this way, one can obtain a patient-specific cardiac heart model.

5.3.1 The cardiac cell heart model

The first heart model is based on modelling the ECS of the heart. The ECS is a network of nerves whose role is to propagate the action potential (AP) through the heart tissue.

Modelling every single cell of the ECS is computationally intensive. Thus, we abstract the conduction system as a network of cells in order to achieve a good trade-off between the complexity of the model and the running time of the experiments. We choose to connect each cell to neighbouring cells, forming a graph of 33 cells as shown in Figure 5.1 of Section 5.1. The ECS of the heart consists of conduction pathways with different *conduction delays*. Cells are connected by pathways. The delays of the pathways depend on the physiology of the tissue considered. Moreover, it is possible to use the pathway

delays to model various known tissue diseases.

Our model consists of the SA node, whose role is to generate sequences of AP signals which are propagated through the ECS of the heart, and 32 cells that share similar properties.

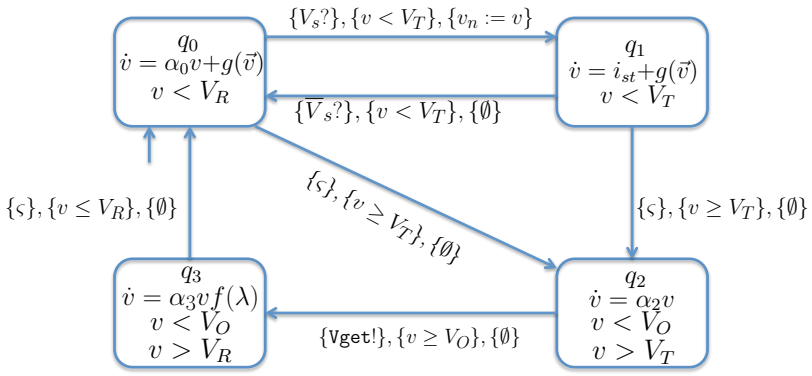


Figure 5.6: Hybrid automaton for a ventricular cardiac cell.

The cell model in Figure 5.6, taken from [YEGS05], consists of four (discrete) modes, each associated with an AP phase: *resting and final repolarisation* (q_0), *stimulated* (q_1), *upstroke* (q_2), and *plateau and early repolarisation* (q_3). The variables of the model are: v, which is the membrane voltage to control mode switches; a restitution-related variable v_n, which is used to modify the next ERP phase upon a new round of excitation; and i_{st}, which is the stimulus current. Notice that the variable v_n serves as the"memory". This is crucial to capture the proper response of AP to pacing frequency, which is an essential feature of cardiac excitation. Accordingly, [YEGS05] defines the parameter $\lambda = \frac{v_n}{V_R}$, where V_R is a model-specific constant called *repolarisation voltage*, and incorporates the function $f(\lambda) = 1 + 13\sqrt[6]{\lambda}$ into mode q_3 (see Figure 5.6).

Given N cells, $\vec{v}$ is the vector of all membrane voltages such that, for $i \leq N$, v_i denotes the voltage of cell i. In the model, we also have the function $g(\vec{v})$ denoting the voltage contribution from the neighbouring cells. Such voltage contribution is essential. In fact, the cells will rarely be stimulated with an external stimulus. Normally a cell is stimulated by neighbouring cells by means of shared voltage. When a cell is stimulated, it propagates its own voltage potential to the neighbouring cells. The neighbouring cells due to these voltage potentials will subsequently stimulate and propagate their own voltage to neighbours. The process continues in this way stimulating all the heart cells. The function $g(\vec{v})$ captures this phenomenon. Let $N-1$ be the total number of cells connected to the current cell k. The function $g_k(\vec{v})$ for the k'th cell is defined as:

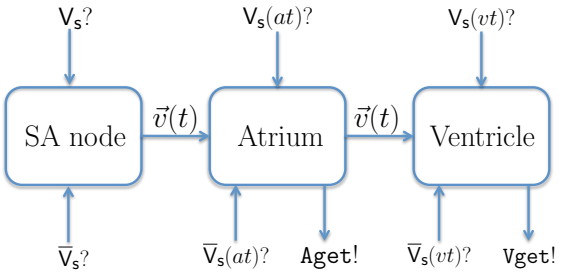
$$g_k(\vec{v}) = \sum_{i=1, i \neq k}^{N} v_i(t - \delta_{ki}) \cdot a_{ki} - v_k \cdot d_k, \tag{5.1}$$

where a_{ki} is the gain applied to the potential v_i from cell i, δ_{ki} is the time it takes for the

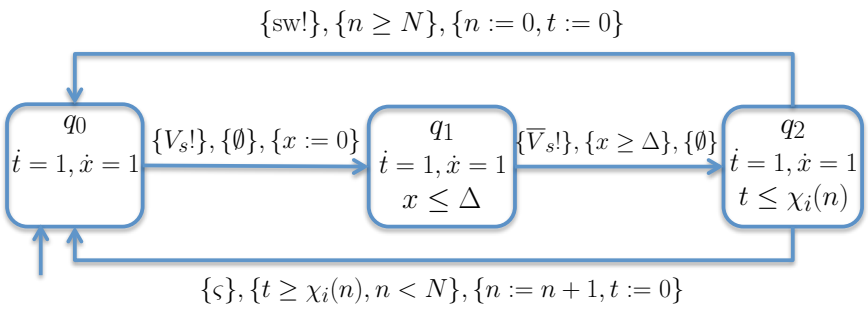
potential to reach cell k, and d_k is the distance coefficient. These coefficients depend on the conduction system, and in particular the conduction delays.

The mode invariants, defined according to Definition 5.2.4, are given by linear inequalities describing the AP. They depend on three model-specific constants: threshold voltage V_T, overshoot voltage V_O, and repolarisation voltage V_R. Initially, the cell starts in q_0. When (externally) stimulated by the event $\mathsf{V_s}?$ (input action), it enters the *stimulated* mode (q_1) and updates its voltage according to the stimulus current (i_{st}). Upon termination of the stimulation, via event $\overline{\mathsf{V}}_\mathsf{s}?$ (input action), with a sub-threshold voltage, the cell returns to resting without firing an AP. If the stimulus is supra-threshold, i.e., $v \geq V_T$ holds, the excited cell will generate an AP by progressing to mode *upstroke* (q_2). From the *upstroke* mode the cell will go to the *plateau and ER* mode (q_3) generating the event `Vget!` (output action). Then the cell transitions to mode *resting and FR* (q_0).

In Figure 5.7(a) we depict three blocks representing the connection of cells in the ECS. The atrium block consists of 14 cells and the ventricle block consists of 18 cells. The connections of these cells are illustrated in Figure 5.1, where atrium cells are shown in the upper part of the figure, while ventricle cells are in the lower part. Every cell in the atrium and the ventricle can be stimulated by the pacemaker (see Section 5.4) using the input actions $\mathsf{V_s}(at)?$, $\overline{\mathsf{V}}_\mathsf{s}(at)?$ and $\mathsf{V_s}(vt)?$, $\overline{\mathsf{V}}_\mathsf{s}(vt)?$, respectively. The output actions `Aget!` and `Vget!` notify the pacemaker that the AP in the atrium and the ventricle (where the pacemaker leads are inserted) have reached a given threshold. The function $\vec{v}(t)$ is the output voltage from a given cell.



(a) Conduction system.



(b) SA node stimulation automaton.

Figure 5.7: Electrical conduction system (ECS) model.

Recall that the SA node is the self-firing cell of the heart, stimulated by the central nervous system. The frequency of the stimulation of the SA node is given by the hybrid automaton depicted in Figure 5.7(b). The main parameter of this automaton is the function $\chi_i(n)$, $i{\in}\{1,2,3\}$, which represents the RR-series. Here the index i denotes three different types of RR-series corresponding to Normal, Tachycardia and Bradycardia heart rhythms. Although we have decided here to represent three heart behaviours, nothing stops us to introduce more. For example, one could model different Bradycardia modes, say a soft and a severe one, as well as different Tachycardia modes. The hybrid automaton in Figure 5.7(b) starts in mode q_0. Then it moves to mode q_1 by generating an output action $\mathsf{V_s}!$ representing the start of the stimulus. In mode q_1 the automaton waits for Δ units of time (which denotes the width of the stimulus) before moving to mode q_2 and generating the end of the stimulus action $\overline{\mathsf{V}}_\mathsf{s}!$. In mode q_2 the automaton waits for at most $\chi_i(n) - \Delta$ units of time. There are two possible transitions from mode q_2. When $n < N$, the value of n is incremented and a new value of the RR-series is chosen. When the automaton reaches the end of the RR-series, i.e., $n \geq N$, a new type of RR-series is picked by generating the action $\mathsf{sw}!$.

5.3.2 The ECG heart model

The heart model that we describe in this section was developed by Clifford *et al.* [CNS10] and it is based on electrocardiogram (ECG) rhythms. In words, an ECG signal is an electric signal recorded from the surface of the human chest. With ECG signals it is possible to track the activity of the human heart and due to its non-invasiveness it is widely used by medical doctors as the first examination to detect arrhythmias. ECG signals are a qualitative over-approximation of the electrical activity inside the human heart, i.e., the ECG signal is the supercomposition of all the electric signals of single cells.

An example ECG is given in Figure 5.8.

ECG signals follow roughly the same cycle in each heart beat. Typically, an ECG signal is composed of three main waves, P, QRS and T. The P wave denotes the *atrial depolarisation*. The QRS wave reflects the rapid *depolarisation of the right and left ventricles*. The T wave denotes the *repolarisation of the ventricles*. [CNS10] presents a mathematical model for generating ECGs based on a system of nonlinear ODEs, which is given as follows:

$$\dot{\theta} = \omega, \quad \dot{x} = -\sum_i \frac{\alpha_i^x \omega}{(b_i^x)^2} \Delta\theta_i^x exp\left[-\frac{(\Delta\theta_i^x)^2}{2(b_i^x)^2}\right]. \tag{5.2}$$

Here α_i^x and b_i^x, respectively, are the amplitude and width of the Gaussian functions used to model the ECG, $\theta \in [-\pi, \pi]$ is the *cardiac phase*, $\Delta\theta_i^x = (\theta - \theta_i^x) \bmod 2\pi$, and $\omega = \frac{2\pi h}{60\sqrt{h_{av}}}$ is the *angular velocity*, where h is the *instantaneous (beat-to-beat) heart rate* in BPM and h_{av} is the mean of the last n heart rates (typically with $n = 6$) normalized by 60 BPM.

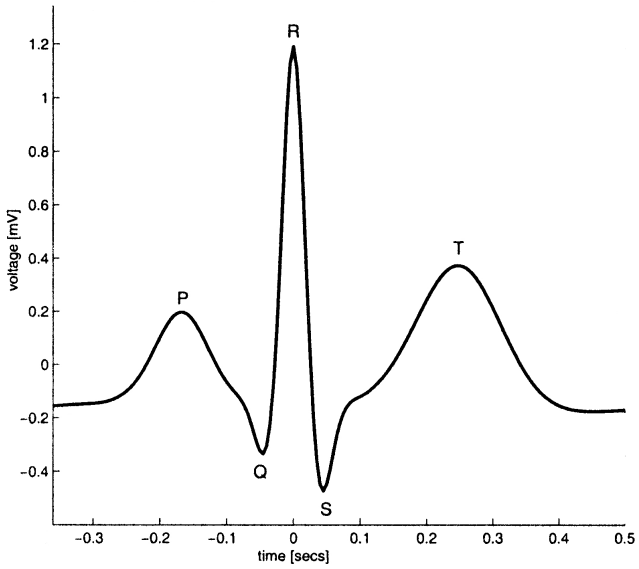


Figure 5.8: Example electrocardiogram [MCTS03].

To use Equation (5.2) one has to define the instantaneous (beat-to-beat) heart rate function $h(t)$ ($t \in \mathbb{R}_{\geq 0}$), which specifies the distance between two consecutive R-events (highest peak in Figure 5.8). Technically, it is equivalent to the so called *RR-series* $\chi(n)$, $n \in \{1, \ldots, N\}$, where N denotes the length of the series. The value of $\chi(n)$ denotes the time between two consecutive heart beats. The RR-series can be generated by constructing the power spectrum $S(f)$ as a sum of two Gaussian distributions

$$S(f) = \frac{\sigma_1^2}{\sqrt{2\pi c_1^2}} e^{\left(\frac{(f-f_1)^2}{2c_1^2}\right)} + \frac{\sigma_2^2}{\sqrt{2\pi c_2^2}} e^{\left(\frac{(f-f_2)^2}{2c_2^2}\right)},$$

which have means f_1 with power σ_1, f_2 with power σ_2 and standard deviations c_1 and c_2 respectively. The RR-series $\chi(n)$ is obtained by taking the inverse Fourier transform of $S(f)$. More details on the construction of the function $h(t)$ can be found in [MCTS03].

Mapping from ECG to action potential

We show how to map from an ECG to the atrium and ventricle AP signals. In our framework, the mapping can be implemented as a hybrid automaton which is depicted in Figure 5.9. The hybrid automaton in Figure 5.9 was previously shown as example of HIOA in Section 5.2.2.

Here θ_1 represents the beginning of the P wave, whereas θ_2 represents the beginning of the Q wave. By this mapping we can create a model consisting of two cardiac cells (one for the atrium and one for the ventricle) in which the propagation delay δ (see Equation (5.1)) is proportional to the angular velocity ω. We do not use the value of the $x(t)$ function in the current analysis based on the ECG heart model. However, the function of $x(t)$ is crucial when one wants to define a patient-specific heart model. In this case, from the

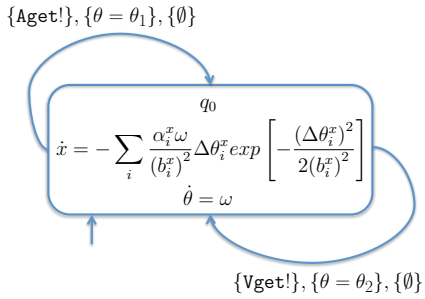


Figure 5.9: ECG hybrid automaton.

given patient ECG data one can learn the parameters of the function $x(t)$ [CAM06]. This is one of the advantages of the ECG model compared to the cardiac cell model.

We would like to point out here that the mapping described in this section is illustrative. We are aware that the so called *"inverse problem"*[PCN+10, MB98], namely reconstructing the electrical activity of single cells from the ECG, is a hard problem. Although it is well established that the P wave denotes the atrial depolarisation, the QRS wave reflects the rapid depolarisation of the right and left ventricles and the T wave denotes the repolarisation of the ventricles, we have no experimental evidence to believe that our mapping is the right one. A deeper study of the mapping from ECG to the electrical activity of single cells is needed. It is true, though, that, once the correct mapping is developed, one can use it as input to instantiate our framework, with a similar procedure to the one presented in this section.

5.3.3 Switching between different heart behaviours

The two heart models described in Section 5.3.1 and Section 5.3.2 can exhibit only a single heart behaviour, such as Normal, Bradycardia or Tachycardia. However, a real human heart exhibits several spontaneous behavioural changes. We consider three of them:

1. the malfunction of the SA node;
2. the malfunction of certain cardiac cells; and
3. the malfunction of the conduction system.

Let χ_i be the set of RR-series, Distr($\{1,2,3\}$) be a discrete probability distribution which assigns a probability value to each element of the set $\{1,2,3\}$, let $\alpha \in$ Distr($\{1,2,3\}$) be an initial distribution and let $\mathbf{P}_i \in$ Distr($\{1,2,3\}$), for $i \in \{1,2,3\}$, denote the transition probabilities between different heart modes. The malfunctioning of the SA node can be modelled through Algorithm 12. Algorithm 12 iterates M times the following procedure:

- pick a new RR-series according to a given probability distribution (Line 4 of Algorithm 12);

- update the current step and heart mode behaviour index (Line 5 of Algorithm 12);
- use the new RR-series (Line 6 of Algorithm 12).

Algorithm 12 Mode switching algorithm

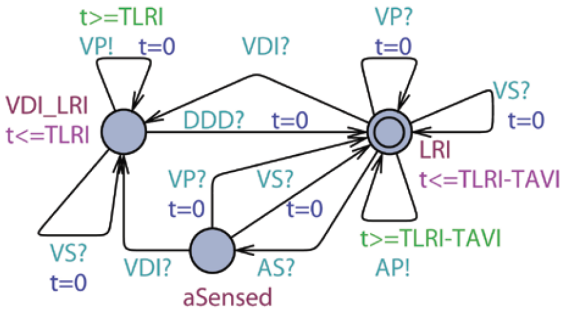
```
1: Pick i∈{1,2,3} according to α;
2: while k < M do
3:   if sw? then
4:     Pick a j according to P_i;
5:     i := j; k := k + 1;
6:     Use χ_i as the new RR-series (see Figure 5.7(b));
7:   end if
8: end while
```

We remark that the initial distribution and the transition probabilities can be learned from patient data [LB13], although we have not used personalised patient data. The learning algorithms of [LB13] were developed after the experiments presented in this thesis. The other heart malfunctions can be modelled by varying the cell model parameters.

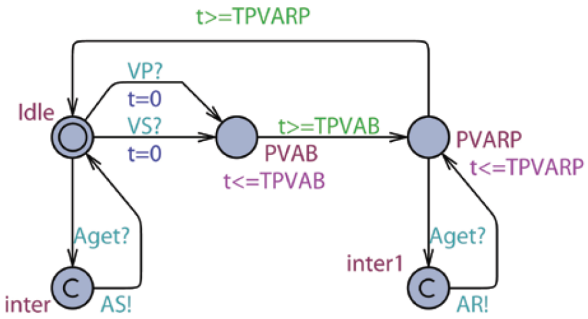
5.4 Pacemaker model

The pacemaker is implanted under the chest skin and sends impulses to the heart at specific time intervals. In most cases the pacemaker comes implanted with two leads: one for the atrium and one for the ventricle. Each lead has the ability to sense or deliver an electrical signal. As mentioned in the introduction of this chapter, the authors in [JPM+12b] develop a pacemaker model based on Timed Automata (TAs) which we use to validate our framework. For completeness of the presentation, below we reproduce this model and its variants developed for our framework.

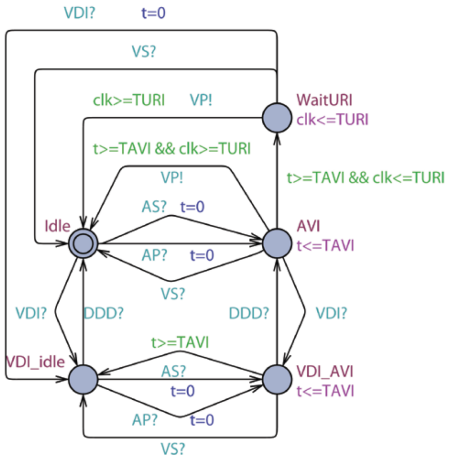
The pacemaker model in [JPM+12b] consists of five basic TA components (Figure 5.10, Figure 5.11(a)) and additional three advanced components (Figure 5.11(b), Figure 5.11(c)). The basic components are: the lower rate interval (LRI) component, the atrio-ventricular interval (AVI) component, the upper rate interval (URI) component, the post ventricular atrial refractory period (PVARP) component and the ventricular refractory period (VRP) component. The LRI component (see Figure 5.10(a)) has the function of keeping the heart rate above a given minimum value. The AVI component (see Figure 5.10(c)) has the purpose to maintain the synchronisation between the atrial and the ventricular events. An event is when the pacemaker senses or generates an action. The AVI component also defines the longest interval between an atrial event and a ventricular event. The PVARP component (see Figure 5.10(b)) notifies all other components that an atrial event has occurred. Notice that there is no AR signal in the PVARP and Interval components as we



(a) LRI component



(b) PVARP component



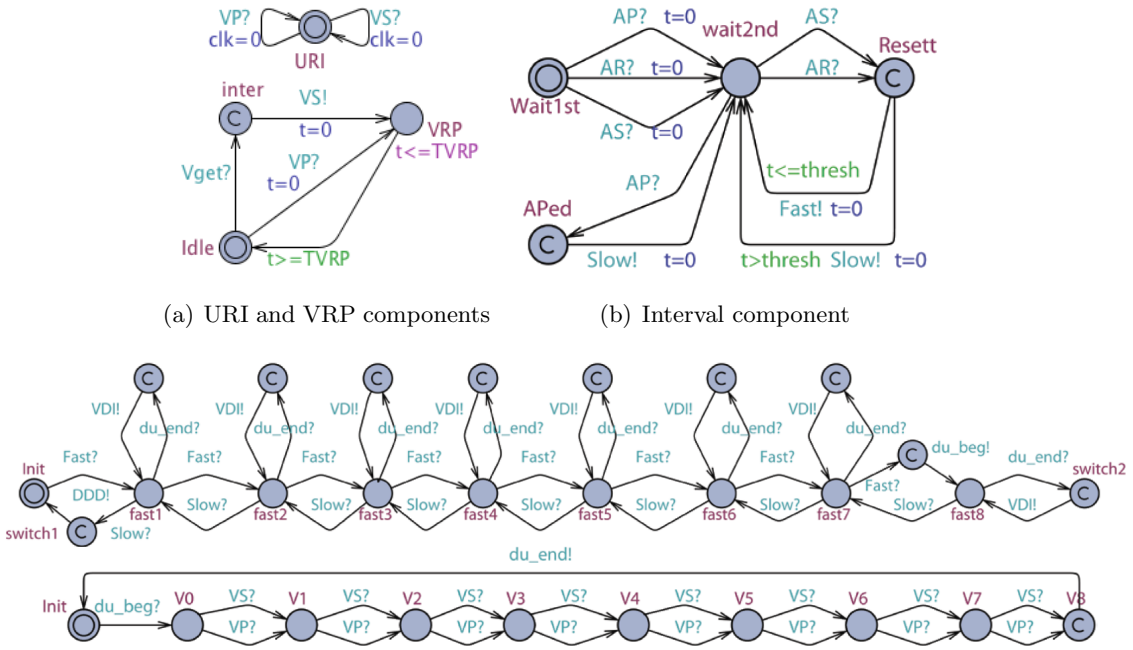
(c) AVI component

Figure 5.10: LRI, PVARP, AVI components used for basic analysis

are not using the advanced algorithms given in [JPM+12b]. The URI component (see Figure 5.11(a)) sets a lower bound on the times between consecutive ventricular events. The VRP component (see Figure 5.11(a)) filters noise and early events that may cause undesired behaviour.

The advanced components, Interval, Counter and Duration, are used for detection and correction of pacemaker mediated Tachycardia. The components switch the functioning modes of the pacemaker from DDD (pacing and sensing of the atrium and ventricle) to VDI (pacing and sensing only the ventricle). Note that in all components the locations labelled with **C** do not allow time to elapse.

There are four actions in the pacemaker model that will be considered in the remainder of the thesis. The input actions Aget? and Vget? will notify the pacemaker when there is an AP from the atrium or from the ventricle, respectively. The output actions AP! and VP! are responsible for pacing the atrium and the ventricle, respectively. After receiving an AP? event, the component in Figure 5.12 generates a cell stimulus of duration Δ_a using the two actions $\mathsf{V_s}(at)!$ and $\overline{\mathsf{V}}_\mathsf{s}(at)!$. The longer the stimulus duration is, the more likely the cell is to be stimulated. If the stimulus is too short the cell will not be stimulated.



(a) URI and VRP components (b) Interval component

(c) Counter and Duration component

Figure 5.11: URI, VRP components used for basic analysis. Interval, Counter and Duration components used for PMT analysis

There is a similar component for the ventricle stimulus generation.

5.4.1 Enhanced pacemaker model

Although the pacemaker model of [JPM+12b] is a fairly detailed reproduction of the algorithms that run inside a real pacemaker device, it is still an over approximation of the reality. Real pacemakers are far more complex. They are subject to noise, failures to sense physiologically relevant events, hardware faults and many other forms of uncertainties that are not captured in the model of [JPM+12b]. Think, for example, of the following situation.

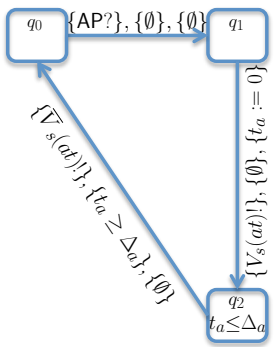


Figure 5.12: Atrium pacing.

In real pacemakers, it is not uncommon that the device fails to sense a natural heart beat, namely the human heart beats but, due to the noise, the beat is lost and hence not received by the pacemaker. The pacemaker will then believe that a beat is missing and will deliver an electric impulse in order to stimulate the heart. Such a stimulus is unnecessary and could, in extreme cases, lead to dangerous heart behaviours. The aforementioned example cannot be modelled with the pacemaker of [JPM+12b].

Even though it was not mentioned before, energy is a crucial feature of pacemaker devices. The battery lifetime of a pacemaker will determine when a new surgery, in order to change the battery, will be needed. It is evident that having the possibility of analysing how the battery of the pacemaker depletes would enrich the trust and understanding that we have of the device. Unfortunately, the model of [JPM+12b] does not include the possibility to model the energy consumption of the pacemaker.

For such reasons, in this section we enhance the pacemaker model in [JPM+12b] by considering noise and energy consumption.

Pacing noise

One of the important design issues of pacemakers is the need to tolerate noise. For instance, when the pacemaker tries to deliver a beat, the beat could be lost due to noise on the channel. As mentioned earlier, for the pacemaker model presented in [JPM+12b] the assumption is that the pacemaker can pace the heart perfectly. This can simplify the modelling considerably, but is not realistic.

We remedy this by introducing the so called "failure-to-capture", which in practice is equivalent to insufficient contact between the lead and the myocardium [GJM93]. In order to model the failure to capture situation, we add to the fixed stimulus current that the pacemaker delivers, i_{st} (cf. Figure 5.6), a *normally distributed noise* with mean μ and variance σ^2 each time the pacemaker wants to pace the cell. The result is the following. If the noise added to the channel is too high, the stimulus from the pacemaker will *not* be high enough to stimulate the cell. In such case that stimulus is considered "missing" and the cell will not be stimulated. Although this is a first improvement to the pacemaker model of [JPM+12b], we would like to remark here that there exist different causes of noise that affect pacemakers, for instance lead displacement, which we have not considered here.

Energy

Pacemaker's life time is limited and is crucially dependent on the battery embedded into the devices. The pacemaker must be re-implanted when the battery depletes. Each re-implant is a new surgery and as any other surgery it carries dangers and discomforts for the patient. For such a reason, energy usage analysis is indispensable.

We use the Kinetic Battery model (KiBaM) [BVF+11] to describe energy consumption

of the pacemaker over time. The model is given as a system of ODEs:

$$\frac{dy_1(t)}{dt} = -\iota(t) + k\left(\frac{y_2(t)}{1-c} - \frac{y_1(t)}{c}\right), \quad \frac{dy_2(t)}{dt} = -k\left(\frac{y_2(t)}{1-c} - \frac{y_1(t)}{c}\right). \tag{5.3}$$

The KiBaM models the battery charge distributing it in two wells: *available-charge* $y_1(t)$ and *bound-charge* $y_2(t)$. The function $\iota(t)$ denotes the current applied to the battery. When the value of $\iota(t)$ is zero the battery enters the recovery mode, where the energy from the bound-charge well flows to the available-charge well. The recovery effect of the battery allows a nearly discharged battery to recover in a period of zero or low current by increasing its available-charge. When the current $\iota(t)$ is not zero, both $y_1(t)$ and $y_2(t)$ decrease over time. If C [Ah] (ampere-hour) is the initial *total capacity* of the battery then $y_1(0) = c \cdot C$ and $y_2(0) = (1-c) \cdot C$, where c is a fraction of the total charge. The conduction parameter k represents the flow rate of charge from the bound-charge well to the available-charge well. The battery is considered to be empty when there is no charge in the available-charge well, i.e., $y_1(t) = 0$.

We compose the KiBaM with the TA pacemaker model as follows. The pacemaker model (see Section 5.4) has in total eight components. Each of them uses $\iota_j(t)$ μA (micro-ampere), $j \in \{1, \ldots, 8\}$. The total current applied at any time to the battery will be $\iota(t) = \sum_{j=1}^{8} \iota_j(t)$, i.e., the sum of the current used by each component. When the pacemaker is in the aSensed state of the LRI component, Idle or VDI_idle states of the AVI component, Idle state of the PVARP component, URI state of the URI component, Idle state of the VRP component, Wait1st or wait2nd states of the Interval component, Init or all fast states of the Counter component, and any states except V8 of the Duration component, then the respective current $\iota_j(t)$ is zero. Technically, when the pacemaker is in the sensing mode or the idle mode, the current applied to the battery is very small, almost zero. On the other hand, when the pacemaker is pacing, counting the duration between successive pacing events or sending a signal, the current applied to the battery increases.

We would like to remark here that we do not have experimental results that show how the battery of the pacemaker depletes. It could be the case that the decision to model the pacemaker energy with the KiBaM model of [BVF+11], in the end, does not match the reality. However, once a good model of the battery of the pacemaker is developed, similar techniques to the ones presented in this chapter could be applied to analyse the energy consumption of the pacemaker.

5.5 Real-time properties

In this section we introduce property patterns and the logic *Counting Metric Temporal Logic* (CMTL) that we use to specify real-time properties.

5.5.1 Property Patterns

Property patterns are specialised real-time properties that we define in order to model check certain important safety properties of medical devices. Moreover, they are implementation driven, meaning that we defined those patterns keeping in mind that they will be implemented in Simulink. Therefore, we defined property patterns with respect to what we called a "simulation step". The simulation step is the implicit Simulink model discretisation step, the step that Simulink uses to discretise its models and for integration routines. The simulation step is a parameter of our property pattern, reflecting the fact that the user can vary the simulation step of Simulink.

We present two important property patterns which are used to analyse the pacemaker. The first one specifies the key safety property of the pacemaker, namely, whether it maintains the number of heart beats in the normal range of 60-100 beat per minute (BPM), and the second specifies whether the energy consumed by the pacemaker at a given time point is less than a specific value.

Definition 5.5.1 (Duration of a path) *Given a discrete path ς (see Definition 5.2.5) and a simulation step h, we define the* duration *of ς in milliseconds as* $\texttt{Dur}(\varsigma) = h \cdot |\varsigma|$.

Definition 5.5.2 (Heart beats) *We define the number of heart beats with respect to the cardiac cell heart model described in Section 5.3.1. Given a discrete path ς, we define the number of heart beats of ς as* $\texttt{Heart\_beats}(\varsigma) = \sum_{i=0}^{|\varsigma|-1} \mathbb{1}(\varsigma, i)$, *where $\mathbb{1}(\varsigma, i)$ is the characteristic function of transitions such that $\mathbb{1}(\varsigma, i) = 1$ if $\varsigma[i] = (q_2, \cdot)$ and $\varsigma[i+1] = (q_3, \cdot)$, and 0 otherwise. Here q_2 and q_3 refer to the upstroke and plateau modes in Figure 5.6, respectively.*

We are now ready to define "normal paths", namely, paths corresponding to normal heart behaviours.

Definition 5.5.3 (Normal path property) *Given a discrete path ς we say that ς is* normal *if, for any i, j,*

$$(\texttt{Dur}(\varsigma[i..j]) = \texttt{1 minute}) \Rightarrow (\texttt{Heart\_beats}(\varsigma[\texttt{i..j}]) \in [\texttt{60}, \texttt{100}])\,.$$

Definition 5.5.4 (Energy property) *Let ς be a discrete path, T a finite time bound, h the simulation step and V an energy level bound. We define the* time bound energy property *to be satisfied only if the following expression is true: $y_1(\lfloor \frac{T}{h} \rfloor \cdot h) \leq V \wedge T \leq |\varsigma| \cdot h$, where $y_1(\cdot)$ is computed by Equation (5.3) in Section 5.4.1.*

Intuitively, the energy property pattern checks that the energy at time T is less than or equal to V. Both the energy and normal path properties are encoded as deterministic automata, where each transition corresponds to h time units.

5.5.2 Counting metric temporal logic

In this section we define the Counting Metric Temporal Logic (CMTL). CMTL extends MTL with basic counting formulas (BCF), such that one can count the number of actions (events) in a given interval of time. We use the pointwise semantics for both BCF and CMTL (see Definition 5.5.6 and Definition 5.5.7). Counting is essential for safety properties of medical devices, but cannot be expressed in MTL. For example, the normal path property introduced in Definition 5.5.3 can be expressed only using counting. In fact, in order to check the latter property we need to count the number of beats that have happened in a given interval of time. This justifies the need for the Counting Metric Temporal Logic. We refer the reader to a survey of the differences between MTL and counting variants of MTL in [Rab10, HR06].

Definition 5.5.5 *Let ρ be a timed path. The* counting function $\#_\ell^u a$ *for an action $a \in (\Sigma_{\text{in}} \cup \Sigma_{\text{out}})$ and time points $\ell \in \mathbb{R}_{\geqslant 0}$, $u \in \mathbb{R}_{\geqslant 0} \cup \{\infty\}$, such that $\ell < u$, is defined as $\#_\ell^u a = \sum_{k=(\rho@\ell)}^{(\rho@u)-1} (a \in \rho[k])$. Here, we write $(a \in \rho[k])$ for the indicator function that returns 1 if the action "a" belongs to the k-th transition of ρ and 0 otherwise.*

Remark 5.5.1 *By abuse of notation we write $(a \in \rho[k])$ to say that the action "a" belongs to the k-th transition of ρ. Previously, in Section 3.2.2, we wrote $(p \in \rho[k])$ to say that the atomic proposition "p" belongs to the state $\rho[k]$. This may seem confusing at first. However, the former notation is used for CMTL, where we are interested in counting the number of occurrences of actions, whereas the latter notation is used for* MTL*, where we are interested in atomic propositions. Moreover, in this thesis we check* MTL *against continuous-time Markov chains and CMTL against network of Timed I/O Automata. Thus, it should be clear from the context, i.e., if we are model checking* CTMC*s or network of TIOAs, whether we refer to $(a \in \rho[k])$ with "a" as action or as atomic proposition and no confusion should arise.*

Definition 5.5.6 *A* basic counting formula *(BCF) $\mathbb{B}$ is of the form $\mathbb{B} = \sum_{j \in J} c_j \#_{\ell_j}^{u_j} a_j$, where J is a finite index set, $a_j \in (\Sigma_{\text{in}} \cup \Sigma_{\text{out}})$, $c_j \in \mathbb{Z}_{>0}$, $\ell_j, u_j \in \mathbb{R}_{\geqslant 0}$ (with the usual constraint that $\ell_j < u_j$ for all j).*

We now define our logic CMTL as an extension of MTL, where we replace atomic propositions with BCF formulas.

Definition 5.5.7 *The syntax of the* Counting Metric Temporal Logic *(CMTL) is defined inductively by*

$$\varphi ::= \mathbb{B} \bowtie b \mid \varphi \wedge \varphi \mid \neg\varphi \mid \varphi \, \mathcal{U}^{[\ell,u]} \varphi,$$

where $\bowtie \in \{>, \geqslant, <, \leqslant\}$, $b \in \mathbb{Z}$, $\ell \in \mathbb{R}_{\geqslant 0}$, $u \in \mathbb{R}_{\geqslant 0} \cup \{\infty\}$ are time points such that $\ell \leqslant u$ and φ is a CMTL formula.

The satisfaction relation for CMTL is defined over timed paths.

Definition 5.5.8 *Let σ be a finite timed path and $i \in \mathbb{N}$ be an index. We say that σ satisfies φ at i, denoted $(\sigma, i) \models \varphi$, iff*

$$
\begin{aligned}
&(\sigma,i) \models \mathbb{B} \bowtie b && \text{iff } \sum_{j\in J} c_j \sum_{k=k'}^{k''-1} (a_j \in \sigma[k]) \bowtie b \\
&(\sigma,i) \models \varphi_1 \wedge \varphi_2 && \text{iff } (\sigma,i) \models \varphi_1 \wedge (\sigma,i) \models \varphi_2 \\
&(\sigma,i) \models \neg\varphi_1 && \text{iff } (\sigma,i) \not\models \varphi_1 \\
&(\sigma,i) \models \varphi_1 \, \mathcal{U}^{[\ell,u]} \varphi_2 && \text{iff } \exists i'.\ i \leqslant i' \text{ s.t. } \sum_{k=i}^{i'} \sigma\langle k\rangle \in [\ell,u] \wedge (\sigma,i') \models \varphi_2, \\
& && \quad \forall i''.\ i \leqslant i'' < i' \wedge (\sigma,i'') \models \varphi_1,
\end{aligned}
$$

where $k' = (\sigma[\![i]\!](t_i, \cdots, t_{n-1})@\ell_j)$, $k'' = (\sigma[\![i]\!](t_i, \cdots, t_{n-1})@u_j)$, φ_1, φ_2 are CMTL formulas, and $i', i'' \in \mathbb{N}$.

We define $\Diamond^{[\ell,u]}\varphi := \mathbf{true}\, \mathcal{U}^{[\ell,u]}\varphi$ and $\Box^{[\ell,u]}\varphi := \neg\Diamond^{[\ell,u]}\neg\varphi$.

Example 5.5.1 *Consider the following CMTL formula: $\Box^{[0,2]}(\#_0^1\mathsf{VP} \geqslant 60 \wedge \#_0^1\mathsf{VP} \leqslant 120)$. It states that, for every time point t between 0 and 2, the number of VP actions in the time interval $[t, t+1]$ is between 60 and 120. Intuitively, the formula represents a sliding time window that counts the number of events in a given interval of time.*

5.6 Verification of pacemakers over hybrid heart models

In this section we present a solution to the model checking problem for pacemakers over hybrid heart models.

As stated at the beginning of the chapter, the problem can be summarised as follows.

Model checking problem

Input:	A network of *Hybrid I/O Automata* (HIOAs) $\mathcal{N}$ and a real-time property pattern
Problem:	Find the probability that the property pattern is satisfied in $\mathcal{N}$.

We describe our implementation of the pacemaker verification framework and present experimental results. First, we introduce the framework and its implementation. Next, we instantiate the framework with the cardiac cell model and the (enhanced) pacemaker model described in Section 5.4.1, and summarise the results of the verification for a broad range of properties, including known physiological scenarios that are problematic for pacemaker designs.

5.6.1 The framework

In this section we briefly describe the model-based verification framework and its implementation.

The framework has been implemented in Simulink/Stateflow [SIM13]. Simulink is a simulation based tool that allows us to analyse dynamical systems. Simulink is a graphical block diagramming tool (see Figure 5.13(a) and Figure5.13(b) for examples of Simulink blocks) interfaced with Matlab that allows users to:

1. Utilise already existing blocks for well-known functions, such as integration, derivation and channel delays.

2. Personalise blocks through the introduction of new Matlab functions created by the users.

One of the main advantages of Simulink is that it allows one to declare real-time variables as well as shared actions among components. Thus, Simulink becomes a valuable tool for the simulation of systems that are modelled as networks of TIOAs and HIOAs.

Stateflow enables the use of state machines and flow charts within a Simulink model. We used both Simulink and Stateflow in our framework but from now on we omit the word Stateflow when we refer to Simulink/Stateflow.

As mentioned earlier in the introduction of this chapter the components of the framework are: a model of the human heart, a pacemaker model and a property to check.

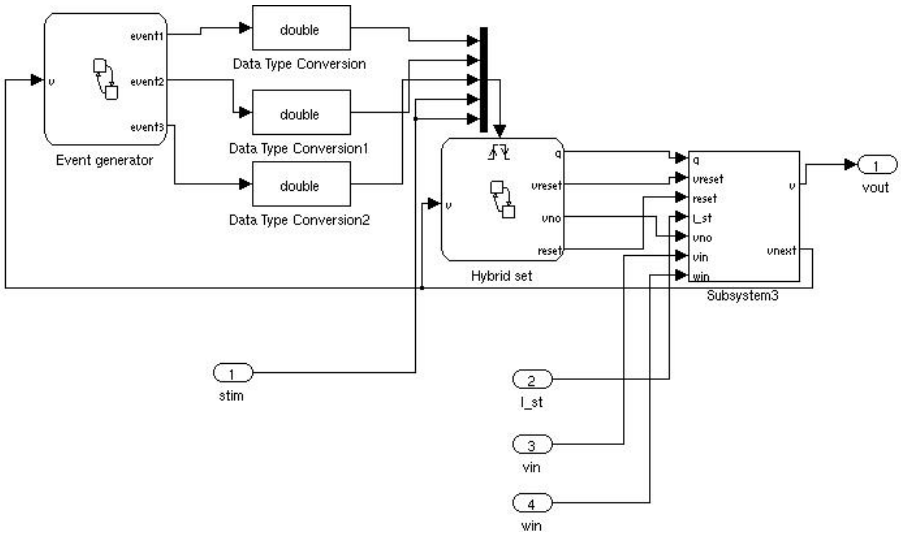
The TA pacemaker model of [JPM+12b] is translated into Simulink blocks. Each TA becomes a Simulink block. The TAs' clocks become Simulink variables and the events on the transitions become Simulink events.

We have created Simulink blocks whose role is to monitor the validity of a property pattern (see Section 5.5.1) over a simulation run of the Simulink model. The monitors flag an action if the property pattern becomes false.

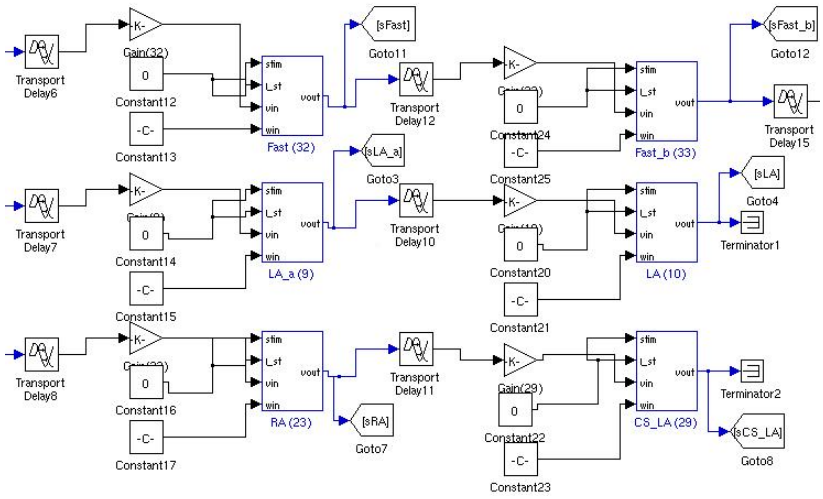
The most interesting part of the implementation is the human heart. The human heart is modelled through a network of 33 cells. Each cell is a four location HIOA (see Section 5.3.1). Cells are connected through pathway delays, i.e., Simulink delay blocks that propagate the cell voltage to its neighbouring cells according to Equation 5.1 previously introduced in Section 5.3.1.

Remark 5.6.1 *We remark here that modelling the human heart with only* 33 *cells cannot be considered a good abstraction. The human heart contains billions of cells. For such a reason, an in depth research should be conducted in order to determine a good abstraction of the human heart composed by only a finite number of cells. However, once such abstraction is found, the framework introduced in this thesis can be reused to verify real-time properties of the abstracted system.*

Figure 5.13(a) shows the Simulink implementation of the SA node. The cell is implemented by means of three Simulink blocks: *Event generator*, *Hybrid set* and *Subsystem*. The *Event generator* block is responsible to generate the input events to the cell. The *Hybrid set* implements the cell hybrid automaton model described in Figure 5.6. The *Subsystem* block performs the integration procedure to compute the voltage level of the cell. Figure 5.13(b) shows a network of six cells. Each cell block is composed from the three sub-blocks shown in Figure 5.13(a) and connected to other cells through delay and gain components.



(a) Cell block



(b) Cell connection

Figure 5.13: Simulink models

5.6.2 Approximate quantitative verification

The complexity of the heart models that we have developed, including non-linearity of the electrical signal and the large number of cells, makes automatic verification using Hybrid Automata tools infeasible. An additional complication is that we also model stochastic features, and specifically noise and probabilistic switching. We therefore employ *approximate* quantitative verification methods based on finitely many simulation runs. The derived simulation trajectories are used to estimate the probability of the satisfaction or violation of the specification expressed as a property pattern. Such a method is necessarily approximate, and so the property can only be established up to a given confidence level, but its advantage is that quantitative properties such as expected energy usage can also be handled.

There are a number of approaches that can be applied to estimate the probability of a property being satisfied based on the set of randomly generated paths according to Definition 5.2.5. These include statistical model checking based on hypothesis testing [YKNP04], Bayesian methods [ZPC10], and probability estimation [LP08]. We use the latter method, which we now explain.

In order to estimate the probabilities of a real-time property being satisfied in our model we use a simple randomized algorithm. We generate random paths in the probabilistic space underlying the model structure of depth k and compute a random variable which estimates the probability of the real-time property being true in all the paths of the system of length up to k. The approximation algorithm is good with confidence $(1-\delta)$ where δ is the confidence level.

Let $0 < \varepsilon < 1$ be the error bound, $1-\delta$ with $0 < \delta < 1$ be the confidence level, T be the time bound and h be the simulation step. Let $k = \frac{T}{h}$ be the discrete path length, p be the probability of all "normal paths", i.e., the probability of all the paths of length k satisfying the property pattern, and $N = \log(\frac{2}{\delta})/2\varepsilon^2$ be the number of random paths of length k. Let N' be the number of random paths which are "normal". By simple results from probability theory (mainly the Chernoff bound), one can show that the probability $Prob\left[\left|\frac{N'}{N} - p\right| \leq \varepsilon\right] \geq 1-\delta$, which intuitively means that, with a very high probability (i.e., $1-\delta$), the probability $\frac{N'}{N}$ that we compute is ε-close to the true probability p. We refer the readers to [LP08] for further details.

We exploit this approach in our experiments involving heart models with probabilistic switching. The probabilistic verification is carried out as follows. We consider a time bound $T = 1$ minute and a simulation step of $h = 3.7$ milliseconds, yielding paths of length $k = 16217$. We encode different heart behaviours, namely Bradycardia, Tachycardia and Normal. We allow our Simulink heart model to switch probabilistically between those different behaviours every 10 seconds. An example path could be that the heart model is beating in the "Normal" mode for 10 seconds, followed by another 10 seconds of "Normal"

mode, followed by 10 seconds of "Tachycardia", etc.. We then generate all the possible paths of length 1 minute. There are 3^{10} such paths. For each path, we check whether the property pattern that we are verifying holds or not. We then apply the approximation method described in this section in order to determine the probability of the property being true.

It should be emphasised that the verification method introduced in this section is not limited to the property patterns previously described, but applies also to other classes of properties, e.g., Metric Temporal Logic or Durational Calculus, that can be checked on paths of finite length.

5.6.3 Experimental results

We run the experiments on a 2.83GHz 4 Core(TM)2 Quad CPU with 3.7GB of memory. All experiments run in less than one hour.

In all our experiment, also the non-probabilistic ones, we consider a time bound $T = 1$ minute and a simulation step of $h = 3.7$ milliseconds, yielding paths of length $k = 16217$. These parameters are good enough to capture the physiological behaviour of the heart.

Bradycardia correction

The aim of the first experiments that we present is to check whether the pacemaker is capable to correct Bradycardia in the human heart. Thus, we set our Simulink heart model in such a way as to produce too few heart beats in a minute. The pacemaker should then correct the Bradycardia by stimulating the heart.

The results of the simulation are shown in Figure 5.14 where we depict two signals. The first one (in blue, continuous line) denotes the AP generated by the SA node. In this scenario the SA node is in the Bradycardia mode. More precisely, we have three beats in six seconds, which is approximately 30 beats per minute. The second signal (in red, dotted line) denotes the AP from one of the cells situated in the ventricle. This is the signal which is captured and paced by the pacemaker. Note that the pacemaker increases the number of beats per minute by first delivering a beat to the ventricle after approximately one second.

Probabilistic switching experiments

We carry out experiments when the probabilistic switching between different heart behaviours is taken into account. Figure 5.15 depicts the results on the relationship between the probability to generate Bradycardia and the number of pacemaker beats to the ventricle. We range the probability from 0.05 to 0.95 and run 40 experiments, each representing 8 minutes of the heart beat. As expected, by increasing the probability the pacemaker delivers more beats to the ventricle.

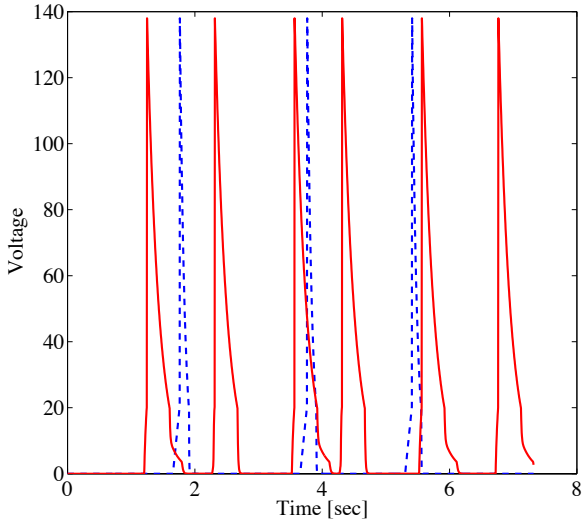


Figure 5.14: Bradycardia correction experiment.

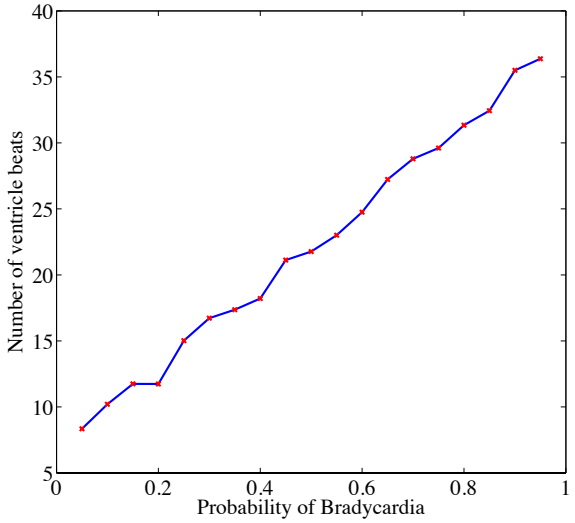


Figure 5.15: Bradycardia experiment

AV node block

In Figure 5.16 we depict the case when the ERP value of the AV node is long enough, so that it filters out the signal from the SA node. As described in Section 5.1, a cell cannot be stimulated during its ERP phase. Thus, increasing the ERP value of the AV node results in filtering some of the signal that comes from the atrium. In this case, the SA node signal (in blue, dotted line) is being blocked by a high ERP value of the AV node (signal in red, continuous line). The factor is $2:1$ (two beats in the atrium result in one beat in the ventricle). A long ERP value for the AV node induces Bradycardia in the ventricle.

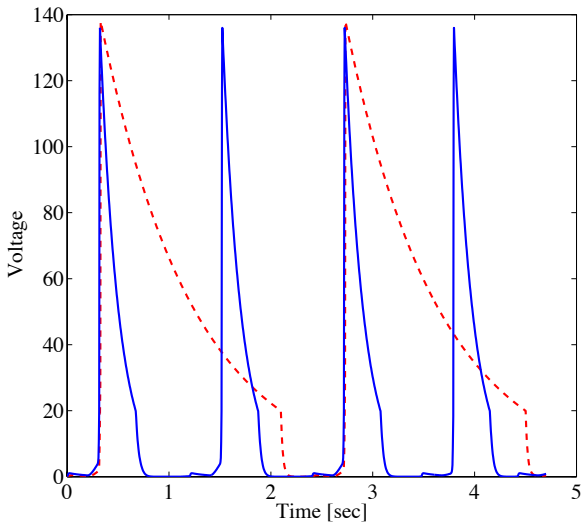


Figure 5.16: AV node block experiments

Noise

In this section we describe experiments that take into account noise on sensors. In the experiments that we run to simulate the noise we have 2 parameters: the mean μ and the variance σ^2 of the normally distributed noise. Figure 5.17 shows the results of the experiments for different values of μ (red line with $\mu = -0.3$, green line with $\mu = -0.2$ and blue line with $\mu = -0.1$). We choose μ as negative in order to simulate the undersensing effect. In each experiment with fixed mean μ, we vary the variance from 0.1 to 1 with step of 0.1. Figure 5.17 demonstrates that, when the noise with large mean (the red line with mean equal to -0.3 for example) is added to the stimulus, the number of beats in the ventricle decreases. This is due to the fact that more beats induced by the pacemaker will be lost. At the same time, increasing the variance of the normal distribution will produce more beats. The reason is that greater variance to the noise, when centred at negative mean, produces better chances of picking positive samples from the normal distribution. This, in turn, yields a better chance for the stimulus to be high enough to stimulate the cell.

Energy

In this section, we analyse energy consumption of the pacemaker. In Figure 5.18 we depict the pacemaker battery charge in one minute period. In this experiment the SA node induces Bradycardia. We varied two parameters, TAVI and TURI, which are the default programmable parameters used by technicians to ensure a heart beat between 60 and 100 BPM. The value of TAVI is 70-300 msec with 10 msec increment and the value of TURI is 50-175 BPM with 5 BPM increment. All the time values are shown in msecs.

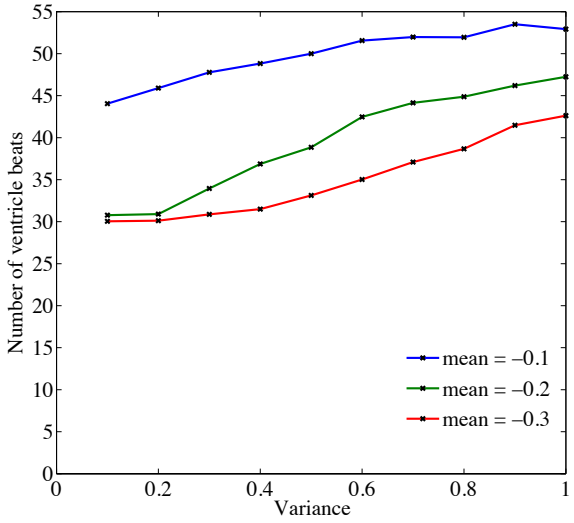


Figure 5.17: Noise experiment

In Figure 5.18 there is an energy rise when TURI< 50 or TAVI> 200. This is due to the fact that we are forcing the pacemaker to wait less between two consecutive ventricular events. Thus, the pacemaker will initiate most of the ventricular beats before a natural beat happens. The experimental results confirm our intuition that, by waiting less, the pacemaker will consume more energy, since it paces more frequently.

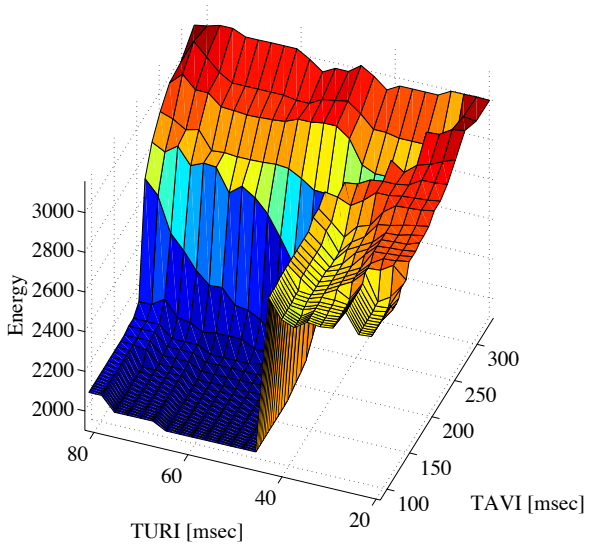


Figure 5.18: Battery charge in 1 min period

Pacemaker mediated Tachycardia

As mentioned before, in some cases the pacemaker can increase the heart rate inappro-

priately, i.e., Pacemaker Mediated Tachycardia (PMT), which is considered to be unsafe. In this section, we show that this scenario can be modelled in our framework. We then verify the advanced pacemaker model which can correct PMT (see Section 5.4).

In human hearts, the atrium can beat faster than the ventricle, at ratio 2:1 or 3:1. The resulting heart beat can still be regular due to a special cell called the AV node which has a refractory period longer than the other cells. The AV node connects the ECS of the atrium to the ECS of the ventricle. The pacemaker tries to maintain a 1:1 AV conduction through the AVI component. Thus, in the event of PMT, the pacemaker increases the beats in the ventricle inappropriately. In order to avoid this behaviour we need to switch the pacemaker from the DDD mode to the VDI mode after the PMT is detected. After a normal heart beat is re-established, the pacemaker can switch back to the DDD mode.

To accomplish this result we generate Tachycardia and assign a longer ERP value to the AV node. The pacemaker algorithm detects the PMT behaviour in the atrium. Then, after confirmed detection, the pacemaker switches from the DDD mode to VDI mode. During the VDI mode, the AV synchrony function of the pacemaker is deactivated, and thus the ventricular rate is decoupled from the fast atrial rate. When the components detect the end of the PMT, the pacemaker switches back to the DDD mode.

For our experiments, using the notation introduced in Section 5.6, we picked $\varepsilon = 0.01$ and $\delta = 0.01$ yielding $N = 11505$ sample paths. The advanced pacemaker model is capable of correcting $N' = 9017$ of them. Intuitively, this means that with confidence 99% we are sure that the computed probability $\frac{N'}{N} = 0.783$ is within 0.01 radius from the real probability p. In Figure 5.19 we show an example containing two graphs. The first graph depicts Tachycardia in the ventricle due to PMT. The second graph shows how the pacemaker switches its mode from DDD to VDI at time 13. As a result, the number of ventricle beats decreases.

5.7 Synthesising parameters for pacemakers using TIOAs

In this section we solve the *optimal parameter synthesis problem*, i.e., the problem of finding optimal model parameters' values. We will define this problem with respect to an objective function. We do not restrict to a single type of objective function, and instead admit a family of them, each of which will correspond to ensuring a particular quantitative property. The content of this section is based on one report awaiting submission [DKM13].

The general idea is to model the human heart and the pacemaker as a network of *Timed I/O Automata* (TIOAs) which are parametrised. Some parameters are under control of the user: we call those parameters *controllable* parameters. Think, for example, of real pacemaker software that technicians can set during a pacemaker implantation. The software allows the technicians to manually set some of the timing behaviours of the pacemakers. Some other parameters of our model are *uncontrollable*, meaning that we

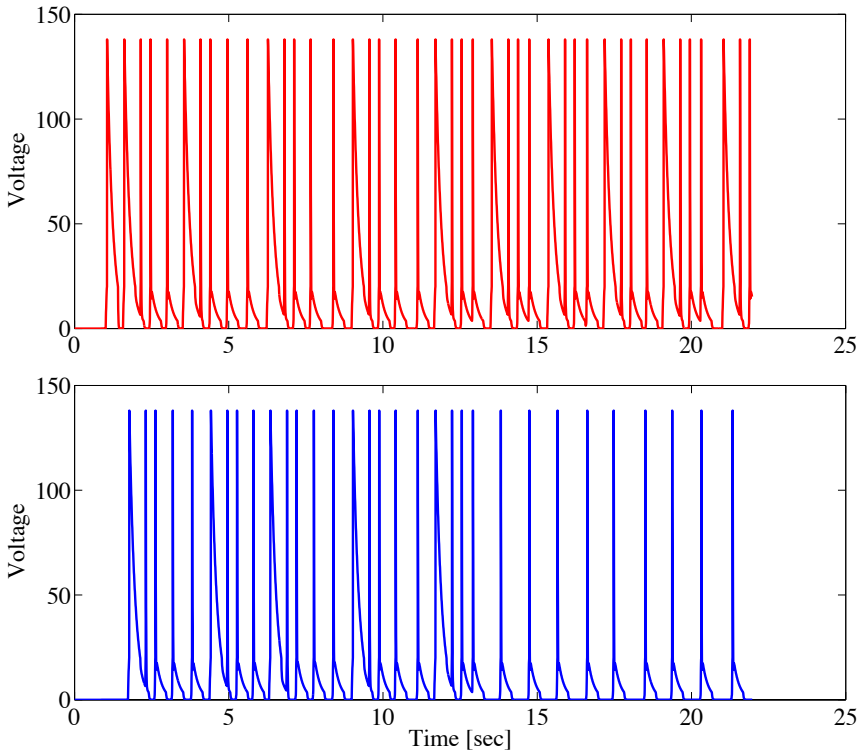


Figure 5.19: PMT correction.

cannot adjust them to our need. Think, for example, about the timing at which ventricle beats are fired. It is clear that we cannot control such timings. Our problem is to find the best values for the controllable parameters.

Due to complexity reasons, in this section, we restrict ourselves to network of TIOAs and not to the more general networks of HIOAs. The reason for that is that the algorithms presented do not scale enough to consider models which are more complex than TIOAs. Network of HIOAs will be considered for future work.

As stated at the beginning of the chapter the problem can be summarised as follows.

Optimal parameter synthesis problem

Input:	A parametric network of *Timed I/O Automata* TIOAs $\mathcal{N}$, a set of parameters $\Gamma = \Gamma_u \cup \Gamma_c$ composed of controllable (Γ_c) and uncontrollable (Γ_u) parameters, a *Counting Metric Temporal Logic* (CMTL) formula φ and a path length n.
Problem:	Find the optimal parameter values for Γ_c for any values of parameters Γ_u with respect to an objective function $\mathcal{O}$ such that φ is satisfied on $\mathcal{N}$, if such values exist.

Note here that one can find optimal values for the set of controllable parameters by discretising the set of model parameters. Theorem 5.7.8 states that it is enough to pick a discretisation step of 1 in order to generate all possible timed paths σ. Here timed paths are finite, even though the theory would work for infinite timed paths as well. The reason for finite timed paths is that, also in this case, we focus on time-bounded verification. In fact, medical devices have finite life-time due to their battery. For each timed path σ we can then check the satisfaction of the CMTL formula φ. This would accomplish the result but at a cost of generating timed paths σ for all the parameter valuations, which is expensive. In order to overcome the high complexity, we instead generate sets of linear constraints $\mathcal{S}$. A set of linear constraints $\mathcal{S}$ is a system of linear inequalities that can be expressed in matrix form as $\mathbf{A} \cdot \mathbf{x} \leqslant \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$ with $m, n \in \mathbb{N}$. The key idea is that multiple model parameters will share the same set of linear constraints $\mathcal{S}$. Thus, instead of generating a timed path σ for a parameter valuation ϑ, we check whether $\vartheta \in \mathcal{S}$. If this is the case, we then skip this valuation and therefore save computation time. Here, we use the notation $\vartheta \in \mathcal{S}$ to say that the parameter valuation ϑ, once plugged into the parameters of $\mathcal{S}$, makes the set of linear inequalities $\mathcal{S}$ true. We say that $\vartheta \notin \mathcal{S}$ otherwise.

In a nutshell, the above synthesis problem can be solved by first generating a set of linear inequalities $\mathcal{S}$ from property φ and path σ of $\mathcal{N}$, described in Section 5.7.1, and then finding an optimal solution for $\mathcal{S}$ with respect to the given objective function $\mathcal{O}$, described in Section 5.7.2.

Remark 5.7.1 *As anticipated, the optimal parameter synthesis problem will be solved with similar techniques to those employed to solve the model checking problem of CTMCs against real-time properties in Chapter 4. In fact, after generating the set of linear inequalities $\mathcal{S}$ from property φ and path σ of $\mathcal{N}$, we need to solve a volume integral over the set $\mathcal{S}$ in order to find the optimal solution for $\mathcal{S}$ with respect to the objective function $\mathcal{O}$. Again, volume integrals and complex integrations constitute the main building block of the solution to our problem.*

5.7.1 Constraint generation

We first describe the intuition for how to compute the set $\mathcal{S}$ that guarantees the satisfaction of the property along the path, and next we present an algorithm, Algorithm 13, which generates $\mathcal{S}$.

The set $\mathcal{S}$ is computed with the following simple steps:

1. Discretise according to Definition 5.2.1 the domains of the model parameters in order to generate a discrete path.

2. For each point in the discretised domain do:

   If the point does not belong to the set of constraints $\mathcal{S}$

   Generate the path σ (Definition 5.2.3).

   Generate the set of inequalities which satisfy φ in σ (Algorithm 14, 18, 19).

Algorithm 13 generates the constraints $\mathcal{S}$ with the help of three subroutines, Algorithm 14, Algorithm 18 and Algorithm 19. We will now describe Algorithm 13 and its subroutines step by step.

Algorithm 13 Constraint generation for $\mathcal{N}$ with m-components, CMTL formula φ and path length n

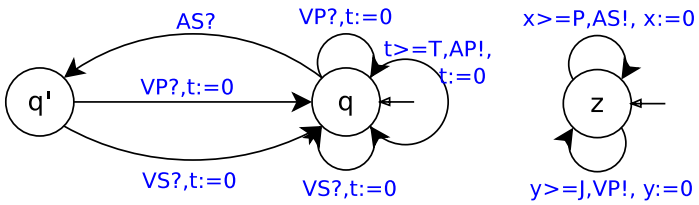
Ensure: Family of linear inequalities $\mathcal{S}$ over the parameters Γ

1: **Function** $\mathsf{Sat}(\mathcal{N}, \varphi, n)$
2: $\bar{\Gamma} := \mathsf{Discretise}(\Gamma, \delta)$
3: **for** $\vartheta \in \bar{\Gamma}$ **do**
4: **if** $\vartheta \notin \mathcal{S}$ **then**
5: $\sigma := \mathsf{Gen\_path}(\mathcal{N}, n, \vartheta)$
6: $(\mathcal{S}', \mathcal{T}) := \mathsf{Path\_Constr\_Gen}(\mathcal{N}, \sigma)$
7: $\mathcal{S}'' := \mathsf{Constr\_Gen}(\sigma, 0, \varphi, \mathcal{T})$
8: $\mathcal{S} := \mathcal{S} \bigvee (\mathcal{S}' \bigwedge \mathcal{S}'')$
9: **end if**
10: **end for**
11: **return** $\mathcal{S}$

The first step of Algorithm 13 discretises the domains $\Gamma^{(i)}$ of each parameter with a discretisation step $\delta \in \mathbb{R}_{\geqslant 0}$ (line 2), obtaining the set of discretised parameters $\bar{\Gamma}$. The algorithm then iterates over each point of $\bar{\Gamma}$ and at every iteration checks whether the discretised set of points under consideration, say ϑ, satisfies the set of linear inequalities $\mathcal{S}$ or not. This operation is indicated in Algorithm 13 with $\vartheta \notin \mathcal{S}$. The second step of the algorithm generates a discrete path σ. We do not describe the function $\mathsf{Gen\_path}(\mathcal{N}, n, \vartheta)$ since it is equivalent to Definition 5.2.3. The function $\mathsf{Gen\_path}$ returns a discrete path

where each transition is labelled with an output action. Afterwards, Algorithm 13 generates the set of linear inequalities over the parameter set Γ from the the discrete path σ of length n. This is accomplished with Algorithm 14, which is composed of three main cycles (reproduced respectively in Algorithm 15, Algorithm 16 and Algorithm 17). The algorithm returns two sets of constraints, $\mathcal{T}$ and $\mathcal{S}'$. The set $\mathcal{T}$ contains the time constraints $t_j^{(i)}$ over the parameter set Γ, $j \in \{0, \ldots, |\sigma| - 1\}$, $i \in \{1, \ldots, m\}$, corresponding to every discrete transition j of σ and component i of $\mathcal{N}$. The set of constraints $\mathcal{S}'$ contains the relationship between the clock valuations $\eta_j^{(i)}$ and guards $g^{(i)}$ for all components i and transition j of $\mathcal{N}$. For instance, if there is a transition labelled with an enabled output action and guard $x \leqslant \gamma$, where $x \in \mathcal{X}^{(i)}$ and $\gamma \in \Gamma^{(i)}$ for some component i, then $\mathcal{S}'$ will contain the constraint $\eta_j^{(i)}(x) \leqslant \gamma$. The first cycle of Algorithm 14 at line 5, presented in Algorithm 15, iterates over the set of components $\mathcal{I}_j$ that have an enabled output transition with maximal priority and generates the symbolic time constraints $t_j^{(i)}$. It also generates the set of constraints $\mathcal{S}'$ corresponding to the discrete transition with $\{<, \leqslant\} \subseteq g^{(i)}.x(1)$. At the end of the cycle (line 13 of Algorithm 15), the algorithm creates a new clock valuation $\eta_{j+1}^{(i)}$ from the symbolic time constraint $t_j^{(i)}$. Each clock valuation is a symbolic expression over $t_j^{(i)}$ with $i \in \mathcal{I}_j$. The second cycle of Algorithm 14 at line 7, presented in Algorithm 16, iterates over the set of components $\mathcal{I}_j^s$ that synchronise with $\mathcal{I}_j$. Every transition in σ, corresponding to a component of $\mathcal{I}_j^s$, is labelled with an input action. This cycle generates the set of constraints $t_j^{(k)}$ and $\mathcal{S}'$ for $k \in \mathcal{I}_j^s$. The last cycle of Algorithm 14 at line 9, presented in Algorithm 17, generates the set of time constraints $t_j^{(k)}$ for the remaining components $\mathcal{I}_j^c := \{1, \ldots, m\} \backslash (\mathcal{I}_j \cup \mathcal{I}_j^s)$. Note that $t_j^{(k')} > t_j^{(i)}$ for any $k' \in \mathcal{I}_j^c$ and $i \in \mathcal{I}_j$. Line 10 of Algorithm 14 adds to $\mathcal{S}'$ the relationships between all time constrains for each component $i \in \mathcal{N}$, namely, $t_j^{(i)} = t_j^{(k)}$ for every component that synchronises, and $t_j^{(k')} > t_j^{(i)}$ for every other component that does not synchronise, where $i, j \in \mathcal{I}_j \cup \mathcal{I}_j^s$ and $k' \in \mathcal{I}_j^c$.

Example 5.7.1 *We show now the functioning of Algorithm 14 with an example. In Figure 5.20, previously shown in Example 5.2.1, we present an example of a network $\mathcal{N}$ composed of two TIOAs, $\mathcal{A}_1$ and $\mathcal{A}_2$.*



(a) $\mathcal{A}_1$ (b) $\mathcal{A}_2$

Figure 5.20: Example network $\mathcal{N}$ with two components.

Here $\mathcal{X}^{(1)} = \{t\}$, $\mathcal{X}^{(2)} = \{x, y\}$, $\Gamma^{(1)} = \{T\}$, $\Gamma^{(2)} = \{P, J\}$, $\Sigma_{\text{in}}^{(1)} = \{\mathsf{AS}, \mathsf{VS}, \mathsf{VP}\}$,

Algorithm 14 Constraints generation for the path σ

Ensure: Family of linear inequalities $\mathcal{S}'$ and sequence of time constraints $\mathcal{T}$ over Γ

Function Path_Constr_Gen($\mathcal{N}, \sigma$)
$t_j^{(i)} := 0$, $\eta_j^{(i)} := \mathbf{0}$, for all $i \in \{1, \ldots, m\}$ and $j \in \{0, \ldots, |\sigma| - 1\}$
for $j := 0$ to $|\sigma| - 1$ **do**
  $\mathcal{I}_j$ - index of components that have an enabled output action with maximal priority
  See Algorithm 15.
  $\mathcal{I}_j^s$ - index of components that synchronise with an output transition from $\mathcal{I}_j$
  See Algorithm 16.
  $\mathcal{I}_j^c := \{1, \ldots, m\} \backslash (\mathcal{I}_j \cup \mathcal{I}_j^s)$
  See Algorithm 17.
  $\mathcal{S}' := \mathcal{S}' \wedge \left\{ \bigwedge_{i,k \in \mathcal{I}_j \cup \mathcal{I}_j^s, i \neq k} (t_j^{(i)} = t_j^{(k)}) \bigwedge_{i \in \mathcal{I}_j, k \in \mathcal{I}_j^c} (t_j^{(i)} < t_j^{(k)}) \right\}$
end for
return $(\mathcal{S}', \mathcal{T})$

Algorithm 15 Constraints generation for the path σ (First **for** cycle)

for $i \in \mathcal{I}_j$ **do**
  maxguard := 0
  $e^{(i)} := (q_j^{(i)}, a, g^{(i)}, X^{(i)}, q_{j+1}^{(i)})$
  for $x \in g^{(i)}$ **do**
    if $g^{(i)}.x(1) =$ " $\geqslant$ " **or** $g^{(i)}.x(1) =$ " $>$ " **then**
      maxguard := $\max\{\text{maxguard}, g^{(i)}.x(2) - \eta_j^{(i)}(x)\}$
    else if $g^{(i)}.x(1) =$ " $\leqslant$ " **then**
      $\mathcal{S}' := \mathcal{S}' \wedge \{\eta_j^{(i)}(x) \leqslant g^{(i)}.x(2)\}$
    else
      $\mathcal{S}' := \mathcal{S}' \wedge \{\eta_j^{(i)}(x) < g^{(i)}.x(2)\}$
    end if
  end for
  $t_j^{(i)} := \text{maxguard}, \eta_{j+1}^{(i)} := (\eta_j^{(i)} + t_j^{(i)})[X^{(i)} := 0]$, $\mathcal{T} := \mathcal{T} \times t_j^{(i)}$
end for

$\Sigma_{\text{out}}^{(2)} = \{\mathsf{AP}\}$, $\Sigma_{\text{in}}^{(2)} = \varnothing$ *and* $\Sigma_{\text{out}}^{(2)} = \{\mathsf{AS}, \mathsf{VP}\}$. *A sample path of the network* $\mathcal{N}$ *is* $\sigma = (q, z) \to (q', z) \to (q, z) \to (q, z)$. *The initial state is* (q, z). *As usual, we omit actions and time stamps from the transitions to ease notation. The automaton* $\mathcal{A}_2$ *triggers the first two transitions with the output actions* AS *and* VP, *moving the system respectively to* (q', z) *with the first action and to* (q, z) *with the second. On these transitions, the automaton* $\mathcal{A}_1$ *will synchronise with* $\mathcal{A}_2$ *via matching inputs,* AS *and* VP. *The third transition of the*

Algorithm 16 Constraints generation for the path σ (Second **for** cycle)

for $k \in \mathcal{I}_j^s$ **do**
  $e^{(k)} := (q_j^{(k)}, a, g^{(k)}, X^{(k)}, q_{j+1}^{(k)})$
  maxguard $:= 0$
  for $x \in g^{(k)}$ **do**
    if $g^{(k)}.x(1) =$ " $\geqslant$ " **or** $g^{(k)}.x(1) =$ " $>$ " **then**
      maxguard $:= \max\{\text{maxguard}, g^{(k)}.x(2) - \eta_j^{(k)}(x)\}$
    else if $g^{(k)}.x(1) =$ " $\leqslant$ " **then**
      $\mathcal{S}' := \mathcal{S}' \wedge \{\eta_j^{(k)}(x) \leqslant g^{(k)}.x(2)\}$
    else
      $\mathcal{S}' := \mathcal{S}' \wedge \{\eta_j^{(k)}(x) < g^{(k)}.x(2)\}$
    end if
  end for
  $t_j^{(k)} := \text{maxguard}, \eta_{j+1}^{(k)} := (\eta_j^{(k)} + t_j^{(k)})[X^{(k)} := 0]$
end for

Algorithm 17 Constraints generation for the path σ (Third **for** cycle)

for $k \in \mathcal{I}_j^c$ **do**
  maxguard $:= 0$
  for every outgoing transition $e^{(k)}$ out of $q_j^{(k)}$ **do**
    $e^{(k)} := (q_j^{(k)}, a, g^{(k)}, X^{(k)}, q_{j+1}^{(k)})$
    for $x \in g^{(k)}$ **do**
      if $g^{(k)}.x(1) =$ " $\geqslant$ " **or** $g^{(k)}.x(1) =$ " $>$ " **then**
        maxguard $:= \max\{\text{maxguard}, g^{(k)}.x(2) - \eta_j^{(k)}(x)\}$
      end if
    end for
  end for
  $t_j^{(k)} :=$ maxguard, $\eta_{j+1}^{(k)} := \eta_j^{(k)} + t_j^{(i)}$, for some $i \in \mathcal{I}_j$
end for

path is instead triggered by $\mathcal{A}_1$ through the output action AP.

The set of constraints generated by Algorithm 14 for each transition is given below in Table 5.1. Note that, in this example, the set $\mathcal{S}'$ will contain only the constraint $T < \max\{J - P, 0\}$, which is due to the fact that the output transition in $\mathcal{A}_1$ labelled with T will be triggered, whereas $\mathcal{A}_2$ has no enabled output transition.

Algorithm 18 generates the set of constraints for a basic counting formula BCF $\mathbb{B}$ (see Definition 5.5.6). The algorithm creates two sets, L and U, for the lower and upper bounds, respectively, appearing in $\mathbb{B}$. The sequence $\bar{w}$ contains the ordered set of elements from $L \cup U$ and the function f maps an element of $L \cup U$, to an element of the sequence $\bar{w}$. The

Step	0	1	2
$\mathcal{I}_j$	$\{2\}$	$\{2\}$	$\{1\}$
$\mathcal{I}_j^s$	$\{1\}$	$\{1\}$	$\varnothing$
$\mathcal{I}_j^c$	$\varnothing$	$\varnothing$	$\{2\}$
$\mathcal{A}_1$	$t_0^{(1)} = P$ $\eta_0^{(1)}(t) = 0$	$t_1^{(1)} = J - P$ $\eta_1^{(1)}(t) = P$	$t_2^{(1)} = T$ $\eta_2^{(1)}(t) = 0$
$\mathcal{A}_2$	$t_0^{(2)} = P$ $\eta_0^{(2)}(x) = 0$ $\eta_0^{(2)}(y) = 0$	$t_1^{(2)} = J - P$ $\eta_1^{(2)}(x) = P$ $\eta_1^{(2)}(y) = P$	$t_2^{(2)} = \max\{J - P, 0\}$ $\eta_2^{(2)}(x) = J$ $\eta_2^{(2)}(y) = J$
$\mathcal{S}'$	$\varnothing$	$\varnothing$	$T < \max\{J - P, 0\}$

Table 5.1: Example Algorithm 14

Algorithm 18 Constraints generation for basic counting formulas (BCFs)

Ensure: Family of linear inequalities $\mathcal{S}''$ over $t_0, \ldots, t_{n-i}$

1: **Function** Sum_Gen(σ, i, φ)

2: $\bar{\sigma} := \sigma[\![i]\!]$, $L := \{l_j \mid j \in J\}$, $U := \{u_j \mid j \in J\}$ and $\bar{w} := \mathrm{sort}(L \cup U)$

3: f maps an element of $L \cup U$ to an element of $\bar{w}$

4: $\mathcal{S}'' := \bigvee_{\substack{y_k \in \{0,\ldots,|\bar{\sigma}|-1\} \\ y_1 \leqslant \cdots \leqslant y_{|\bar{w}|}}} \left(\bigwedge_{z \in \{0,\ldots,|\bar{w}|\}} \bar{\sigma}@\bar{w}(z) = y_z \right) \wedge \left(\sum_{j \in J} c_j \sum_{\iota = y_{f(\ell_j)}}^{y_{f(u_j)}-1} a_j \in \bar{\sigma}[\iota] \bowtie b \right)$

5: Where we define $(\bar{\sigma}@\bar{w}(z) = y_z) := \left(\sum_{\iota=0}^{y_z} t_\iota > \bar{w}(z) \wedge \sum_{\iota=0}^{y_z - 1} t_\iota < \bar{w}(z) \right)$

6: **return** $\mathcal{S}''$

main phase of the algorithm involves generating *all possible orderings* of the transitions occurring in $\bar{\sigma}$, where $\bar{\sigma}$ is the untimed suffix of length i of σ, with respect to the elements of $\bar{w}$. This is achieved with the outer disjunction over the set $\{0, \ldots, |\bar{\sigma}| - 1\}$. For every possible ordering, the algorithm checks whether the formula $\sum_{j \in J} c_j \sum_{\iota = y_{f(\ell_j)}}^{y_{f(u_j)}-1} a_j \in \bar{\sigma}[\iota] \bowtie b$ holds.

Example 5.7.2 *In this example we show the function of Algorithm 18 for the path* $\sigma = (q, z) \to (q', z) \to (q, z) \to (q, z)$ *and formula* $\varphi = \#_5^7 \mathsf{VP} \geqslant 1$. *The first column of Table 5.2 shows all possible ordering of variables* y_1 *and* y_2. *Note that, for a path of length* 3, $y_i \in \{0, 1, 2\}$, $i \in \{1, 2\}$. *The second column of Table 5.2 shows how the time constraints are generated, while the third column shows the formula that checks whether there is at least one* VP *action present in the interval of time* 5 *to* 7.

Algorithm 19 generates the set of constraints for a CMTL formula. The algorithm proceeds by induction over the structure of the formula and generates the set of linear inequalities

Ordering	$\bigwedge_{z\in\{0,\dots,\lvert\bar{w}\rvert\}} \bar{\sigma}@\bar{w}(z) = y_z$	$\sum_{j\in J} c_j \sum_{\iota=y_{f(\ell_j)}}^{y_{f(u_j)}-1} a_j \in \bar{\sigma}[\iota] \bowtie b$
$(y_1 = 0 \wedge y_2 = 0)$	$(t_0 > 5)$	**false**
$(y_1 = 0 \wedge y_2 = 1)$	$(t_0 > 5) \wedge$ $(t_0 + t_1 > 7 \wedge t_0 < 7)$	**false**
$(y_1 = 0 \wedge y_2 = 2)$	$(t_0 > 5) \wedge$ $(t_0 + t_1 + t_2 > 7 \wedge t_0 + t_1 < 7)$	**true**
$(y_1 = 1 \wedge y_2 = 1)$	$(t_0 + t_1 > 5 \wedge t_0 < 5) \wedge$ $(t_0 + t_1 > 7 \wedge t_0 < 7)$	**false**
$(y_1 = 1 \wedge y_2 = 2)$	$(t_0 + t_1 > 5 \wedge t_0 < 5) \wedge$ $(t_0 + t_1 + t_2 > 7 \wedge t_0 + t_1 < 7)$	**true**
$(y_1 = 2 \wedge y_2 = 2)$	$(t_0 + t_1 + t_2 > 5 \wedge t_0 + t_1 < 5) \wedge$ $(t_0 + t_1 + t_2 > 7 \wedge t_0 + t_1 < 7)$	**false**

Table 5.2: Constraint generation for BCF

$\mathcal{S}''$ over Γ. We use the function Rewrite to rewrite each t_j in terms of parameters in Γ.

Finally, we state two theorems which establish the correctness of Algorithm 13. Theorem 5.7.4 deals with the correctness of the generated set $\mathcal{S}$ of constraints, whereas Theorem 5.7.8 shows that any CMTL formula is preserved even if we discretise the domain of the model parameters.

From here on we consider infinite timed paths ρ instead of finite timed paths σ. The reason is that the theory applies to both.

In the rest of this section we will assume a timed path ρ in which $t_0, t_1, \dots$ are the times at which transitions occur. In order to prove Theorem 5.7.4 we need to introduce some lemmas that prove intermediate results for simple CMTL formulas, namely, basic counting formulas. The first lemma presented is Lemma 5.7.1. Lemma 5.7.1 states that ρ satisfies a basic CMTL formula $\mathbb{B} \bowtie b$ at step i if and only if the i-th suffix of the untimed version of ρ, $\rho[\![i]\!]$, satisfies the formula $\mathbb{B} \bowtie b$ when the time instants $t_i, \dots, t_{n-1}$ are plugged back into ρ. The lemma is intuitive since (ρ, i) and $\rho[\![i]\!](t_i, \dots, t_{n-1}, 0)$ are essentially the same timed path. Note that (ρ, i), according to the semantics of CMTL (see Section 5.5), satisfies a CMTL formula φ if the suffix of ρ starting at step i satisfies φ. On the other hand, $\rho[\![i]\!](t_i, \dots, t_{n-1}, 0)$ is the untimed suffix of the timed path ρ starting at step i, $\rho[\![i]\!]$, which is subsequently instantiated with the original time stamps $(t_i, \dots, t_{n-1})$, making the definition of (ρ, i) and $\rho[\![i]\!](t_i, \dots, t_{n-1}, 0)$ the same.

Lemma 5.7.1 *Given a network of TIOAs $\mathcal{N}$, a timed path ρ, $i \in \mathbb{N}$, $t_i, \dots, t_{n-1}$ time*

Algorithm 19 Constraints generation for CMTL formulas

Require: A finite timed path σ of length $n > 0$, an index i, a CMTL formula φ and a set of time constraints $\mathcal{T}$

Ensure: Family of linear inequalities $\mathcal{S}''$ over Γ

Function $\mathsf{Constr\_Gen}(\sigma, i, \varphi, \mathcal{T})$

case(φ) :

$$\varphi = \sum_{j\in J} c_j \#_{\ell_j}^{u_j} a_j \bowtie b \quad : \quad \mathcal{S}'' := \mathsf{Sum\_Gen}(\sigma, i, \varphi)$$

$$\varphi = \neg\varphi_1 \quad : \quad \mathcal{S}'' := \neg\mathsf{Constr\_Gen}(\sigma, i, \varphi_1, \mathcal{T})$$

$$\varphi = \varphi_1 \wedge \varphi_2 \quad : \quad \mathcal{S}'' := \mathsf{Constr\_Gen}(\sigma, i, \varphi_1, \mathcal{T}) \wedge \mathsf{Constr\_Gen}(\sigma, i, \varphi_2, \mathcal{T})$$

$$\varphi = \varphi_1 \mathcal{U}^{[\ell,u]} \varphi_2 \quad : \quad \mathcal{S}'' := \Big(\bigvee_{i'=i}^{n} \mathsf{Constr\_Gen}(\sigma, i', \varphi_2, \mathcal{T}) \wedge \ell \leq \sum_{k=i}^{i'} t_k \leq u \wedge \Big(\bigwedge_{i''=i}^{i'-1} \mathsf{Constr\_Gen}(\sigma, i'', \varphi_1, \mathcal{T})\Big)\Big)$$

if $\mathcal{T} \neq \varnothing$ **then**

return $\mathsf{Rewrite}(\mathcal{S}'', \mathcal{T})$

else

return $\mathcal{S}''$ (where $\mathcal{S}''$ is the set of constraints over $t_0, \ldots, t_n$ used in Theorem 5.7.4)

end if

instants, $b \in \mathbb{Z}$ and a basic counting formula $\mathbb{B}$ we have the following.

$$(\rho, i) \models^{\mathcal{N}} \mathbb{B} \bowtie b \quad \text{iff}$$
$$(\rho[\![i]\!](t_i, \ldots, t_{n-1}), 0) \models^{\mathcal{N}} \mathbb{B} \bowtie b.$$

Proof The proof follows simply by the definition of the semantics of CMTL. ■

Next we present Lemma 5.7.2 and Lemma 5.7.3. The two lemmas prove a timed path ρ satisfies $\mathbb{B} \bowtie b$ at step i if and only if the time sequence $t_0, t_1, \ldots$ satisfies the set of linear constraints returned by $\mathsf{Sum\_Gen}(\rho[\![0]\!], i, \mathbb{B} \bowtie b)$.

Lemma 5.7.2 *Given a network of TIOAs $\mathcal{N}$, a timed path ρ, $i \in \mathbb{N}$, $t_i, \ldots, t_{n-1}$ time instants, $b \in \mathbb{Z}$ and a basic counting formula $\mathbb{B}$ we have the following.*

$$(\rho, i) \models^{\mathcal{N}} \mathbb{B} \bowtie b \quad \Longrightarrow$$
$$(t_0, \ldots, t_{n-1}) \in \mathsf{Sum\_Gen}(\rho[\![0]\!], i, \mathbb{B} \bowtie b)$$

Proof By Lemma 5.7.1 this is equivalent to proving that

$$(\rho[\![i]\!](t_i, \ldots, t_{n-1}), 0) \models^{\mathcal{N}} \mathbb{B} \bowtie b \quad \Longrightarrow \tag{5.4}$$
$$(t_i, \ldots, t_{n-1}) \in \mathsf{Sum\_Gen}(\rho[\![i]\!], 0, \mathbb{B} \bowtie b)$$

Algorithm 13 will generate multiple sets of inequalities over the variables $\lambda_0, \ldots, \lambda_{n-i}$. Let us instantiate the variables with the values of $(t_i, \ldots, t_{n-1})$ taken from $\rho[\![i]\!](t_i, \ldots, t_{n-1}) = \vec{q_j} \xrightarrow{t_j} \vec{q}_{j+1} \xrightarrow{t_{j+1}} \cdots \xrightarrow{t_{n-1}} \vec{q_n}$. Due to the fact that ρ is a concrete timed path of the system, it must be true that the values of $\lambda_0, \ldots, \lambda_{n-i}$ instantiated with $(t_i, \ldots, t_{n-1})$ satisfy the constraints in $\bigwedge_{z \in \{0,\ldots,|\bar{w}|\}} \left(\sum_{\iota=0}^{y_z} \lambda_\iota > \bar{w}(z) \wedge \sum_{\iota=0}^{y_z-1} \lambda_\iota < \bar{w}(z) \right)$ for a given vector $\bar{w}$. The vector $\bar{w}$ is the one obtained by considering the original timed path $\rho@x$ where $x \in \{L \cup U\}$ with $L := \{l_j \mid j \in J\}$, $U := \{u_j \mid j \in J\}$, where each l_j, u_j is taken from the CMTL formula $\mathbb{B}$.

Now consider the formula $\left(\sum_{j \in J} c_j \sum_{\iota=y_{f(\ell_j)}}^{y_{f(u_j)}-1} a_j \in \rho[\![i]\!](t_i, \cdots, t_{n-1})[\iota] \bowtie b \right)$ generated by Algorithm 13 and substitute $y_{f(\ell_j)}$ and $y_{f(\ell_j)}$ with $\rho@w(z) = y_z$ for the z that gives you $y_{f(\ell_j)}$ or $y_{f(\ell_j)}$. The formula thus generated is equivalent to $\sum_{j \in J} c_j \sum_{k=(\rho[\![i]\!](t_i,\cdots,t_{n-1})@\ell_j)}^{(\rho[\![i]\!](t_i,\cdots,t_{n-1})@u_j)-1} (a_j \in \rho[k]) \bowtie b$ which is true if and only if the Equation (5.4) is satisfied. ■

Lemma 5.7.3 *Given a network of TIOAs $\mathcal{N}$, a timed path ρ, $i \in \mathbb{N}$, $t_i, \ldots, t_{n-1}$ time instants, $b \in \mathbb{Z}$ and a basic counting formula $\mathbb{B}$ we have the following.*

$$(t_0, \ldots, t_{n-1}) \in \mathsf{Sum\_Gen}(\rho[\![0]\!], i, \mathbb{B} \bowtie b) \implies$$
$$(\rho, i) \models^{\mathcal{N}} \mathbb{B} \bowtie b$$

Proof By Lemma 5.7.1 this is equivalent to proving that

$$(t_i, \ldots, t_{n-1}) \in \mathsf{Sum\_Gen}(\rho[\![i]\!], 0, \mathbb{B} \bowtie b) \implies \tag{5.5}$$
$$(\rho[\![i]\!](t_i, \ldots, t_{n-1}), 0) \models^{\mathcal{N}} \mathbb{B} \bowtie b$$

Here the reverse reasoning of Lemma 5.7.2 is applied. If there is a solution to the system of constraints returned from Algorithm 13, it means that it is possible to find a sequence of $\lambda_0, \ldots, \lambda_{n-i}$ that satisfies the constraint $\bigwedge_{z \in \{0,\ldots,|\bar{w}|\}} \left(\sum_{\iota=0}^{y_z} \lambda_\iota > \bar{w}(z) \wedge \sum_{\iota=0}^{y_z-1} \lambda_\iota < \bar{w}(z) \right)$ for a given vector $\bar{w}$. The vector $\bar{w}$ defines a sequence of states to visit (under the time constraints $\lambda_0, \ldots, \lambda_{n-i}$) such that it is possible to satisfy the linear inequality $\left(\sum_{j \in J} c_j \sum_{\iota=y_{f(\ell_j)}}^{y_{f(u_j)}-1} a_j \in \rho[\![i]\!](\lambda_0, \ldots, \lambda_{n-i})[\iota] \bowtie b \right)$. The timed path $\rho[\![i]\!](\lambda_0, \ldots, \lambda_{n-i})$ must then satisfy $\mathbb{B} \bowtie b$, which concludes the proof. ■

We are now ready to prove Theorem 5.7.4, which is a generalisation of Lemma 5.7.2 and Lemma 5.7.3 to the whole syntax of CMTL, rather than only considering basic CMTL formulas.

Theorem 5.7.4 *Let $\rho(t_0, \ldots, t_{n-1})$ be the instantiated timed path of the network $\mathcal{N}$ of TIOAs and $i \in \mathbb{N}$ an index ($i \leqslant n$). For every CMTL formula φ it holds*

$$(\rho(t_0, \ldots, t_{n-1}), i) \models^{\mathcal{N}} \varphi \quad \text{iff} \quad (t_0, \ldots, t_{n-1}) \in \mathcal{S},$$

where $\mathcal{S} := \mathsf{Constr\_Gen}(\rho, i, \varphi, \varnothing)$.

Proof Let $\rho = \vec{q_0} \xrightarrow{t_0} \vec{q_0} \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} \vec{q_n}$ be the finite timed path of the network of TIOAs $\mathcal{N}$, $i \in \mathbb{N}$ be an index ($i \leqslant n$) and $\mathbb{B} \bowtie b$ a CMTL formula. We now prove the main theorem. The proof proceeds by induction on the length of the formula. As usual, φ, φ_1 and φ_2 are CMTL formulas and $\ell, u \in \mathbb{R}$. We have:

$$(\rho(t_0, \ldots, t_{n-1}), i) \models^{\mathcal{N}} \varphi \quad \text{iff} \quad (t_0, \ldots, t_{n-1}) \in \mathcal{S}.$$

- $\varphi = \mathbb{B} \bowtie b$. The theorem is true by the Lemma 5.7.2 and Lemma 5.7.3.
- $\varphi = \varphi_1 \wedge \varphi_2$, $\varphi = \neg\varphi_1$. Trivial just by induction hypothesis.
- $\varphi = \varphi_1 \mathcal{U}^{[\ell,u]} \varphi_2$. The proof follows from [CDKM11, CDKM13b]. ■

The next step is to prove Theorem 5.7.8, which shows that any CMTL formula is preserved even if we discretise the domain of the model parameters. In order to do so we introduce three lemmas: Lemma 5.7.5, Lemma 5.7.6 and Lemma 5.7.7.

Lemma 5.7.5 proves that, given a network of TIOAs $\mathcal{N} = \{\mathcal{A}^{(i)} \mid i \in \{1, \ldots, m\}\}$ for which all the guard constants are integers, then it must be the case that each entering clock valuation of each clock, namely the value of the clocks when entering a new state, is integer as well.

Lemma 5.7.5 *Let* $\mathcal{N} = \{\mathcal{A}^{(i)} \mid i \in \{1, \ldots, m\}\}$ *be a network of TIOAs* $\mathcal{A}^{(i)}$ *for which* $\Gamma^{(i)} = \varnothing$ *and all guard constants are integers for every* $i \in \{1, \ldots, m\}$. *Then each entering clock valuation* $\eta^{(i)}(x)$, *for each* $i \in \{1, \ldots, m\}$ *and* $x \in \mathcal{X}^{(i)}$, *is an integer.*

Proof Simply follows from the semantics of TIOAs which states that as soon as a guard becomes true the corresponding transition must be taken. Thus, if all the guards of the transitions contain integers, it must be the case that the transition is taken at an integer time point. ■

Lemma 5.7.6 states that, given a parameter instantiation $\vartheta^{(i)}$, $i \in \{1, \ldots, m\}$, which generates a timed path ρ, there exists an integer version of $\vartheta^{(i)}$, which we indicate with $\lfloor \vartheta^{(i)} \rceil$, that can reproduce the same timed path ρ. Lemma 5.7.6 is crucial for proving Theorem 5.7.8.

Lemma 5.7.6 *Let* $\vartheta^{(i)}$, $i \in \{1, \ldots, m\}$, *be a parameter instantiation and* ρ *be the associated path of length* n *for which* $\eta_j^{(i)}$ *is the entering clock valuation of component* i *at step* $j \in \{0, \ldots, n-1\}$. *Then there exists a parameter instantiation* $\lfloor \vartheta^{(i)} \rceil$ *such that* $\rho[\![0]\!] = \bar{\rho}[\![0]\!]$, *where* $\bar{\rho}$ *is the path corresponding to* $\lfloor \vartheta^{(i)} \rceil$ *and* $\lfloor \cdot \rceil$ *is the closest integer.*

Proof Let $(q_j^{(i)}, a^{(i)}, g^{(i)}, X^{(i)}, q_{j+1}^{(i)})$ be the transition of ρ in component i taken at step j. It holds that $\eta_j^{(i)} + t_j \models g^{(i)}$, where $\rho\langle j\rangle = t_j$. Therefore, we get

$$\begin{aligned}\eta_j^{(i)} + t_j \models \bigwedge_{x\in X^{(i)}} x \bowtie g^{(i)}.x(2) \quad \text{iff}\\ \bigwedge_{x\in X^{(i)}} \eta_j^{(i)} + t_j \models x \bowtie g^{(i)}.x(2) \quad \text{iff}\\ \bigwedge_{x\in X^{(i)}} \eta_j^{(i)}(x) + t_j \bowtie g^{(i)}.x(2) \quad .\end{aligned}$$

Due to the fact that every transition of a TIOA is forced we have that

$$t_j = \min_{x\in g^{(i)}} \{g^{(i)}.x(2) - \eta_j^{(i)}(x)\} \text{ if } \exists x \in g^{(i)} \wedge g^{(i)}.x(1) = \geqslant,$$
$$t_j = 0, \text{otherwise.}$$

Let $\bar{\rho}$ be the path corresponding to $\lfloor \vartheta^{(i)} \rceil$ with $\bar{\eta}_j^{(i)}$ as the entering clock valuation of $\bar{\rho}$ and $\bar{\rho}\langle j\rangle = \bar{t}_j$. Also, we have

$$\bar{t_j} = \min_{x\in g^{(i)}} \{\lfloor g^{(i)}.x(2) \rceil - \bar{\eta}_j^{(i)}(x)\} \text{ if } \exists x \in g^{(i)} \wedge g^{(i)}.x(1) = \geqslant,$$
$$\bar{t_j} = 0, \text{otherwise.}$$

First, we prove

$$(\bar{\rho}[\![0]\!] = \rho[\![0]\!]) \implies$$
$$|\bar{\eta}_j^{(i)} - \eta_j^{(i)}| \leqslant 0.5 \wedge |\bar{t}_j - t_j| \leqslant 1, \forall j \in \{0, \ldots, n-1\}$$

by induction on j (the base case trivially holds) and we distinguish the following cases.

1. If $\nexists x \in g^{(i)} \wedge g^{(i)}.x(1) = \geqslant$ then $t_j = 0$ and it must the case that $\bar{t_j} = 0$ because $\bar{\rho}[\![0]\!] = \rho[\![0]\!]$. Then we have that $|\bar{t}_j - t_j| \leqslant 1$ and given that $|\bar{\eta}_j^{(i)} - \eta_j^{(i)}| \leqslant 0.5$ we have
$$|\bar{\eta}_{j+1}^{(i)} - \eta_{j+1}^{(i)}| = |\bar{\eta}_j^{(i)} + \bar{t_j} - (\eta_j^{(i)} + t_j)| = |\bar{\eta}_j^{(i)} - \eta_j^{(i)}| \leqslant 0.5.$$

2. If $\exists x \in g^{(i)} \wedge g^{(i)}.x(1) = \geqslant$ and $t_j = 0$. From the induction hypothesis we know that $|\bar{\eta}_j^{(i)} - \eta_j^{(i)}| \leqslant 0.5$ and it must be the case that $\bar{\eta}_j^{(i)} = \lfloor \eta_j^{(i)} \rceil$. Given that $t_j = 0$ we have that
$$\begin{aligned}\bigwedge_{x\in g^{(i)} \wedge g^{(i)}.x(1)=\geqslant} \eta_j^{(i)} \geqslant g^{(i)}.x(2) \implies\\ \bigwedge_{x\in g^{(i)} \wedge g^{(i)}.x(1)=\geqslant} \lfloor \eta_j^{(i)} \rceil \geqslant \lfloor g^{(i)}.x(2) \rceil \implies\\ \bigwedge_{x\in g^{(i)} \wedge g^{(i)}.x(1)=\geqslant} \bar{\eta}_j^{(i)} \geqslant \lfloor g^{(i)}.x(2) \rceil \implies\\ \bar{t_j} = 0 \wedge |\bar{\eta}_{j+1}^{(i)} - \eta_{j+1}^{(i)}| \leqslant 0.5\end{aligned}$$

3. If $\exists x \in g^{(i)} \wedge g^{(i)}.x(1) =\geqslant$ and $t_j > 0$. It holds that

$$\bigwedge_{x \in g^{(i)}} |\bar{t}_j - t_j| = |\lfloor g^{(i)}.x(2) \rceil - \bar{\eta}_j^{(i)} - (g^{(i)}.x(2) - \eta_j^{(i)})| \implies$$
$$\bigwedge_{x \in g^{(i)}} |\bar{t}_j - t_j| = |\lfloor g^{(i)}.x(2) \rceil - g^{(i)}.x(2) + \eta_j^{(i)} - \bar{\eta}_j^{(i)}| \implies$$
$$\bigwedge_{x \in g^{(i)}} |\bar{t}_j - t_j| \leq |\lfloor g^{(i)}.x(2) \rceil - g^{(i)}.x(2)| + |\eta_j^{(i)} - \bar{\eta}_j^{(i)}| \implies$$
$$|\bar{t}_j - t_j| \leq 1.$$

Notice that $\eta_j^{(i)} + t_j = \eta_{j+1}^{(i)} = g^{(i)}.x(2)$ for some $x \in g^{(i)}$. We also have that $\bar{\eta}_j^{(i)} + \bar{t_j} = \bar{\eta}_{j+1}^{(i)} = \lfloor g^{(i)}.x'(2) \rceil$ for some $x' \in g^{(i)}$. Now we prove that $x = x'$ when $|g^{(i)}| > 1$. If the clock x is the minimum for t_j, there exists a clock $x_1 \in g^{(i)}$ such that

$$g^{(i)}.x(2) - \eta_j^{(i)}(x) \leqslant g^{(i)}.x_1(2) - \eta_j^{(i)}(x_1) \quad \text{and}$$
$$\lfloor g^{(i)}.x(2) - \eta_j^{(i)}(x) \rceil \leqslant \lfloor g^{(i)}.x_1(2) - \eta_j^{(i)}(x_1) \rceil.$$

In Table 5.3 we show the difference between $\lfloor a - b \rceil$ and $\lfloor a \rceil - \lfloor b \rceil$ according to different values of the fractional part of a and b.

Fractional part		Difference
a	**b**	$(\lfloor a - b \rceil) - (\lfloor a \rceil - \lfloor b \rceil)$
$\leqslant 0.5$	$\leqslant 0.5$	0
$\leqslant 0.5$	> 0.5	-1
> 0.5	$\leqslant 0.5$	1
> 0.5	> 0.5	0

Table 5.3: Case distinction table

We want to prove that:

$$\lfloor g^{(i)}.x(2) - \eta_j^{(i)}(x) \rceil \leqslant \lfloor g^{(i)}.x_1(2) - \eta_j^{(i)}(x_1) \rceil \implies \tag{5.6}$$
$$\lfloor g^{(i)}.x(2) \rceil - \bar{\eta}_j^{(i)}(x) \leqslant \lfloor g^{(i)}.x_1(2) \rceil - \bar{\eta}_j^{(i)}(x_1).$$

Now we will analyse how the differences $(\lfloor g^{(i)}.x(2) - \eta_j^{(i)}(x) \rceil) - (\lfloor g^{(i)}.x(2) \rceil - \bar{\eta}_j^{(i)}(x))$ and $(\lfloor g^{(i)}.x_1(2) - \eta_j^{(i)}(x_1) \rceil) - (\lfloor g^{(i)}.x_1(2) \rceil - \bar{\eta}_j^{(i)}(x_1))$ behave.

Three cases are possible according to the fractional part of $g^{(i)}.x(2)$ and $\eta_j^{(i)}(x)$.

- The fractional part of $g^{(i)}.x(2)$ and $\eta_j^{(i)}(x)$ is $\leqslant 0.5$ (same if it is > 0.5). In this case, $(\lfloor g^{(i)}.x(2) - \eta_j^{(i)}(x) \rceil) - (\lfloor g^{(i)}.x(2) \rceil - \bar{\eta}_j^{(i)}(x)) = 0$. Moreover, $\lfloor g^{(i)}.x(2) \rceil - \bar{\eta}_j^{(i)}(x) \leqslant \lfloor g^{(i)}.x_1(2) \rceil - \bar{\eta}_j^{(i)}(x_1)$ if $(\lfloor g^{(i)}.x_1(2) - \eta_j^{(i)}(x_1) \rceil) - (\lfloor g^{(i)}.x_1(2) \rceil -$

$\bar{\eta}_j^{(i)}(x_1)) = 0$ or 1. The only critical case would be when $(\lfloor g^{(i)}.x_1(2) - \eta_j^{(i)}(x_1)\rceil) - (\lfloor g^{(i)}.x_1(2)\rceil - \bar{\eta}_j^{(i)}(x_1')) = -1$. However, this case is not possible given the fact that $\lfloor g^{(i)}.x(2) - \eta_j^{(i)}(x)\rceil \leqslant \lfloor g^{(i)}.x_1(2) - \eta_j^{(i)}(x_1)\rceil$ and the condition on the fractional part of $g^{(i)}.x_1(2) \leqslant 0.5$ and the fractional part of $\bar{\eta}_j^{(i)}(x_1) > 0.5$.

- The fractional part of $g^{(i)}.x(2) \leqslant 0.5$ and $\eta_j^{(i)}(x) > 0.5$. In this case, no matter what the value of $(\lfloor g^{(i)}.x_1(2) - \eta_j^{(i)}(x_1)\rceil) - (\lfloor g^{(i)}.x_1(2)\rceil - \bar{\eta}_j^{(i)}(x_1))$ is, the relation is preserved.
- The fractional part of $g^{(i)}.x(2) > 0.5$ and $\eta_j^{(i)}(x) \leqslant 0.5$. Now we have that $(\lfloor g^{(i)}.x(2) - \eta_j^{(i)}(x)\rceil) - (\lfloor g^{(i)}.x(2)\rceil - \bar{\eta}_j^{(i)}(x)) = 1$ and we want to show that $(\lfloor g^{(i)}.x_1(2) - \eta_j^{(i)}(x_1)\rceil) - (\lfloor g^{(i)}.x_1(2)\rceil - \bar{\eta}_j^{(i)}(x_1)) = 1$ as well. Since we know that $\lfloor g^{(i)}.x(2) - \eta_j^{(i)}(x)\rceil \leqslant \lfloor g^{(i)}.x_1(2) - \eta_j^{(i)}(x_1)\rceil$ it must be the case that
  (a) the integer part of $g^{(i)}.x(2) = g^{(i)}.x_1(2)$ and the integer part of $\eta_j^{(i)}(x) = \eta_j^{(i)}(x_1)$. In this case the condition on the fractional parts must hold, as otherwise it could not be the case that $\lfloor g^{(i)}.x(2) - \eta_j^{(i)}(x)\rceil \leqslant \lfloor g^{(i)}.x_1(2) - \eta_j^{(i)}(x_1)\rceil$. Thus, $(\lfloor g^{(i)}.x_1(2) - \eta_j^{(i)}(x_1)\rceil) - (\lfloor g^{(i)}.x_1(2)\rceil - \bar{\eta}_j^{(i)}(x_1)) = 1$;
  (b) the integer part of $g^{(i)}.x(2) < g^{(i)}.x_1(2)$ (or the integer part of $\eta_j^{(i)}(x) > \eta_j^{(i)}(x_1)$). Then, it also holds that $(\lfloor g^{(i)}.x_1(2) - \eta_j^{(i)}(x_1)\rceil) - (\lfloor g^{(i)}.x_1(2)\rceil - \bar{\eta}_j^{(i)}(x_1)) = 1$.

Therefore, we have

$$|\bar{\eta}_{j+1}^{(i)} - \eta_{j+1}^{(i)}| = |\lfloor g^{(i)}.x(2)\rceil - g^{(i)}.x(2)| \leqslant 0.5.$$

Now we prove

$$\forall j \in \{0, \ldots, n-1\}.|\bar{\eta}_j^{(i)} - \eta_j^{(i)}| \leqslant 0.5 \wedge |\bar{t}_j - t_j| \leqslant 1 \implies (\bar{\rho}[\![0]\!] = \rho[\![0]\!]).$$

by induction on j.

1. $j = 0$: In this case $\bar{\eta}_0^{(i)} = \eta_0^{(i)} = \mathbf{0}$ and we have that

$$t_0 = g^{(i)}.x_1(2) \leqslant g^{(i)}.x_2(2) \leqslant \cdots \leqslant g^{(i)}.x_{|g^{(i)}|}, \tag{5.7}$$

where $x_z \in g^{(i)}$, $\forall z \in \{1, \ldots, |g^{(i)}|\}$. Let $e_0 = (q_0^{(i)}, a^{(i)}, g^{(i)}, X^{(i)}, q_1^{(i)})$ be the first transition of ρ. We show that $e_0' = (q_0^{(i)}, a^{(i)}, \bar{g}^{(i)}, X^{(i)}, q_1^{(i)})$ is also the first transition of ρ', where $\bar{g}^{(i)}.x(1) = g^{(i)}.x(1)$ and $\bar{g}^{(i)}.x(2) = \lfloor g^{(i)}.x(2)\rceil$, for all $x \in g^{(i)}$. From Equation (5.7) we get that

$$\bar{t_0} = \lfloor g^{(i)}.x_1(2)\rceil \leqslant \lfloor g^{(i)}.x_2(2)\rceil \leqslant \cdots \leqslant \lfloor g^{(i)}.x_{|g^{(i)}|}\rceil,$$

which proves the base case.

2. $j \to j+1$: It is enough to show that Equation (5.6) holds, which was proven above. ■

Lemma 5.7.7 *Let* $\{\rho[\![0]\!] \mid \rho \in \mathsf{Gen\_path}(\mathcal{N}, n, \vartheta), \vartheta \in \mathcal{V}(\Gamma)\}$ *be the set of location sequences of network* $\mathcal{N}$*. Then it holds*

$$\{\rho[\![0]\!] \mid \rho \in \mathsf{Gen\_path}(\mathcal{N}, n, \vartheta), \vartheta \in \mathcal{V}(\Gamma)\} = \{\rho[\![0]\!] \mid \rho \in \mathsf{Gen\_path}(\mathcal{N}, n, \bar{\vartheta}), \bar{\vartheta} \in \bar{\Gamma}\},$$

where $\bar{\Gamma} := \mathsf{Discretise}(\Gamma, 1)$.

Proof Let $\vartheta \in \mathcal{V}(\Gamma)$. Then, there is $\bar{\vartheta} \in \bar{\Gamma}$ such that $\bar{\vartheta} = \lfloor \vartheta \rceil$. The Lemma holds from Lemma 5.7.6. ■

Theorem 5.7.8 *Let* $\mathcal{N} = \{\mathcal{A}^{(i)} \mid i \in \{1, \ldots, m\}\}$ *be a network of TIOAs* $\mathcal{A}^{(i)}$ *and* $n \in \mathbb{N}$*. We have that*

$$\bigvee_{\vartheta \in \Gamma} \left(\mathcal{S}' \wedge \mathsf{Constr\_Gen}(\rho, 0, \varphi, \mathcal{T})\right) = \bigvee_{\bar{\vartheta} \in \bar{\Gamma}} \left(\bar{\mathcal{S}}' \wedge \mathsf{Constr\_Gen}(\rho, 0, \varphi, \bar{\mathcal{T}})\right)$$

where $(\mathcal{S}', \mathcal{T}) := \mathsf{Path\_Constr\_Gen}(\mathcal{N}, \mathsf{Gen\_path}(\mathcal{N}, n, \vartheta))$ *and* $(\bar{\mathcal{S}}', \bar{\mathcal{T}}) := \mathsf{Path\_\ Constr\_Gen}(\mathcal{N}, \mathsf{Gen\_path}(\mathcal{N}, n, \bar{\vartheta}))$ *for all* $\vartheta \in \Gamma$*, and* $\bar{\vartheta} \in \bar{\Gamma}$*. Here we say that two constraints are equal if they share the same solution set.*

Proof The proof follows from Lemma 5.7.5, Lemma 5.7.6, Lemma 5.7.7. ■

The reader should note the similarities with techniques used in Chapter 4. Once again, we generate the set of linear constraints over residence times, and we will show later that the solution of the parameter synthesis problem in some cases boils down to computing multidimensional integrals over this set. The models considered here, similarly to the CTMCs presented in Chapter 4, are continuous-time models. The solution algorithms share similar steps, enforcing the idea that computing multidimensional integrals over domains characterised by sets of linear inequalities, which in turn defines constraints over residence times in states of the system, is a valuable solution technique for a vast class real-time problems on continuous-time systems.

5.7.2 Parameter optimisation

After generating the set of linear inequalities $\mathcal{S}$ we are ready to tackle the parameter synthesis problem, i.e., to find the optimal solution for the set of controllable parameters Γ_c with respect to an objective function $\mathcal{O}$. The optimal solution will be the one that maximises $\mathcal{O}$. We emphasise that there is no single optimal solution. The optimal solution should be the one that best fits the domain of the application. For this reason, we present

two different choices of the objective functions that we believe are relevant for pacemaker applications. The first consists in maximizing the value of an integral over the domain $\mathcal{V}(\Gamma_u)$, i.e.,

$$\text{opt}_v := \underset{\vartheta_c \in \mathcal{V}(\Gamma_c)}{\text{argsup}} \int_{\vartheta_u \in \mathcal{V}(\Gamma_u), (\vartheta_c, \vartheta_u) \in \mathcal{S}} \mathit{Distr}_{\Gamma_u}(d\vartheta_u).$$

The idea of the integral is to find a valuation for the controllable parameters that satisfies the set of constraints $\mathcal{S}$ and that also maximises the probability mass associated with the uncontrollable parameter set. In the above objective function, we assume that the set of uncontrollable parameters, Γ_u, are distributed according to $\mathit{Distr}_{\Gamma_u}$. If $\mathit{Distr}_{\Gamma_u}$ is a discrete probability distribution, then the above objective function can be reduced to a linear programming problem for which standard solution algorithms exist. If $\mathit{Distr}_{\Gamma_u}$ is continuous, then it is always possible to discretise $\mathcal{V}(\Gamma_u)$ or apply Monte-Carlo simulation techniques. In the special case when $\mathit{Distr}_{\Gamma_u}$ is the uniform distribution, the above objective function becomes a volume integral parametric in $\mathcal{V}(\Gamma_c)$ for which efficient solutions also exist [CDKM11, CDKM13b].

In some practical examples it does not suffice to have an optimal solution unless it is also robust (see [FP09, FP07, FP06] for various definitions of robustness). Intuitively, we say that a set of model parameters is robust if a small variation at the values of the model parameters does not affect the validity of the formula under consideration. We show the concept with an abstract example. For instance, consider the problem of finding optimal parameters for a pacemaker. Running Algorithm 13 we find the optimal controllable parameters opt_v for which the pacemaker satisfies the safety property φ. Let opt'_v be a sub-optimal solution, i.e., $\text{opt}'_v < \text{opt}_v$. Now consider that a small change of ϵ in opt_v invalidates φ, whereas the same change in opt'_v does not affect the validity of φ. In this case it makes sense to chose opt'_v rather than opt_v because opt'_v is more "robust". In light of this example, we introduce a new optimal problem (opt_r) that captures the concept of robustness:

$$B_\epsilon(\vartheta) = \{\vartheta' \in \mathcal{V}(\Gamma) \mid ||\vartheta' - \vartheta||_\infty \leqslant \epsilon\},$$

$$\text{opt}_r := \underset{\vartheta_c \in \mathcal{V}(\Gamma_c)}{\text{argsup}} \{\sup_{\epsilon}\{\epsilon \mid \vartheta_u \in \mathcal{V}(\Gamma_u), B_\epsilon((\vartheta_c, \vartheta_u)) \subseteq \mathcal{S}\}\}$$

where the norm $||\vartheta' - \vartheta||_\infty$ for $\vartheta, \vartheta' \in \mathcal{V}(\Gamma)$ and $\Gamma = \{v_1, \ldots, v_n\}$ is defined as $\max\{|\vartheta(v_1) - \vartheta'(v_1)|, \ldots, |\vartheta(v_{n'}) - \vartheta'(v_n)|\}$. Note that the above optimisation problem can be transformed into a linear programming problem.

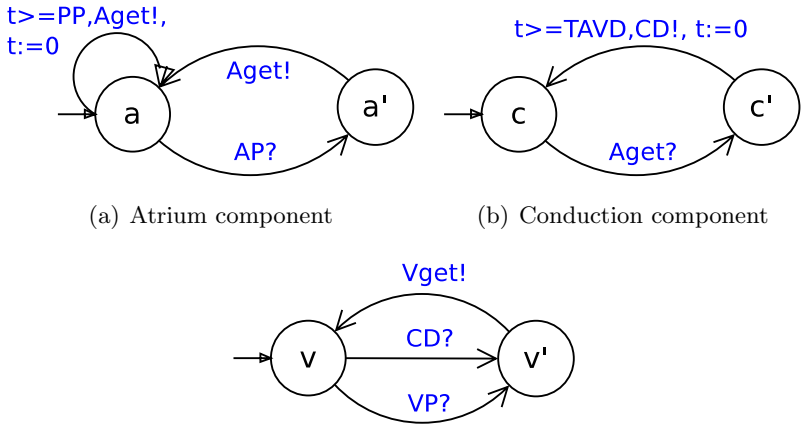
5.7.3 Parameter synthesis case study

In this section we present a pacemaker case study where we apply the techniques described in Section 5.7. The goal is to synthesise one of the parameters of the pacemaker in order

to ensure its correct behaviour, while at the same time optimising the value of a given objective function.

We have implemented all the algorithms in Python for the full logic CMTL, using the NumPy package for numerical computation and the SymPy package for symbolic computation. All linear inequalities are encoded in SymPy.

For this case study, we consider the basic pacemaker model of five components introduced in Section 5.4. We show how to synthesise the parameter $\mathsf{TLRI} - \mathsf{TAVI}$ of the pacemaker LRI component of Figure 5.10(a).



(a) Atrium component (b) Conduction component

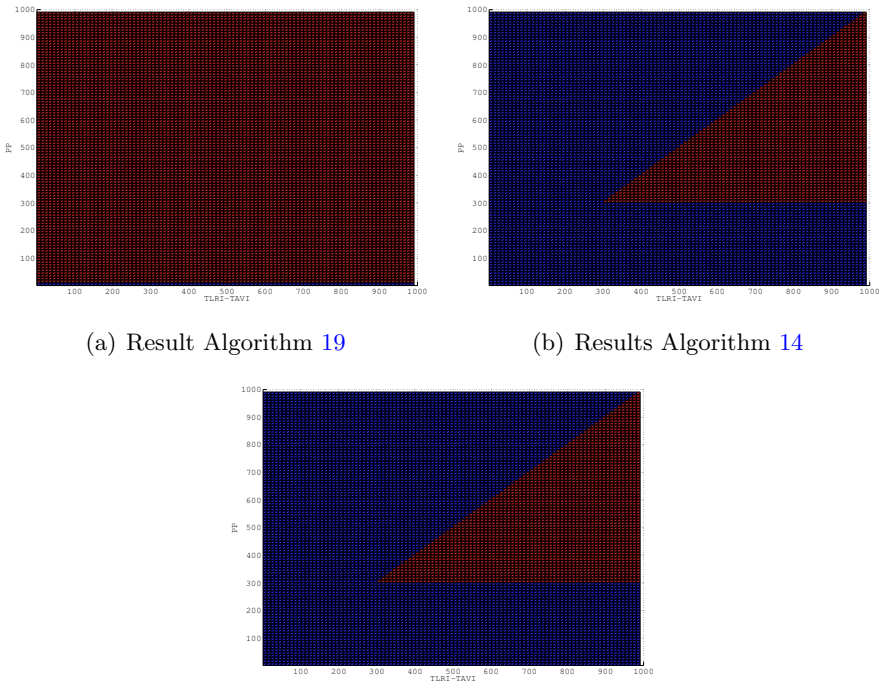
(c) Ventricle component

Figure 5.21: Heart components.

The heart model is composed of three TIOA components (see Figure 5.21): *atrium*, *conduction* and *ventricle*. The *atrium* component (Figure 5.21(a)) is responsible for generating atrial beats. It waits for a signal (action potential) from the SA-node, which is the natural pacemaker of the heart, or from the pacemaker by means of action AP. The waiting time is modelled by a transition labelled with the guard $t \geq \mathsf{PP}$, which defines the firing frequency of the SA-node. The *atrial* component generates atrial beats by means of action Aget. The *conduction* component models the propagation delay of the atrial signal through the atrium and the AV-node. The delay is given by the parameter TAVD. When the action potential originating from the atrium reaches the ventricle, the *conduction* component notifies the *ventricle* component by means of action CD. The *ventricle* component is responsible for generating ventricle beats. It can receive a signal VP from the pacemaker or from the conduction component CD. The *ventricle* component generates ventricle beats by means of action Vget. We emphasise that both PP and TAVD can be estimated from real patient data, and we have done so [LB13] to validate our approach.

We synthesise the pacemaker parameter $\mathsf{TLRI} - \mathsf{TAVI}$ (Figure 5.10(a)), which is the amount of time that the pacemaker waits before delivering an atrial pace. $\mathsf{TLRI} - \mathsf{TAVI}$ is a controllable parameter in our model and its value is critical for the correct functioning

of the pacemaker device. The frequency of the atrial beat PP is instead an uncontrollable variable of the model. We assume that PP and TLRI − TAVI are the only two parameters of the system, both taking values in $[0, 1000]$ (milliseconds). All other parameters are constant. We check the correctness of the pacemaker against the CMTL formula $\varphi = \Box^{[0,\tau]}(\#_0^\tau \texttt{Vget} \geqslant 60 \wedge \#_0^\tau \texttt{Vget} \leqslant 120)$ ($\tau = 60000$ milliseconds), which states that it is always the case that the heart beats (ventricle beat) at least 60 and no more than 120 times in one minute.



(a) Result Algorithm 19

(b) Results Algorithm 14

(c) Intersection between Fig 5.22(a) and Figure 5.22(b)

Figure 5.22: Constraint generation algorithms

We propose an optimised algorithm which has been implemented in order to efficiently solve the parameter synthesis problem. Our solution reduces the complexity of Algorithm 13 for a subclass of CMTL, namely, the safety part of MTL and including BCF. For example, the formula $\varphi = \Box^{[0,\tau]}(\#_0^\tau \texttt{Vget} \geqslant 60 \wedge \#_0^\tau \texttt{Vget} \leqslant 120)$ ($\tau = 60000$ milliseconds) belongs to this subclass. Note that the main disadvantage of Algorithm 13 is that it needs to discretise the set of all parameters and generate constraints for each discretised value. However, as mentioned at the end of Section 5.7, multiple model parameters will share the same set of linear constraints $\mathcal{S}$. Our solution transforms the network of TIOAs into a labelled transition system and generates an untimed path $\rho[\![0]\!]$ such that it contains k Vget events where $k \in \{60, \ldots, 120\}$. This step replaces the function Gen_Path in Algorithm 13. We then use $\rho[\![0]\!]$ to generate the set of linear constraints $\mathcal{S}$ by means of functions Path_Constr_Gen and Constr_Gen. $\mathcal{S}$ will contain the time constraints on model

parameters such that the instantiated timed path ρ contains k `Vget` actions. We start with $k = 60$ and iterate the procedure for all the remaining values in the set $\{61, \ldots, 120\}$. In Figure 5.22 we show three graphs, where we depict the satisfiability of the constraints $\mathcal{S}$ for discretised values of PP and $\mathsf{TLRI} - \mathsf{TAVI}$. The first, Figure 5.22(a), is the result of Algorithm 19 (set of constraints $\mathcal{S}''$ in Algorithm 13), the second Figure 5.22(b) is the result of Algorithm 14 (set of constraints $\mathcal{S}'$ in Algorithm 13) and the third Figure 5.22(c) is the conjunction $\mathcal{S}'' \bigwedge \mathcal{S}'$. In Figure 5.22 the red region denotes the admissible parameter valuations, i.e., the valuations that satisfy the CMTL formula φ, whereas the blue region denotes inadmissible parameter valuations. Recall that PP is an uncontrollable variable, whereas $\mathsf{TLRI} - \mathsf{TAVI}$ is controllable. This means that we should synthesise a value for $\mathsf{TLRI} - \mathsf{TAVI}$ such that the validity of the formula φ is preserved for *any* value of PP. As discussed in Section 5.7.2, the optimal parameter valuation might not be robust. In this example, we have that a value for $\mathsf{TLRI} - \mathsf{TAVI}$ of around 1000 is optimal. This is due to the fact that when $\mathsf{TLRI} - \mathsf{TAVI}$ is in that range the pacemaker model satisfies the formula φ for the largest set of parameter valuations of PP. However, setting $\mathsf{TLRI} - \mathsf{TAVI}$ to 1000 is not robust from an implementation point of view. In fact, if we have small perturbation of $\mathsf{TLRI} - \mathsf{TAVI}$, say from 1000 to 1001, the formula φ is invalidated. A more robust choice is to pick values for $\mathsf{TLRI} - \mathsf{TAVI}$ around 750 (and this is the value returned by Algorithm 13 using the robust optimisation function). Picking the value of $\mathsf{TLRI} - \mathsf{TAVI}$ around 750 reduces the number of PP behaviours that we cover. However, in this case, a small change of $\mathsf{TLRI} - \mathsf{TAVI}$ will not invalidate the formula φ. Notice that some major pacemaker manufacturers, such as Boston Scientific [BOS07], suggest that these values be set between 750 and 900, which validates the result of our algorithms. In addition to ensuring the correct number of beats, we can also guarantee that the pacemaker consumes no more than a given amount of energy in an interval of time. This property can be expressed in CMTL by $\varphi = \Box^{[0,\tau]}((10 \cdot \#_0^\tau \mathtt{AP} + 20 \cdot \#_0^\tau \mathtt{VP}) \leqslant \mathsf{E})$, where τ is a time bound as before and E is a given energy bound. The formula states that, for every atrial beat AP and ventricle beat VP, the pacemaker respectively consumes 10 and 20 units of energy, and the total energy consumption should be less than E.

5.8 Summary

In this chapter we have introduced a general model-based framework which can help to develop embedded software for medical devices. We have presented algorithms to perform model checking of real-time properties over medical devices in Section 5.6, as well as parameter synthesis in Section 5.7.

For the model checking problem of Section 5.6, we show how the framework can be used for quantitative analysis of pacemaker software that incorporates stochasticity to model probabilistic switching and noise. The framework can be instantiated with a hybrid

automaton model for embedded pacemaker software and a hybrid heart model. We have developed a Simulink implementation of the framework that is based on discrete-time simulation semantics and endowed it with a range of quantitative property checks tailored to the verification of pacemakers and expressed as property patterns. The high complexity of the models as well as the ad-hoc property formalisms introduced come at a price: the verification process becomes challenging and we needed to employ approximate model checking techniques. Unfortunately, our current methods are not practical if one wants to use small confidence intervals and error bounds (see Section 5.6.2) due to the high computational complexity of simulating sufficiently many paths. Moreover, some of the assumptions that we make are still too strong to be considered realistic. Consider, for example, the pacing noise introduced in Section 5.4.1. We assume that the pacing noise can be modelled by a normal distribution. However, in order to model the pacing noise more accurately, one can consider additional parameters, such as the influence of the body temperature, vibrations, or the position of the pacing lead that are currently not supported in our implementation.

The synthesis problem of Section 5.7 is even more challenging than the model checking problem. In Section 5.7 we have developed an algorithm to synthesise optimal timing delays for real-time embedded systems modelled as an extension of Timed I/O Automata with priorities and parametric guards. Focusing on medical devices as an application domain, we propose the Counting Metric Temporal Logic (CMTL), an extension of the Metric Temporal Logic with counting formulas, which can express fundamental safety properties for pacemakers, as well as quantitative requirements for energy consumption. We show the feasibility of our approach on a simplified model of the human heart and the pacemaker. Some parameters of the human heart model have been validated using real electrocardiogram data. We synthesise an important parameter that is critical for the safety of the pacemaker, while also affecting energy usage. The results of our case study are encouraging as they comply with specifications from major pacemaker manufacturers. We use techniques similar to Chapter 4, namely, we reduce the problem to computing a volume integral over a complex domain. The domain of integration is defined as a set of linear inequalities over the residence time in system states (as for Chapter 4). As expected, the drawback of our approach is the high complexity of computing the constraints that guarantee the satisfaction of a given CMTL formula. Unfortunately, the algorithms do not scale for real-life examples. For example, the parameter synthesis algorithm was not capable to analyse paths of length greater than 10 even for simple CMTL formulas. The set of linear constraints that the algorithm returned was too large to store on a typical desktop computer.

Although we are aware that the techniques presented in this chapter have some limitations due to the high complexity of the algorithms, these techniques represent the first step for future work. For instance, it is clear that we should concentrate on developing

algorithms to efficiently solve volume integrals, which in turn would significantly speed up the algorithms presented in this thesis.

The aim of the work in this thesis was to propose a first model-based framework that enables to perform verification and parameter synthesis for medical devices. We achieved this goal while introducing a broad range of new features, such as probabilities, complex heart behaviours, and continuous variables to monitor cells' voltage levels that were not considered before. We believe that the framework introduced in this chapter, after optimising some of the algorithms that we have developed, can assist in the design and verification of software embedded in real medical devices.

Chapter 6

Conclusions and future work

6.1 Conclusions

In this thesis we have developed a framework for the analysis of real-time properties over different classes of continuous-time systems, such as Continuous-Time Markov Chains (CTMCs), networks of Timed I/O Automata and networks of Hybrid I/O Automata. We have considered a broad range of real-time specifications, including properties expressed as Timed Automata, as Metric Temporal Logic formulas, as Counting Metric Temporal Logic formulas and as Linear Duration Properties.

The main drawback of our approach is the state space explosion. In most of our verification algorithms, we need to enumerate all the discrete paths of length up to N for a given step bound N. The procedure is exponential in the step bound N. Furthermore, in most cases presented in this thesis, the verification problem reduces to computing a volume integral over a complex domain. The domain of integration depends on the set of linear constraints that we generate during the course of the verification algorithm. Such a set of constraints is usually very large, making the solution of the volume integral challenging.

We have provided a collection of results, ranging from theoretical foundations to verification tools, which provide a framework for the formal analysis of real-time properties. Chapter 4 and Chapter 5 include the main contribution of this thesis.

- **Chapter 4.** In this chapter we have considered the problem of verifying continous-time Markov chains against different real-time specifications. More specifically, we presented model checking algorithms for CTMCs against properties specified as Metric Temporal Logic (MTL) formula, as Timed Automata (TA) or as Linear Duration Property (LDP). The central question of this chapter is the following: given a CTMC $\mathcal{C}$ and a real-time property φ, "what is the probability of the set of timed paths of $\mathcal{C}$ that satisfy the property φ over a time interval of fixed, bounded length?". We provide approximation algorithms to solve these problems. The solutions for the model checking problem of CTMCs against MTL formulas, TAs or

LDP follow the following steps. First, we determine a step bound N such that it is enough to consider all the timed paths of $\mathcal{C}$ with at most N discrete jumps, to approximate the desired probability up to ε. Then, we take each discrete path, ς, of length at most N, and we generate timed constraints over variables determining the residence time of each state along ς, depending on the real-time specification under consideration. Next, we formulate a multidimensional integral that allows us to express the probability of the set of timed paths determined by the discrete path and the associated timed constraints. The algorithm concludes by summing up all such probabilities and yielding the final result. All the algorithms have been implemented in Matlab [MAT13]. To our knowledge, there were no previous solutions to the model checking problem of CTMCs against MTL formulas, general TA or LDPs.

The key idea of Chapter 4 is to generate sets of linear constraints and solve multidimensional integrals over those sets. It turns out that such a technique is a powerful tool to analyse real-time properties. In fact, we can always use sets of linear constraints to characterise the validity of a timed property in a path, independently from the formalism chosen, as long as the property under consideration can be related to the residence time of a state in a path of the CTMC.

- **Chapter 5.** In this chapter we have tackled two main problems: model checking networks of Hybrid I/O Automata; and synthesising model parameters for networks of Timed I/O Automata. The model checking problem consists in taking a network of Hybrid I/O Automata which represents the human heart, a network of Timed I/O Automata which represents a pacemaker specification, a property pattern which represents a safety property that we want to verify, and applying approximation algorithms to determine the probability of the property being satisfied in the composed system. The parameter synthesis problem takes as input a network of Timed I/O Automata for the human heart and the pacemaker, a Counting Metric Temporal Logic formula, and an objective function, and finds the values of the parameters of the composed system such that the objective function is maximised and the Counting Metric Temporal Logic formula is satisfied in the system. We have implemented our algorithms in Simulink [SIM13] and in Python, and provided extensive case studies which are encouraging.

  The main contribution of Chapter 5 is to provide a framework which is tailored for the verification of pacemakers. Our framework can be instantiated with different types of human heart models. The human heart models can be patient-specific through the analysis of real data. In Chapter 5 we give an example translation from ECG data to a heart model based on Hybrid I/O Automata. The pacemaker model that we use is taken from a specification of a real pacemaker manufacturer,

Boston Scientific. Our novelty is to introduce probabilities for the formal verification of pacemaker algorithms and to enhance the pacemaker model in order to consider energy consumption and signal loss. We also consider a probabilistic mode switching behaviour of the human heart, which allows us to represent different, normal and diseased, human heart behaviours. Probabilities are essential when dealing with physiologically-relevant systems. In fact, the human heart, as well as real pacemakers, are inherently probabilistic systems. We believe that the work in this chapter can be considered as a theoretical and practical initial building block for the formal verification of pacemakers and other similar medical devices.

6.2 Future work

The work done in this thesis has raised many interesting research questions that still need to be addressed. In this section we discuss several of them. One issue with both model checking CTMCs and medical devices is the state space explosion. Investigating abstraction techniques as well as symbolic methods could yield a tangible improvement in the execution time of the algorithms, thus making them amenable for real-life applications.

- **Chapter 4.** The verification of continuous-time Markov chains is a well established field of computer science. However, there are still open questions. For example, it was recently showed in [NP10] that, under the *bounded-variability assumption* (BVA), an MTL formula can be transformed into a deterministic timed automaton. Roughly, a timed path satisfies the BVA if there exist δ and k such that for every interval of the form $[t, t+\delta]$ the number of discrete jumps is at most k. Clearly, this is related to the bound on discrete jumps in $[0, T]$. However, the BVA is a "global" assumption over $[0, \infty)$, so it does not apply to time-bounded verification. Also, at the moment it is not clear how to bound the error under this assumption. It would be interesting to investigate whether one could obtain a deterministic timed automaton from MTL under our assumption of finite jumps over $[0, T]$, which could yield an alternative way to solve the model checking problem of CTMCs against MTL formulas, based on the work in [CHKM09]. Another natural question is how to tackle the traditional (time-unbounded) verification. The general scheme introduced in this thesis still works. However, one cannot guarantee approximation to within the given error bound ε, which implies loss of precision. It is also interesting to study specifications combining duration properties and temporal properties (in traditional real-time logics, e.g., MTL). The verification of these specifications would be challenging. Extending the current work to continuous-time Markov decision processes is another possible direction.

- **Chapter 5.** The verification of medical devices is still in its infancy. Thus, there are

many possible future directions. We mention here only some of them. For example, it is clear that we should focus on transformation techniques from real patient data to physiologically relevant heart models. The most sophisticated heart models are at a cellular level. Although this allows one to construct a fairly detailed heart model, it has a major drawback. It is hard to gather patient-specific data at a cellular level. The issue is that the patient data that we can usually gather comes in the form of ECG signals, monitored from the torso of the patient. An ECG signal is a super-composition of the electrical activity of multiple cells. It is hard to map back the ECG signal to the action potential of single cells. This is a hard problem known as the *inverse problem* [PCN+10, MB98].

Another important direction is to enhance the pacemaker model even further. Although we have already augmented the pacemaker model with energy consumption and signal loss, real pacemakers are far more complex. To give an example, consider the case in which a person with a pacemaker is running. It is clear that the heart beat of that person will increase. This, however, it is not a critical situation and modern pacemakers can sense when the patient is exercising through the use of accelerometers which are embedded into the devices. Our pacemaker model lacks models for those advanced sensors.

From a pure modelling point of view, it would be interesting to consider more sources of randomness. A very common situation that would be quite obvious to consider is the effect of movements of the leads of the pacemaker. In fact, during the lifetime of a pacemaker, there is a small chance that the leads implanted within the heart move slightly, compromising the correct functioning of the device. This and other sources of randomness should be taken into account during the validation process.

Last, but not least, we should think of how to expand our analysis to more complicated medical devices. Pacemakers are just a starting point that we decided to begin with due to their relative simplicity. An interesting future direction to follow would be to consider cardiac defibrillators. These are devices similar to pacemakers which not only produce a heart beat with an electric impulse, but also prevent one by giving a strong electrical shock. This is used in the case the heart is beating too fast. Cardiac defibrillators, although more complex than pacemakers, share multiple similarities with them. In fact, cardiac defibrillators can be considered as enhanced pacemakers with multiple functionalities. It should not be too hard to extend our analysis to those types of medical devices. A different approach is necessary, instead, when one wants to consider more complex medical devices such as neurostimulators. For neurostimulators the models developed for pacemakers may not apply, and for such reasons more investigation is needed.

Bibliography

[ACFE08] É. André, T. Chatain, L. Fribourg, and E. Encrenaz. An Inverse Method for Parametric Timed Automata. *Electron. Notes Theor. Comput. Sci.*, 223:29–46, December 2008.

[ACH97] R. Alur, C. Courcoubetis, and T. A. Henzinger. Computing Accumulated Delays in Real-Time Systems. *Formal Methods in System Design*, 11(2):137–155, 1997.

[ACHH92] R. Alur, C. Courcoubetis, T. A. Henzinger, and P-H. Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer, 1992.

[AD94] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[AF10] É. André and L. Fribourg. Behavioral Cartography of Timed Automata. In Antonn Kucera and Igor Potapov, editors, *RP*, volume 6227 of *Lecture Notes in Computer Science*, pages 76–90. Springer, 2010.

[AFH96] R. Alur, T. Feder, and T. A. Henzinger. The Benefits of Relaxing Punctuality. *J. ACM*, 43(1):116–146, January 1996.

[AH93] R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. *Inf. Comput.*, 104(1):35–77, 1993.

[AKRS08] R. Alur, A. Kanade, S. Ramesh, and K. C. Shashidhar. Symbolic Analysis for Improving Simulation Coverage of Simulink/Stateflow models. In *EMSOFT*, pages 89–98, 2008.

[AKV98] R. Alur, R. P. Kurshan, and M. Viswanathan. Membership Questions for Timed and Hybrid Automata. In *RTSS*, pages 254–263. IEEE Computer Society, 1998.

[ASSB00] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model-Checking Continuous-Time Markov Chains. *ACM Trans. Comput. Logic*, 1(1):162–170, July 2000.

[BCH+07] C. Baier, L. Cloth, B. R. Haverkort, M. Kuntz, and M. Siegle. Model Checking Markov Chains with Actions and State Labels. *IEEE Trans. Software Eng.*, 33(4):209–224, 2007.

[BCH+11] B. Barbot, T. Chen, T. Han, J.-P. Katoen, and A. Mereacre. Efficient CTMC Model Checking of Linear Real-Time Objectives. In *TACAS*, pages 128–142, 2011.

[BCM+90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *LICS*, pages 428–439. IEEE Computer Society, 1990.

[BES93] A. Bouajjani, R. Echahed, and J. Sifakis. On Model Checking for Real-Time Properties with Durations. In *LICS*, pages 147–159, 1993.

[BFT01] A. Bemporad, K. Fukuda, and F. D. Torrisi. Convexity Recognition of the Union of Polyhedra. *Comput. Geom.*, 18(3):141–154, 2001.

[BHHK00] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. On the Logical Characterisation of Performability Properties. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *ICALP*, volume 1853 of *Lecture Notes in Computer Science*, pages 780–792. Springer, 2000.

[BHHK03] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Model-Checking Algorithms for Continuous-Time Markov Chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003.

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

[BOS07] PACEMAKER System Specification, Boston Scientific, 2007.

[BR07] V. Bruyre and J.-F. Raskin. Real-Time Model-Checking: Parameters Everywhere. *Logical Methods in Computer Science*, 3(1), 2007.

[Bra05] M. S. Branicky. Introduction to Hybrid Systems. In Dimitrios Hristu-Varsakelis and William S. Levine, editors, *Handbook of Networked and Embedded Control Systems*, pages 91–116. Birkhäuser, 2005.

[BRP13] M. M .Bersani, M. Rossi, and P. S. Pietro. Deciding Continuous-Time Metric Temporal Logic with Counting Modalities. In Parosh Aziz Abdulla and Igor Potapov, editors, *RP*, volume 8169 of *Lecture Notes in Computer Science*, pages 70–82. Springer, 2013.

[BVF+11] S. Butterbach, B. Vulturescu, C. Forgez, G. Coquery, and G. Friedrich. Lead-Acid Battery Model for Hybrid Energy Storage. In *Vehicle Power and Propulsion Conference (VPPC), 2011 IEEE*, pages 1–5, 2011.

[CAM06] G. D. Clifford, F. Azuaje, and P. McSharry. *Advanced Methods and Tools for ECG Data Analysis*. Artech House Publishers, 2006.

[CDKM11] T. Chen, M. Diciolla, M. Z. Kwiatkowska, and A. Mereacre. Time-Bounded Verification of CTMCs against Real-Time Specifications. In Uli Fahrenberg and Stavros Tripakis, editors, *FORMATS*, volume 6919 of *Lecture Notes in Computer Science*, pages 26–42. Springer, 2011.

[CDKM12a] T. Chen, M. Diciolla, M. Z. Kwiatkowska, and A. Mereacre. Quantitative Verification of Implantable Cardiac Pacemakers. In *RTSS*, pages 263–272. IEEE Computer Society, 2012.

[CDKM12b] T. Chen, M. Diciolla, M. Z. Kwiatkowska, and A. Mereacre. Verification of Linear Duration Properties over Continuous-Time Markov Chains. In Thao Dang and Ian M. Mitchell, editors, *HSCC*, pages 265–274. ACM, 2012.

[CDKM13a] T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre. Quantitative Verification of Implantable Cardiac Pacemakers over Hybrid Heart Models. *Information and Computation*, 2013.

[CDKM13b] T. Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre. Verification of Linear Duration Properties over Continuous-Time Markov Chains. *Transactions on Computer Logic*, 2013.

[CDKM13c] T. Chen, M. Diciolla, M. Z. Kwiatkowska, and A. Mereacre. A Simulink Hybrid Heart Model for Quantitative Verification of Cardiac Pacemakers. In Calin Belta and Franjo Ivancic, editors, *HSCC*, pages 131–136. ACM, 2013.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[CGP99] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

[CHKM09] T. Chen, T. Han, J.-P. Katoen, and A. Mereacre. Quantitative Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications. In *LICS*, pages 309–318, 2009.

[CHKM11] T. Chen, T. Han, J.-P. Katoen, and A. Mereacre. Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications. *Logical Methods in Computer Science*, 7(1), 2011.

[CHR91] Z. Chaochen, C. A. R. Hoare, and A. P. Ravn. A Calculus of Durations. *Inf. Process. Lett.*, 40(5):269–276, 1991.

[CJLX94] Z. Chaochen, Z. Jingzhong, Y. Lu, and L. Xiaoshan. Linear Duration Invariants. In *FTRTFT*, pages 86–109, 1994.

[Clo06] L. Cloth. *Model Checking Algorithms for Markov Reward Models.* PhD thesis, Univerisity of Twente, The Netherlands, 2006.

[CNS10] G. D. Clifford, S. Nemati, and R. Sameni. An Artificial Vector Model for Generating Abnormal Electrocardiographic Rhythms. *Physiological Measurement*, 31(5):595, 2010.

[Con78] J.B. Conway. Functions of One Complex Variable. *Graduate Texts in Math*, 1978.

[CPW01] C.G. Cassandras, D. L. Pepyne, and Y. Wardi. Optimal Control of a Class of Hybrid Systems. *Automatic Control, IEEE Transactions on*, 46(3):398 –415, March 2001.

[CY88] C. Courcoubetis and M. Yannakakis. Verifying Temporal Properties of Finite-State Probabilistic Programs. In *FOCS*, pages 338–345. IEEE Computer Society, 1988.

[CY95] C. Courcoubetis and M. Yannakakis. The Complexity of Probabilistic Verification. *J. ACM*, 42(4):857–907, 1995.

[Dah58] G. Dahlquist. *Stability and Error Bounds in the Numerical Integration of Ordinary Differential Equations.* PhD thesis, Stockholm College, 1958.

[Dav93] M. H. A. Davis. *Markov Models and Optimization.* Chapman and Hall, 1993.

[DHK13] F. Dannenberg, E. M. Hahn, and M. Kwiatkowska. Computing Cumulative Rewards using Fast Adaptive Uniformisation. In Ashutosh Gupta and Thomas A. Henzinger, editors, *Computational Methods in Systems Biology*, volume 8130 of *Lecture Notes in Computer Science*, pages 33–49. Springer Berlin Heidelberg, 2013.

[DHS09] S. Donatelli, S. Haddad, and J. Sproston. Model Checking Timed and Stochastic Properties with CSL^{TA}. *IEEE Trans. Software Eng.*, 35(2):224–240, 2009.

[DKM13] M. Diciolla, M. Z. Kwiatkowska, and A. Mereacre. Synthesising Model Parameters for Cardiac Pacemakers Using Timed I/O Automata, 2013. Report.

[DKNP06] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker. A Formal Analysis of Bluetooth Device Discovery. *Int. Journal on Software Tools for Technology Transfer*, 8(6):621–632, 2006.

[DKTT13] F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A. J. Turberfield. DNA Walker Circuits: Computational Potential, Design, and Verification. In David Soloveichik and Bernard Yurke, editors, *DNA Computing and Molecular Programming*, volume 8141 of *Lecture Notes in Computer Science*, pages 31–45. Springer International Publishing, 2013.

[Doy07] L. Doyen. Robust Parametric Reachability for Timed Automata. *Inf. Process. Lett.*, 102(5):208–213, 2007.

[ESY12] K. Etessami, A. Stewart, and M. Yannakakis. Polynomial Time Algorithms for Branching Markov Decision Processes and Probabilistic Min(Max) Polynomial Bellman Equations. *CoRR*, abs/1202.4798, 2012.

[FP06] G. E. Fainekos and G. J. Pappas. Robustness of Temporal Logic Specifications. In Klaus Havelund, Manuel Núñez, Grigore Rosu, and Burkhart Wolff, editors, *FATES/RV*, volume 4262 of *Lecture Notes in Computer Science*, pages 178–192. Springer, 2006.

[FP07] G. E. Fainekos and G. J. Pappas. Robust Sampling for MITL Specifications. In Jean-François Raskin and P. S. Thiagarajan, editors, *FORMATS*, volume 4763 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2007.

[FP09] G. E. Fainekos and G. J. Pappas. Robustness of Temporal Logic Specifications for Continuous-Time Signals. *Theor. Comput. Sci.*, 410(42):4262–4291, 2009.

[GBF+11] R. Grosu, G. Batt, F. H. Fenton, J. Glimm, C. Le Guernic, S. A. Smolka, and E. Bartocci. From Cardiac Cells to Genetic Regulatory Networks. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 396–411. Springer, 2011.

[GH01] D. Giannakopoulou and K. Havelund. Automata-Based Verification of Temporal Properties on Running Programs. In *Automated Software Engineering, 2001. (ASE 2001). Proceedings. 16th Annual International Conference on*, pages 412–416, 2001.

[GH10] D. P. Guelev and D. V. Hung. Reasoning about QoS Contracts in the Probabilistic Duration Calculus. *Electr. Notes Theor. Comput. Sci.*, 238(6):41–62, 2010.

[GJM93] S.E. Greenhut, J.M. Jenkins, and R.S. MacDonald. A Stochastic Network Model of the Interaction Between Cardiac Rhythm and Artificial Pacemaker. *Biomedical Engineering, IEEE Transactions on*, 40(9):845–858, 1993.

[GO09] A. O. Gomes and M. V. Oliveira. Formal Specification of a Cardiac Pacing System. In *Proceedings of the 2nd World Congress on Formal Methods*, FM '09, pages 692–707, Berlin, Heidelberg, 2009. Springer-Verlag.

[GSC+09] R. Grosu, S. A. Smolka, F. Corradini, A. Wasilewska, E. Entcheva, and E. Bartocci. Learning and Detecting Emergent Behavior in Networks of Cardiac Myocytes. *Commun. ACM*, 52(3):97–105, 2009.

[GT07] M. Gribaudo and M. Telek. Fluid Models in Performance Analysis. In Marco Bernardo and Jane Hillston, editors, *SFM*, volume 4486 of *Lecture Notes in Computer Science*, pages 271–317. Springer, 2007.

[HCH+02] B. R. Haverkort, L. Cloth, H. Hermanns, J.P. Katoen, and C. Baier. Model Checking Performability Properties. In *DSN*, pages 103–112. IEEE Computer Society, 2002.

[Hen91] T. A. Henzinger. *The Temporal Specification and Verification of Real-Time Systems.* PhD thesis, Stanford University, Palo Alto, California, USA, 1991.

[Hig02] N. J. Higham. *Accuracy and Stability of Numerical Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

[HJ90] R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge University Press, 1990.

[HJ94] H. Hansson and B. Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing*, 6:102–111, 1994.

[HKNT98] G. Horton, V.G. Kulkarni, D.M. Nicol, and K.S. Trivedi. Fluid Stochastic Petri Nets: Theory, Applications, and Solution Techniques. *European Journal of Operational Research*, 105(1):184–201, February 1998.

[HKPV95] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's Decidable about Hybrid Automata? In *Journal of Computer and System Sciences*, pages 373–382. ACM Press, 1995.

[HR06] Y. Hirshfeld and A. M. Rabinovich. Expressiveness of Metric Modalities for Continuous Time. In D. Grigoriev, J. Harrison, and E. A. Hirsch, editors, *CSR*, volume 3967 of *Lecture Notes in Computer Science*, pages 211–220. Springer, 2006.

[HRSV01] T. Hune, J. Romijn, M. Stoelinga, and F. W. Vaandrager. Linear Parametric Model Checking of Timed Automata. In Tiziana Margaria and Wang Yi 0001, editors, *TACAS*, volume 2031 of *Lecture Notes in Computer Science*, pages 189–203. Springer, 2001.

[HUL94] J.B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms: Fundamentals*, volume 305. Springer-Verlag, 1994.

[HZ99] D. V. Hung and C. Zhou. Probabilistic Duration Calculus for Continuous Time. *Formal Asp. Comput.*, 11(1):21–44, 1999.

[HZ07] D. V. Hung and M. Zhang. On Verification of Probabilistic Timed Automata against Probabilistic Duration Properties. In *RTCSA*, pages 165–172, 2007.

[Jen53] A. Jensen. Markov Chains as an Aid in the Study of Markov processes. *Skand. Aktuarietidskrift*, 36:87–91, 1953.

[JPC+10] Z. Jiang, M. Pajic, A. Connolly, S. Dixit, and R. Mangharam. Real-Time Heart Model for Implantable Cardiac Device Validation and Verification. In *ECRTS*, pages 239–248. IEEE Computer Society, 2010.

[JPM12a] Z. Jiang, M. Pajic, and R. Mangharam. Cyber-Physical Modeling of Implantable Cardiac Medical Devices. *Proceedings of the IEEE*, 100(1):122–137, 2012.

[JPM+12b] Z. Jiang, M. Pajic, S. Moarref, R. Alur, and R. Mangharam. Modeling and Verification of a Dual Chamber Implantable Pacemaker. In Cormac Flanagan and Barbara König, editors, *TACAS*, volume 7214 of *Lecture Notes in Computer Science*, pages 188–203. Springer, 2012.

[JY10] Karl Henrik Johansson and Wang Yi, editors. *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2010, Stockholm, Sweden, April 12-15, 2010*. ACM, 2010.

[KHM+98] T. J. Koo, F. Hoffmann, F. Ho Mann, H. Shim, B. Sinopoli, and S. Sastry. Hybrid Control of an Autonomous Helicopter. In *IFAC Workshop on Motion Control*, pages 285–290, 1998.

[KLSV10] D. K. Kaynar, N. A. Lynch, R. Segala, and F. W. Vaandrager. *The Theory of Timed I/O Automata, Second Edition*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2010.

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[KNP12] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic Verification of Hermans Self-Stabilisation Algorithm. *Formal Aspects of Computing*, 24(4):661–670, 2012.

[Koy90] R. Koymans. Specifying Real-Time Properties with Metric Temporal Logic. *Real-Time Systems*, 2(4):255–299, 1990.

[KP12] M. Knapik and W. Penczek. Bounded Model Checking for Parametric Timed Automata. *Timed Petri Nets and Other Models of Concurrency*, 5:141–159, 2012.

[KPSY99] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Decidable Integration Graphs. *Inf. Comput.*, 150(2):209–243, 1999.

[KV01] O. Kupferman and M. Y. Vardi. Model Checking of Safety Properties. *Form. Methods Syst. Des.*, 19(3):291–314, October 2001.

[LB13] H. Lea-Banks. The Rate-Adaptive Pacemaker: Developing Simulations and Applying Patient ECG Data, Internship report, University of Oxford, 2013.

[LGS96] J. Lygeros, D. N. Godbole, and S. Sastry. Verified Hybrid Controllers for Automated Vehicles. *IEEE Transactions on Automatic Control*, 43:522–539, 1996.

[LHZ97] X. Li, D. V. Hung, and T. Zheng. Checking Hybrid Automata for Linear Duration Invariants. In R. K. Shyamasundar and Kazunori Ueda, editors, *ASIAN*, volume 1345 of *Lecture Notes in Computer Science*, pages 166–180. Springer, 1997.

[LKM10] J. Lian, H. Krätschmer, and D. Müssig. Open Source Modeling of Heart Rhythm and Cardiac Pacing. *The Open Pacing, Electrophysiology & Therapy Journal*, pages 28–44, 2010.

[LP08] R. Lassaigne and S. Peyronnet. Probabilistic Verification and Approximation. *Ann. Pure Appl. Logic*, 152(1-3):122–131, 2008.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *STTT*, 1(1-2):134–152, 1997.

[LS83] J. D. Lehmann and S. Shelah. Reasoning with Time and Chance. In Josep Daz, editor, *ICALP*, volume 154 of *Lecture Notes in Computer Science*, pages 445–457. Springer, 1983.

[LSVW95] N. A. Lynch, R. Segala, F. W. Vaandrager, and H. B. Weinberg. Hybrid I/O automata. In *Hybrid Systems*, pages 496–510, 1995.

[LZ01] J. B. Lasserre and E. S. Zeron. A Laplace Transform Algorithm for the Volume of a Convex Polytope. *J. ACM*, 48(6):1126–1140, 2001.

[MAT13] MATLAB. *version 8.2 (R2013b)*. The MathWorks Inc., Natick, Massachusetts, 2013.

[MB98] R. S. MacLeod and D.H. Brooks. Recent Progress in Inverse Problems in Electrocardiology. *Engineering in Medicine and Biology Magazine, IEEE*, 17(1):73–83, 1998.

[MCTS03] P.E. McSharry, G.D. Clifford, L. Tarassenko, and L.A. Smith. A Dynamical Model for Generating Synthetic Electrocardiogram Signals. *IEEE Transactions on Biomedical Engineering*, 50(3):289 –294, march 2003.

[MLF08] H. Macedo, P. Larsen, and J. Fitzgerald. Incremental Development of a Distributed Real-Time Model of a Cardiac Pacing System Using VDM. In Jorge Cuellar, Tom Maibaum, and Kaisa Sere, editors, *FM 2008: Formal Methods*, volume 5014 of *Lecture Notes in Computer Science*, pages 181–197. Springer Berlin / Heidelberg, 2008.

[MS09] D. Méry and N. K. Singh. Pacemaker's Functional Behaviors in Event-B. Rapport de recherche, MOSEL - INRIA Lorraine - LORIA, 2009.

[MT09] S. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. Cambridge University Press, New York, NY, USA, 2nd edition, 2009.

[Nat07] A. Natale. *Handbook of Cardiac Electrophysiology*. CRC Press, 2007. Editor: Frank Marchlnski.

[NNN10] B. F. Nielsen, F. Nielson, and H. R. Nielson. Model Checking Multivariate State Rewards. In *QEST*, pages 7–16. IEEE Computer Society, 2010.

[NP10] D. Nickovic and N. Piterman. From MTL to Deterministic Timed Automata. In Krishnendu Chatterjee and Thomas A. Henzinger, editors, *FORMATS*, volume 6246 of *Lecture Notes in Computer Science*, pages 152–167. Springer, 2010.

[NPK+05] G. Norman, D. Parker, M. Kwiatkowska, S. Shukla, and R. Gupta. Using Probabilistic Model Checking for Dynamic Power Management. *Formal Aspects of Computing*, 17(2):160–176, August 2005.

[Pat93] K.R. Pattipati. *Markov-Reward Models and Hyperbolic Systems*. University of Connecticut, Department of Electrical and Systems Eng., 1993.

[PCN+10] A. J. Pullan, L. K. Cheng, M. P. Nash, A. Ghodrati, R. MacLeod, and D. H. Brooks. The Inverse Problem of Electrocardiography. In PeterW. Macfarlane, A. van Oosterom, Olle Pahlm, Paul Kligfield, Michiel Janse, and John Camm, editors, *Comprehensive Electrocardiology*, pages 299–344. Springer London, 2010.

[PJ08] M. Prandini and H. Jianghai. Application of Reachability Analysis for Stochastic Hybrid Systems to Aircraft Conflict Prediction. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 4036 –4041, December 2008.

[Pnu77] A. Pnueli. The Temporal Logic of Programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977.

[PRI] PRISM Website. https://www.prismmodelchecker.org/.

[QQM01] Q. Qiu, Q. Qu, and M.Pedram. Stochastic Modeling of a Power-Managed System-Construction and Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(10):1200 –1217, 2001.

[QS94] M. A. Qureshi and W. H. Sanders. Reward Model Solution Methods with Impulse and Rate Rewards: an Algorithm and Numerical Results. *Performance Evaluation*, 20(4):413 – 436, 1994.

[Rab10] A. M. Rabinovich. Complexity of Metric Temporal Logics with Counting and the Pnueli Modalities. *Theor. Comput. Sci.*, 411(22-24):2331–2342, 2010.

[SC85] A. P. Sistla and E. M. Clarke. The Complexity of Propositional Linear Temporal Logics. *J. ACM*, 32(3):733–749, July 1985.

[SIM13] SIMULINK. The MathWorks Inc., Natick, Massachusetts, 2013.

[SK11] A. Sharma and J.-P. Katoen. Weighted Lumpability on Markov Chains. In Edmund M. Clarke, Irina Virbitskaite, and Andrei Voronkov, editors, *Ershov Memorial Conference*, volume 7162 of *Lecture Notes in Computer Science*, pages 322–339. Springer, 2011.

[TH04] P. H. Thai and D. V. Hung. Verifying Linear Duration Constraints of Timed Automata. In *ICTAC*, pages 295–309, 2004.

[TZT10] A. Tuan, M. C. Zheng, and Q. T. Tho. Modeling and Verification of Safety Critical Systems: A Case Study on Pacemaker. In *SSIRI*, pages 23–32. IEEE Computer Society, 2010.

[Var85] M. Y. Vardi. Automatic Verification of Probabilistic Concurrent Finite-State Programs. In *FOCS*, pages 327–338, 1985.

[Wik] Wikipedia Website. http://en.wikipedia.org/wiki/Medical_device/.

[YEGS05] P. Ye, E. Entcheva, R. Grosu, and S. A. Smolka. Efficient Modeling of Excitable Cells Using Hybrid Automata. In *In Proceedings of Computational Methods in System Biology*, pages 216–227, 2005.

[YEGS08] P. Ye, E. Entcheva, R. Grosu, and S. A. Smolka. Modelling Excitable Cells Using Cycle-Linear Hybrid Automata. *IET Syst. Biol.*, 2(1):24–32, 2008.

[YKNP04] H. L. S. Younes, M. Z. Kwiatkowska, G. Norman, and D. Parker. Numerical vs. Statistical Probabilistic Model Checking: an Empirical Study. In Kurt Jensen and Andreas Podelski, editors, *TACAS*, volume 2988 of *Lecture Notes in Computer Science*, pages 46–60. Springer, 2004.

[ZHL08] M. Zhang, D. V. Hung, and Z. Liu. Verification of Linear Duration Invariants by Model Checking CTL properties. In John S. Fitzgerald, Anne Elisabeth Haxthausen, and Hüsnü Yenigün, editors, *ICTAC*, volume 5160 of *Lecture Notes in Computer Science*, pages 395–409. Springer, 2008.

[ZPC10] P. Zuliani, A. Platzer, and E. M. Clarke. Bayesian Statistical Model Checking with Application to Simulink/Stateflow Verification. In Johansson and Yi [JY10], pages 243–252.