Statistical Model Checking of Rich Models and Properties

Poulsen, Danny Bøgsted

DOI (link to publication from Publisher):
10.5278/vbn.phd.engsci.00031

Publication date:
2015

Document Version
Publisher's PDF, also known as Version of record

STATISTICAL MODEL CHECKING OF RICH MODELS AND PROPERTIES

BY
DANNY BØGSTED POULSEN

DISSERTATION SUBMITTED 2015

AALBORG UNIVERSITY
DENMARK

Ph.D. thesis

Statistical Model Checking of Rich Models and Properties

Danny Bøgsted Poulsen

Aalborg Univeristy
Department of Computer Science

Dissertation submitted:	October 27, 2015
Assistant PhD supervisor:	Professor Kim Guldstrand Larsen Aalborg University
PhD committee:	Associate Professor René Rydhof Hansen (chairman) Aalborg University.
	Professor Martin Fränzle Oldenburg University
	Professor Wang Yi Uppsala University
PhD Series:	Faculty of Engineering and Science, Aalborg University

ISSN (online): 2246-1248
ISBN (online): 978-87-7112-527-6

Published by:
Aalborg University Press
Skjernvej 4A, 2nd floor
DK – 9220 Aalborg Ø
Phone: +45 99407140
aauf@forlag.aau.dk
forlag.aau.dk

Printed in Denmark by Rosendahls, 2016

Abstract

Software is in increasing fashion embedded within safety- and business critical processes of society. Errors in these embedded systems can lead to human casualties or severe monetary loss. Model checking technology has proven formal methods capable of finding and correcting errors in software. However, software is approaching the boundary in terms of the complexity and size that model checking can handle. Furthermore, software systems are nowadays more frequently interacting with their environment hence accurately modelling such systems requires modelling the environment as well - resulting in undecidability issues for the traditional model checking approaches.

Statistical model checking has proven itself a valuable supplement to model checking and this thesis is concerned with extending this software validation technique to stochastic hybrid systems. The thesis consists of two parts: the first part motivates why existing model checking technology should be supplemented by new techniques. It also contains a brief introduction to probability theory and concepts covered by the six papers making up the second part. The first two papers are concerned with developing online monitoring techniques for deciding if a simulation satisfies a property given as a $\text{WMTL}_{[a,b]}$ formula. The following papers develops a framework allowing dynamical instantiation of processes, in contrast to traditional static encoding of systems. A logic, QDMTL, is developed to express properties of these dynamically evolving systems. The fifth paper shows how stochastic hybrid automata are useful for modelling biological systems and the final paper is concerned with showing how statistical model checking is efficiently distributed. In parallel with developing the theory contained in the papers, a substantial part of this work has been devoted to implementation in Uppaal SMC.

Dansk Sammenfatning

Software er i stigende grad indlejret i vores sikkerhedskritiske processer. Fejl i indlejrede softwaresystemer kan have store økonomiske konsekvenser og resultere i tab af menneskeliv. Model checking har vist, at man med formelle metoder kan finde fejl i software, som er svære at finde med traditionel testning. Kompleksititeten af software har efterhånden nået en så kritisk størrelse at model checking ikke længere kan håndtere modellernes størrelse. Endvidere interagerer software i højere grad med dets omgivelser hvilket nødvendiggør modellering af disse som del af systemet. Dette gør model checking-spørgsmålet uafgørbart. Statistical model checking har vist sit værd som et alternativ til model checking, og denne afhandling vil videreudvikle teknikken til stochastiske hybride systemer. Afhandlingen består af to dele: Den første del giver belæg for at model checking værktøjer skal suppleres af en anden teknologi. Den indeholder også en kort introduktion til sandsynlighedsregning, samt koncepter fra de seks artikler den anden del består af. De første artikler i anden del omhandler udviklingen af en online monitoreringsteknik for en $\text{WMTL}_{[a,b]}$ formel. De efterfølgende artikler udvikler et framework, der tillader dynamisk instantiering af processeer og logikken QDMTL introduceres til at udtrykke egenskaber af disse dynamiske systemer. Den femte artikel viser, at stochastiske hybride automater kan bruges til at modellere biologiske systemer. Til slut viser den sjette artikel, hvordan statistical model checking kan blive effektivt distribueret. Simultant med udvikling af teorien i artiklerne har en stor del af arbejdet været tilegnet implementering i Uppaal SMC.

Mandatory Page

Thesis Title:	Statistical Model Checking of Rich Models and Properties
Ph.D. Student:	Danny Bøgsted Poulsen
Supervisor:	Prof. Kim Guldstrand Larsen, Aalborg University
Co-supervisor:	Assoc. Prof. Alexandre David, Aalborg University

The main body of this thesis consist of the following papers:

[35] Peter E. Bulychev, Alexandre David, Kim Guldstrand Larsen, Axel Legay, Guangyuan Li, Danny Bøgsted Poulsen, and Amélie Stainer. Monitor-Based Statistical Model Checking for Weighted Metric Temporal Logic. In *LPAR*, volume 7180 of *LNCS*, pages 168–182, 2012. Paper A in this thesis

[34] Peter E. Bulychev, Alexandre David, Kim G. Larsen, Axel Legay, Guangyuan Li, and Danny Bøgsted Poulsen. Rewrite-Based Statistical Model Checking of WMTL. In *RV*, volume 7687 of *LNCS*, pages 260–275, 2012. Paper B in this thesis

[52] Alexandre David, Kim G. Larsen, Axel Legay, and Danny Bøgsted Poulsen. Statistical Model Checking of Dynamic Networks of Stochastic Hybrid Automata. *ECEASST*, 66, 2013. Paper C in this thesis

[53] Alexandre David, Kim G. Larsen, Axel Legay, Guangyuan Li, and Danny Bøgsted Poulsen. Quantified Dynamic Metric Temporal Logic for Dynamic Networks of Stochastic Hybrid Automata. In *14th International Conference on Application of Concurrency to System Design, ACSD 2014, Tunis La Marsa, Tunisia, June 23-27, 2014*, pages 32–41. IEEE, 2014. ISBN 978-1-4799-4281-7. doi:10.1109/ACSD.2014.21. Paper D in this thesis

[54] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical model checking for biological systems. *STTT*, 17(3):351–367, 2015. doi:10.1007/s10009-014-0323-4. Paper E in this thesis

[32] Peter Bulychev, Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Danny Bøgsted Poulsen. Checking & Distributing Statistical Model Checking. In Alwyn E. Goodloe and Suzette Person, editors, *4th NASA FORMAL METHODS SYMPOSIUM*, volume 7226 of *LNCS*, pages 449–463. Springer, 2012. Paper F in this thesis

This thesis has been submitted for assessment in partial fulfillment of the PhD degree. The thesis is based on the submitted or published scientific papers which are listed above. Parts of the papers are used directly or indirectly in the extended summary of the thesis. As part of the assessment, co-author statements have been made available to the assessment committee and are also available at the Faculty. The thesis is not in its present form acceptable for open publication but only in limited and closed circulation as copyright may not be ensured.

Acknowledgements

First, a big thanks to Kim G. Larsen, my supervisor and collaborator. Without your many comments and suggestions this thesis and the papers within had never been written.
An equally big thanks goes to Alexandre David who in addition to co-authoring my papers acted as co-supervisor and gave invaluable help for the implementations inside Uppaal. A thank you also go to Guangyuan Li and Axel Legay that kindly invited me to visit their research institutions in Beijing and Rennes respectively. Life is undoubtedly more than work, and without the mental breaks I got with my friends in Aalborg Soobak, I had never finished this thesis.

Contents

Part I
Introduction

Software Verification

1

For decades computers were "nice-to-have" but not essential for a household. Contrary to this, computers are nowadays such an integral part of our lifes that we no longer think about it: few people consider an interactive toy for kids has an integrated computer system or that their washing machine is software controlled. Software is "hidden"/embedded in our equipment to such an extent that the only times we "worry" about it is when it breaks down. Software systems embedded in gadgets, called *embedded software systems*, are characterised by their tight integration with devices and that they respond to stimuli from their environment. The embedding of software in safety and business critical operations has made methods for guaranteeing correctness a necessity. One approach is using a development process that minimises errors by starting software validation early on in the development phase. Early software validation and regression validation is promoted by the agile software development movement to ensure quality of the final product. In industry, the traditional validation technique is *testing* where the software is exposed to certain parameter settings and the result compared to a precomputed correct answer. Testing can, however, only verify a small fraction of the computational paths of a software system and requires manually constructed test cases. The testing community has developed measures for quantifying the proportion of computational paths covered by a set of test cases. Although testers attempt to maximise these measures, it is well-known that all paths are never covered except for the most trivial systems. A supplement to testing is *formal methods* that rely on mathematically anchored techniques to exclude the existence of errors. One such method is model checking (MC) [12, 39, 40, 101].

1.1 Model Checking

Model checking is a formal method developed independently by Clarke and Emerson [40] and Queille and Sifakis [101]. Clarke, Emerson and Sifakis were jointly awarded the Turing Award for their pioneering work on MC. The key

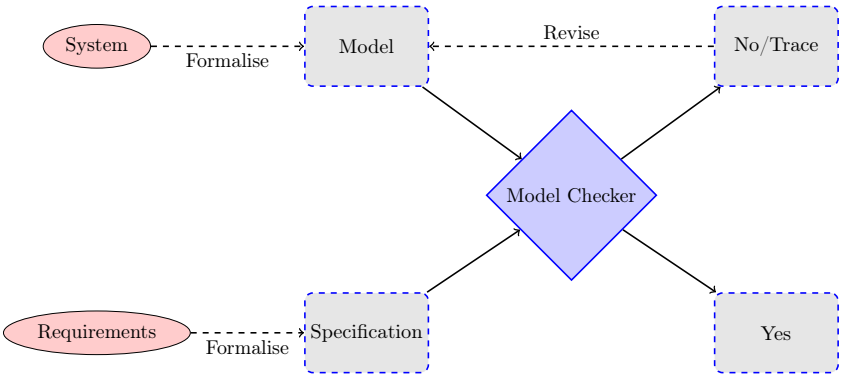


Figure 1.1: The model checking procedure. A model checker is given a model and property and returns either Yes or No. In case the result is no the model checker provides a diagnostic trace that can be used to resolve the problem.

hypothesis in MC is that errors are not only due to flaws in the implementation, but also due to flaws in the design and that errors might be caught earlier if the design is verified prior to implementation. Model checking relies on a formal description of both the design and the requirements. In Figure 1.1 a schematic overview of the three phases of the model checking process is provided. These are

1. model the design in a modelling language,
2. specify the requirements in a specification language and
3. pass the model and requirement to a model checking engine.

The MC algorithm explores the entire model description in search of a computational path violating the requirement. A violation returns a diagnostic trace witnessing the error. From this witness, the design can either revised or it can be determined the requirement was wrong.

In classical model checking a system is modelled by a finite state machine (FSM) and specifications given in linear temporal logic (LTL) [100] or computational tree logic (CTL) [40]. An important feature of FSMs is their compositionality that allows modelling individual components and compose them to an overall system. The "catch" is that a linear state space increase of each component results in an exponential increase of the entire system state space. This *state space explosion* is problematic for model checkers because MC algorithms perform an exhaustive search of the state space. Research in MC has to a large extend focused on developing efficient data structures for representing the state space and developing techniques that minimises the state space to be searched. Combined with increased computing power this has made MC applicable to industrial cases and not only toy examples [69, 70, 88].

A series of classic MC tools has surfaced over the years: an incomplete list includes SPIN [79], NuSMV [38], PAT [114], FDR3 [62] and CWB [94]. A notable example of these is SPIN having its own model description language (Promela) and giving specifications in LTL. The notability of SPIN lies in its use of partial order reduction where certain parts of the state space is not searched because other parts are guaranteed to reach the same state as those skipped. Whereas SPIN reduces the state space to be searched, a tool like NuSMV has focused on representing the state space efficiently using Binary Decision Diagrams. Despite these successful tools, MC has some downfalls: firstly the modelling language must adhere to certain restrictions to make the model checking problem decidable, secondly the state space explosion puts a natural limit to the size of models MC can be applied to.

1.2 Probabilistic Model Checking

Designs of computer systems may be made such that errors are allowed to occur but remain unlikely due to randomisation: an example of this is the IEEE 802.15.4 CSMA/CA[113] protocol that specifies how nodes in a network avoid collisions. The nodes are unable to sense if a collision occurs while transmitting so to avoid collisions they choose a random time in which they sense for other senders. In case no one has been transmitting the node initiates transmission. It is clear that this does not guarantee collision freedom and any attempt of verifying collision-freedom is futile. Instead of the qualitative yes/no answer provided by MC, we are interested in the probability that a collision occurs and assert it is lower than some desired threshold value. Probabilistic model checking (PMC) [42, 115] allows calculating this probability given a probabilistic model of the design. The model is given as a discrete time Markov chain or a continuous time Markov chain. As for standard MC there is tool support for PMC of which the most well-known is PRISM [84]. The PMC problem is as difficult as the classic MC problem thus PMC also requires certain restrictions to the modelling language and also suffers from the state space explosion problem. In addition, PMC uses costly matrix multiplications that further limits the size of models applicable to PMC.

1.3 Real Time Model Checking

An issue not covered by classical MC is real-time guarantees e.g. guaranteeing that an air-bag deploys within milliseconds of a collision. To give such guarantees, modelling formalisms must accommodate for specifying timed behaviours and specification languages must be able to express timing constraints. A successful modelling formalism is timed automaton (TA) [6, 7] that extends FSM with clock variables on which behaviour can be conditioned.

Timed specifications are made in metric temporal logic (MTL) [83] being an extension of LTL or Timed CTL [8] being an extension of CTL. Adding time into the model may give an infinite state space, making an exhaustive

concrete state space exploration incomprehensible. Luckily, there exists a finite partitioning into *symbolic* states making the MC problem decidable. Real time model checking is supported by various tools [30, 49, 86] of which UPPAAL [86] is established as the standard tool for real time model checking. UPPAAL uses an extended timed automata formalism for modelling and a subset of Timed CTL as specification language.

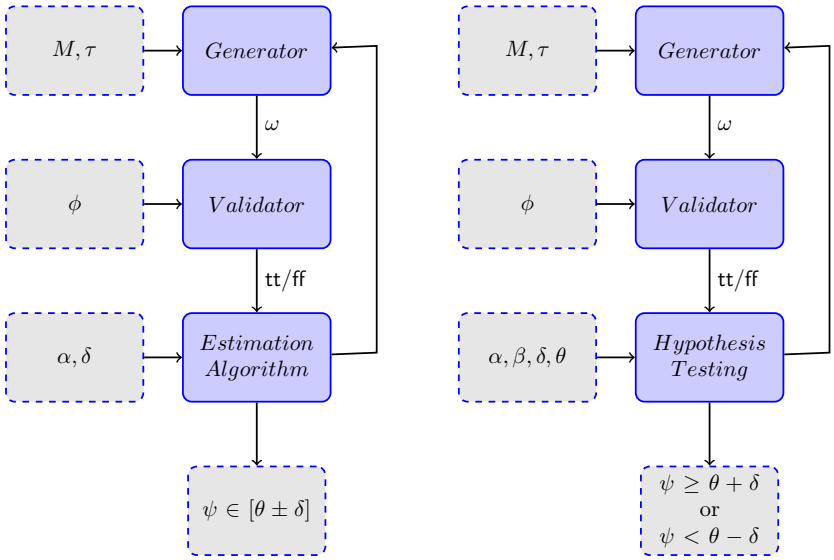
1.4 Hybrid Model Checking

The natural step after real time model checking is generalising time to physical quantities such as heat. This is not a purely academic idea as software systems already control objects that depend on physical quantities - those software systems are called *Hybrid systems*. Hybrid systems exhibit a tighter integration with their environment than embedded systems: an early example is the stability control of fighter air planes - such air crafts are built aerodynamically unstable and are only manoeuvrable because a computer system constantly adjusts the flight controls. Other examples are the envisioned smart houses that adjusts heating in rooms to meet the comfort level of its inhabitants [48] and smart grids adjusting power consumption of individual house to load balance. Inherent to these systems is the interaction/control with their environment, thus, to be accurately modelled their environment must be taken into account. Behaviours of physical quantities are commonly modelled by differential equations and modelling formalisms that incorporates such exist. An example is *Hybrid Automata* [72] used by the tool HYTECH [74]. HYTECH can analyse a sub-class of Hybrid Automata, *Linear Hybrid Automata*, exact and over-approximate the behaviour of a general hybrid automaton by a linear hybrid automaton. Clearly, if the over-approximated model does not violate a requirement then the original model does not - on the other hand a violation in the over-approximated system does not prove a violation. The over-approximating strategy of HYTECH is the only option for analysing hybrid systems as the MC problem is undecidable for general hybrid systems.

1.5 Statistical Model Checking

Software is getting increasingly complex and without increased computing power, it is unlikely that traditional MC software can keep up with the complexity introduced by software of the future. Younes [123] and Sen et al. [107] independently developed an approximate software verification technique not facing the state space explosion problem. The technique was coined statistical model checking (SMC) and attributed Younes. Prior to these works Larsen and Skou [85] applied ideas similar to SMC while defining probabilistic bisimulation. The corner stone of SMC is, that a fully stochastic model can be simulated and that it is possible to validate if a simulation satisfy a specification. After generating a number of simulations and counting the number of satifying simulations, hypothesis testing can be applied to decide if the probability of satisfying the

requirement exceeds a threshold. Alternative, a confidence interval can be made if we want to estimate the probability.



(a) An estimation algorithm estimates the unknown probability ψ to an interval $[\theta \pm \delta]$ with confidence α.

(b) A hypothesis testing algorithm asserts if the unknown probability ψ is greater than a threshold $\theta + \delta$ or less than $\theta - \delta$.

Figure 1.2: Architectural overview of a statistical model checker. The generator component produces a simulation, ω, of the model M of length τ. The validator component validates if the run satisfies the requirement ϕ and produces tt/ff for satisfaction and violation respectively. The Estimation/Hypothesis testing algorithm takes some algorithm specific parameters.

In Figure 1.2 a high level overview of a SMC tool is presented: A *generator* accepts a fully stochastic model M and outputs a simulation, ω, of a user specified length, τ. Afterwards, a *validator* component gets that simulation and a property ϕ and returns true (tt) or false (ff) to a statistical algorithm (either *hypothesis testing* or *estimation algorithm*) taking some additional parameters. The feedback loop from the statistical algorithm in Figure 1.2 indicates the statistical algorithm may need additional samples to draw a conclusion. If another sample is needed can be determined in two ways: either (1) the statistical algorithm precomputes a number of samples required to draw a conclusion or (2) a sequential algorithm is used where, after each generated sample, it is determined if a new sample is needed by keeping track of the current number of satisfying samples. The required statistical algorithms have been known for a long time thus the key components that need to be developed for SMC are efficient generation and validation techniques.

Numerous standalone SMC tools have already been developed: YMER [120], VESTA [109], PVESTA [4], PLASMA [80] and COSMOS [13] and traditional model checking tools, UPPAAL [86] and PRISM [84], also started incorporating SMC features. Interestingly, many of these tools inherit their modelling and specification formalisms from MC tools. This gives modellers means to circumvent the state space explosion problem, but it does not give the full advantage of SMC. An exception to this is the work by Zuliani et al. [124] doing SMC for Simulink® models and UPPAAL SMC [45] which, during the work of this thesis, has been extended to incorporate features from hybrid systems and dynamically reconfigurable systems.

1.6 Dynamic Systems

The tighter integration between software and environment is not the only problem faced by MC tools. Software is often designed as a network where computational units may join and leave at will and researchers are talking about *the internet of things* where all utility appliances are connected by a network, making the appliances controllable from virtually anywhere. This produces a new type of reconfigurable systems where the environment is allowed to change the conditions under which a system operates. Many MC tools rely on a static of encoding the system thus such dynamical systems are not modelled naturally in them. Process algebras, such as CCS [91] and CSP [77], allow instantiating processes on the fly using a recursion operator, and the π-calculus [92, 93] extended CCS to express mobility of processes by sending channel names between processes. Hence, The idea of dynamically reconfigurable systems is not completely new to the formal methods community, but few tools uses π-calculus as their modelling language. Exceptions are the "Stochastic Pi Machine" [99] developed by Microsoft Research using the π-calculus with a stochastic semantics to model biological systems and the Mobility Workbench [116].

1.7 Research Objectives

The previous sections gave an overview of MC and some issues regarding modelling the behaviourally rich and complex systems of the future with MC tools. One issue is that the state space explosion restricts MC to certain sizes of systems. A second issue is the nature of software: software is being more tightly integrated with its environment, thus modelling its behaviour requires modelling physical aspects. It is however known that the MC problem is undecidable for general hybrid systems. Statistical model checking has already been proven a promising alternative to MC but has yet to separate itself from the modelling and specification formalisms of MC.

The goal of this thesis is to bring the SMC approach to the realms of real-time, hybrid and dynamically reconfigurable systems. The work is divided into 4 subgoals:

1. developing requirement validation techniques required by the SMC algorithm for various specification formalisms,
2. extending the timed automaton formalism to hybrid systems and provide a stochastic semantics and run generation technique for SMC,
3. extending the hybrid automaton formalism to dynamically reconfigurable systems with a stochastic semantics and run generation technique for SMC and
4. develop a specification formalism along with a validation technique for the formalism of 3.

Probability Theory

2

This section recalls basic concepts from probability theory and establishes notation used throughout the thesis. The content is based on text books by Ash and Doléans-Dade [11] and Olofsson [96]. Probability theory reasons on the outcome of phenomena that appear random due to human inability to collect enough information to "calculate" the result. A σ-algebra collects all the events relevant to an experiment where an experiment is any process with a random outcome.

Definition 1 (σ-algebra)**.** A σ-algebra over the set Ω is a family $\mathcal{F} \subseteq 2^{\Omega}$ where

- $\Omega \in \mathcal{F}$,
- if $F_0, F_1, \ldots, \in \mathcal{F}$ then $(\bigcup_{i \in \mathbb{N}} F_i) \in \mathcal{F}$ i.e. $\mathcal{F}$ is closed under countable union and
- if $F \in \mathcal{F}$ then $(\Omega \setminus F) \in \mathcal{F}$ i.e. $\mathcal{F}$ is closed under complement.

Notice the requirements imply that $\emptyset \in \mathcal{F}$ since $\Omega \in \mathcal{F}$ and $\Omega \setminus \Omega = \emptyset$. Also, by De Morgan's laws, a σ-algebra is closed under countable intersection. Two canonical σ-algebras definable for any set Ω are $\{\emptyset, \Omega\}$, the indiscrete σ-algebra of Ω, and $\mathcal{P}(\Omega) = \{F \mid F \subseteq \Omega\}$, the discrete σ-algebra of Ω. In addition to a structure defining the events of an experiment, a measure defining the probability of each event is needed: a *probability measure* for a σ-algebra, $\mathcal{F}$, assigns a value between 0 and 1 to each event $F \in \mathcal{F}$, with 0 meaning that F almost surely will not happen and 1 meaning F is almost certain to happen.

Definition 2 (Measurable Space)**.** A measurable space is a pair $(\Omega, \mathcal{F})$ where Ω is a set and $\mathcal{F}$ is a σ-algebra over Ω.

Definition 3 (Probability Space)**.** A probability space is a triple $(\Omega, \mathcal{F}, \mathtt{m})$ where $(\Omega, \mathcal{F})$ is a measurable space and $\mathtt{m} : \mathcal{F} \to [0, 1]$ with the conditions

- $\mathtt{m}(\Omega) = 1$ and
- if $F_0, F_1 \ldots$ are pair-wise disjoint sets in $\mathcal{F}$ then $\mathtt{m}(\bigcup_{i \in \mathbb{N}} F_i) = \sum_{i \in \mathbb{N}} \mathtt{m}(F_i)$.

The last requirement of Definition 3 asserts the probability measure $\mathtt{m}$ is a σ-additive set function. If a fair die is thrown once, then the probability that the outcome is a two or a six is $\frac{2}{6}$ because the probability of getting two is $\frac{1}{6}$ and so is the probability of getting a six.

Lemma 1. Let $(\Omega, \mathcal{F}, \mathtt{m})$ be a probability space and let $F_1, F_2 \in \mathcal{F}$ then

- $\mathtt{m}(\Omega \setminus F_1) = 1 - \mathtt{m}(F_1)$,
- $\mathtt{m}(F_1 \setminus F_2) = \mathtt{m}(F_1) - \mathtt{m}(F_2 \cap F_1)$,
- $\mathtt{m}(F_1 \cup F_2) = \mathtt{m}(F_1) + \mathtt{m}(F_2) - \mathtt{m}(F_1 \cap F_2)$ and
- if $F_1 \subseteq F_2$ then $\mathtt{m}(F_1) \leq \mathtt{m}(F_2)$.

In Definition 4 the important notion of conditional probability is defined. Conditional probabilities captures the fact that the probability of an event may change if we have extra knowledge: if a die is thrown once the probability of getting a six is $\frac{1}{6}$, but if somehow we know that the result is will either be a six or a five then the probability of getting a six is $\frac{1}{2}$.

Definition 4 (Conditional Probability)**.** Let $(\Omega, \mathcal{F}, \mathtt{m})$ be a probability space and let $F_1 \in \mathcal{F}$ be an event s.t. $\mathtt{m}(F_1) > 0$ then for any event $F_2 \in \mathcal{F}$ we define the conditional probability of F_2 given F_1 as

$$\mathtt{m}(F_2 \mid F_1) = \frac{\mathtt{m}(F_2 \cap F_1)}{\mathtt{m}(F_1)}$$

2.1 Random Variables

In many cases we are not interested in the exact outcome of an experiment, but a number giving an aggregate view on the result. A *random variable* obtains a value after performing an experiment.

Definition 5. Let $(\Omega_1, \mathcal{F}_1, \mathtt{m})$ be a probability space and $(\Omega_2, \mathcal{F}_2)$ be a measurable space then a $(\Omega_2, \mathcal{F}_2)-$valued random variable is a function $X : \Omega_1 \to \Omega_2$ such that for any $F \in \mathcal{F}_2$,

$$X^{-1}(F) = \{f \in \Omega_1 \mid X(f) \in F\} \in \mathcal{F}_1.$$

Definition 5 states that a random variable is a function mapping from a probability space to a measurable space and that given a random variable $X : (\Omega_1, \mathcal{F}_1, \mathtt{m}_1) \to (\Omega_2, \mathcal{F}_2)$ one can always define a measure $\mathtt{m}_2$ on $(\Omega_2, \mathcal{F}_2)$ as $\mathtt{m}_2(F) = \mathtt{m}_1(X^{-1}(F))$ for all $F \in \mathcal{F}_2$.

Example 1. Consider the experiment of two consecutive throws of a single die. Then for this experiment $\Omega = \{1, \ldots, 6\} \times \{1, \ldots, 6\}$ and $\mathcal{F}$ is the discrete σ-algebra of Ω. Since all outcomes are equally likely, we consider as probability measure $\mathtt{m} : \mathcal{F} \to [0, 1]$ the unique one such that $\mathtt{m}(\{x\}) = \frac{1}{|\Omega|} = \frac{1}{36}$ for all $x \in \Omega$. If we are only interested in the sum of the outcomes after the two consecutive throws, we can capture it by defining random variable $X : \Omega \to \{2, 3, \ldots, 12\}$ and $X(x_1, x_2) = x_1 + x_2$.

Assume we are interested in the probability that the resulting sum is 4 i.e. the probability that "$X = 4$". We can find this probability by calculating $X^{-1}(\{4\}) = \{(1, 3), (2, 2), (3, 1)\}$ and considering the probability of this set according to $\mathtt{m}$:

$$\mathtt{m}(\{(1,3),(2,2),(3,1)\}) = \mathtt{m}(\{(1,3)\}) + \mathtt{m}(\{(2,2)\}) + \mathtt{m}(\{(3,1)\}) = \frac{1}{36} + \frac{1}{36} + \frac{1}{36} = \frac{3}{36} = \frac{1}{12}.$$

Definition 5 is general in the codomain of the random variable. For the remainder we only consider real-valued random variables i.e. those with codomain given by the measurable space $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ where $\mathcal{B}(\mathbb{R})$ denotes the Borel σ-algebra on $\mathbb{R}$.

Considering again the experiment of Example 1; we may only be interested in the probability that the total number of eyes is less than some threshold k. For a random variable X we call this function its cumulative distribution function (cdf).

Definition 6. Let $(\Omega, \mathcal{F}, \mathtt{m})$ be a probability space and let X be a $(\mathbb{R}, \mathcal{B}(\mathbb{R}))-$valued random variable. The cumulative distribution function, $\mathtt{F}_X$, of X is

$$\mathtt{F}_X(x) = \mathtt{m}(\{f \in \Omega | X(f) \leq x\})$$

A *discrete random variable* assumes values from a countable subset of $\mathbb{R}$. Let N be the countable set of values a discrete random variable X assumes, then the

cdf of X is a step function with a discontinuity at each $x \in N$. The magnitude of each discontinuity is given by a function $\gamma : \mathbb{R} \to [0, 1]$, called the probability mass function (pmf), where $\gamma(x)$ gives the probability that $X = x$. If $\mathtt{F}_X$ is the cdf of X then $\mathtt{F}_X(x) = \sum_{t \leq x} \gamma(t)$.

A *continuous random variable* is a random variable Y with cdf, $\mathtt{F}_Y$, for which there exists a Lebesgue-integrable function $\mu : \mathbb{R} \to \mathbb{R}_{\geq 0}$ such that $\mathtt{F}_Y(x) = \int_{-\infty}^{x} \mu(t)\,\mathrm{d}t$ for all x. The function μ is called the probability density function (pdf) of Y.

For a random variable X with pmf (pdf) γ (μ) we write $X \sim \gamma$ ($X \sim \mu$) to denote the random variable obtains its probability mass (density) from that function. We say the variable is *distributed* according to γ (μ).

For a $(\mathbb{R}, \mathcal{B}(\mathbb{R}))-$valued random variable X mapping from $(\Omega, \mathcal{F}, \mathtt{m})$ we denote by $\mathbb{P}(X \leq x)$, $\mathtt{m}(X^{-1}(\{y \in \mathbb{R} \mid y \leq x\}) = \mathtt{F}_X(x)$. Conditional probabilities are expressed similarly as when using normal probability measures.

Expected Value and Variance

Being based on chance it is natural to consider what the expected value of a random variable is. In layman terms, the expected value is the average value observed if an experiment is repeated a sufficient number of times. For a discrete random variable $X \sim \gamma$ with support N and a continuous random variable $Y \sim \mu$ the expected value is

$$\mathtt{E}[X] = \sum_{x \in N} \left(x \cdot \gamma(x)\right), \text{ and } \mathtt{E}[Y] = \int_{-\infty}^{\infty} y \cdot \mu(y)\,\mathrm{d}y.$$

The expected value tells us where the cdf of a random variable is centred, but not how much results are expected to deviate from the expected value. The *variance* of a random variable is a quantification of this and defined as

$$\mathtt{Var}[X] = \mathtt{E}[(X - \mathtt{E}[X])^2].$$

Often the *standard deviation*, $\mathtt{sd}[X] = \sqrt{\mathtt{Var}[X]}$, is used instead of the variance. Figure 2.1 gives a graphical interpretation of the expected value and the standard deviation.

Lemma 2. Let X be a random variable and $a, b \in \mathbb{R}$ then

- $\mathtt{E}[a \cdot X + b] = a \cdot \mathtt{E}[X] + b$ and
- $\mathtt{Var}[a \cdot X + b] = a^2 \cdot \mathtt{Var}[X]$.

2.2 Distributions

In the following we present the distributions used throughout this thesis. These are the *binomial* distribution, the *exponential* distribution, the *uniform distribution* (both continuous and discrete) and the *normal distribution*. Table 2.1 gives a brief overview of various distributions.

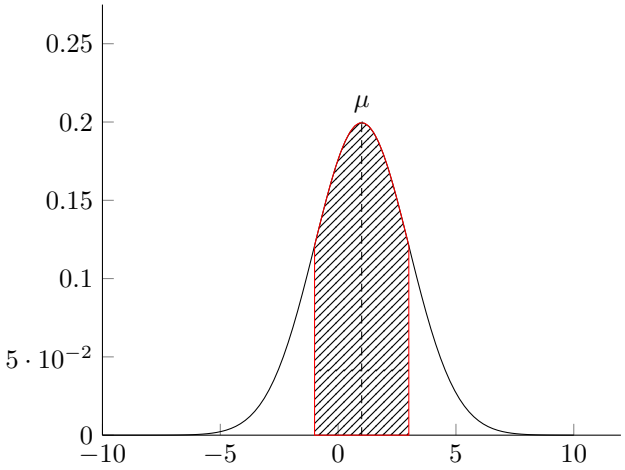


Figure 2.1: Normal distribution with mean value $\mu = 1$ and standard deviation $\sigma = 2$. The scratched area marks the area $[\mu - \sigma, \mu + \sigma]$ and thus the area where most of the probability mass is.

The Uniform Distribution

The uniform distribution is the distribution referred to in everyday language when something is said to be random: every outcome is equally likely. For a discrete random variable (X) the probability mass is divided equally among all possible outcomes thus if X assumes values from the finite set Ω then the pmf is $\gamma(x) = \frac{1}{|\Omega|}$. We denote the pmf for the discrete uniform distribution over a finite set Ω by $\texttt{Uni}^{\gamma}_{\Omega}$.

For a continuous uniform distribution assuming values in $[a, b] \subseteq \mathbb{R}$ the cdf increases with a constant rate. As a result, the pdf is constant in the interval

$X \sim$	$X \in$	$\mu(x)$	$\gamma(x)$	$\texttt{E}[X]$	$\texttt{Var}[X]$
$\texttt{Binom}_{n,p}$	$\{0, \ldots, n\}$		$\binom{n}{x} \cdot p^x \cdot (1-p)^{n-x}$	$n \cdot p$	$n \cdot p \cdot (1-p)$
$\texttt{Uni}^{\mu}_{a,b}$	$[a, b]$	$\frac{1}{b-a}$		$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
$\texttt{Exp}_{\lambda}$	$\mathbb{R}_{\geq 0}$	$\lambda \cdot e^{-\lambda \cdot x}$		$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
$\texttt{Exp}_{\lambda,L}$	$[L, \infty]$	$\texttt{Exp}_{\lambda}(x - L)$		$L + \frac{1}{\lambda}$	$\frac{1}{\lambda^2}$
$\texttt{Norm}_{\mu,\sigma}$	$\mathbb{R}$	$\frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2 \cdot \sigma^2}}$		μ	σ

Table 2.1: Overview of various distributions. The $X \in$ column gives the values the random value may assume, the $\gamma(x)$ column gives the pmf for discrete distributions while the $\mu(x)$ coolumn gives the pdf for continuous distribution. The $\gamma(x)$ and $\mu(x)$ columns only apply for values from the $X \in$ column. Everywhere else the pdf (pmf) is 0.

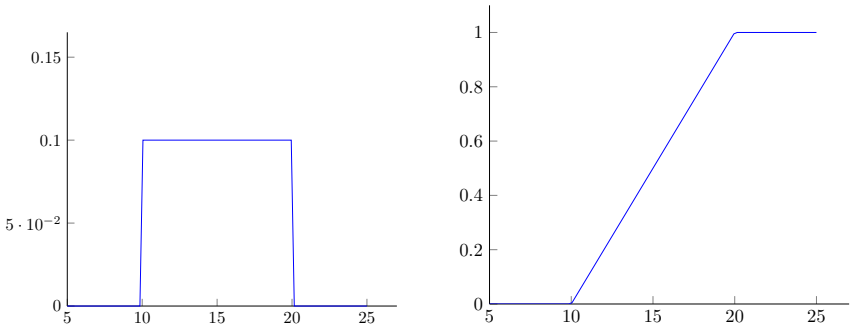
and zero everywhere else. Let this constant be c then we can write the following:

$$\int_a^b c \, \mathrm{d}t = 1 \Leftrightarrow b \cdot c - a \cdot c = 1 \Leftrightarrow c = \tfrac{1}{b-a}, b \neq a$$

hence the pdf is

$$\texttt{Uni}_{a,b}^{\mu}(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}.$$

In Figure 2.2a and Figure 2.2b the pdf and cdf of a uniform distribution over the interval $[10, 20]$ are shown.



(a) Uniform pdf over interval $[10, 20]$. (b) Uniform cdf over interval $[10, 20]$.

Figure 2.2: Example of the uniform distribution.

The Binomial Distribution

The binomial distribution arises after performing the same experiment a number of times. Each experiment has two possible outcomes: success or failure with probability p and $1 - p$ respectively. Assume the experiment is performed n times and let X be a random variable that counts the number of successes i.e. X assumes values in the set $\{0, 1, \ldots, n\}$. The pmf $\texttt{Binom}_{n,p}$ is then given as

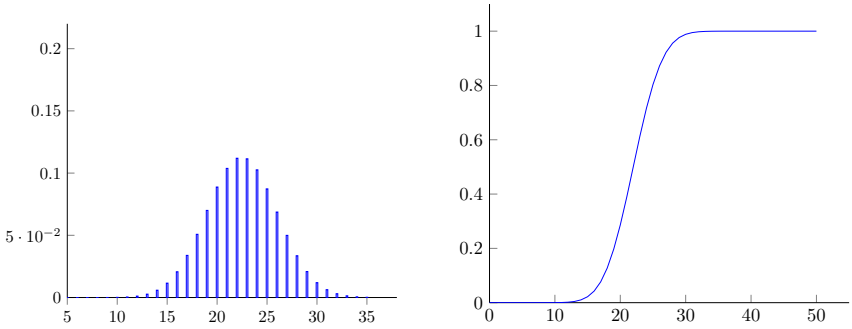
$$\texttt{Binom}_{n,p}(x) = \binom{n}{x} \cdot p^x \cdot (1-p)^{n-x},$$

where $\binom{n}{x}$ is the binomial coefficient.

In Figure 2.3a and Figure 2.3b are shown the pmf and cdf for a binomially distributed random variable.

The Exponential Distribution

The exponential distribution is a continuous distribution arising when expressing a specific property. Assume a component has a functioning time independent of



(a) Binomial pmf with 50 trials and $p = 0.45$ (b) Binomial cdf with 50 trials and $p = 0.45$

Figure 2.3: Example of the binomial distribution.

its age, then the probability that it is working after x time units is independent of its current age y. Let X be the value of the functioning time, then the property of our concern is expressed as

$$\mathbb{P}(X > x + y | X > y) = \mathbb{P}(X > x).$$

By definition of conditional probability we have

$$\mathbb{P}(X > x + y | X > y) = \frac{\mathbb{P}\left(\{X > x + y\} \cap \{X > y\}\right)}{\mathbb{P}(X > y)} = \frac{\mathbb{P}(X > x + y)}{\mathbb{P}(X > y)},$$

and combining with the previous formula $\mathbb{P}(X > x) \cdot \mathbb{P}(X > y) = \mathbb{P}(X > x + y)$. If we let $G(x) = \mathbb{P}(X > x)$ we have the equation

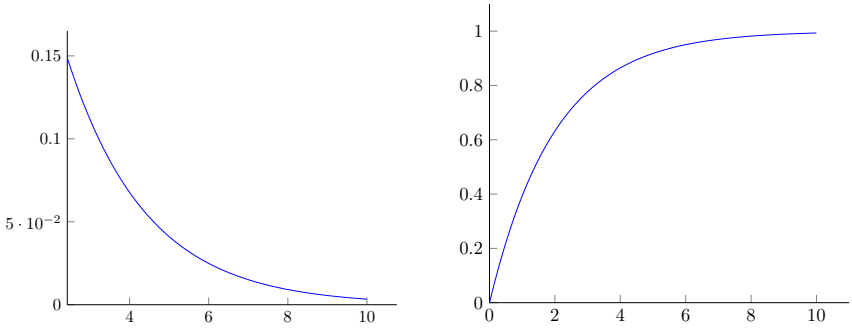
$$G(x) \cdot G(y) = G(x + y),$$

which in our setting has only one meaningful solution [96]: $G(x) = e^{-\lambda \cdot x}$ for some constant λ. Let $\mathtt{F}_X$ be the cdf of X then by construction $\mathtt{F}_X(x) = 1 - G(x)$ and by differentiation we get the pdf

$$\mathtt{Exp}_\lambda = \begin{cases} \lambda \cdot e^{-\lambda \cdot x} & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

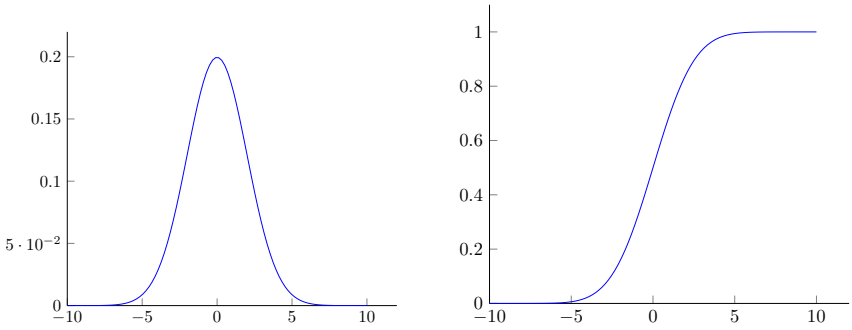
In general we parameterise the exponential distribution by λ. Figure 2.4a and Figure 2.4b show the pdf and cdf of an exponential distribution with $\lambda = 0.5$.

An exponential distribution can be shifted L units on the x-axis such that it assumes values in $[L, \infty[$. This is called a shifted exponential distribution and has the pdf $\mathtt{Exp}_{\lambda,L}(x) = \mathtt{Exp}_\lambda(x - L)$.



(a) Exponential pdf with rate 0.5 (b) Exponential cdf with rate 0.5

Figure 2.4: Example of the exponential distribution.



(a) Normal pdf with $\mu = 0$ and $\sigma = 2$ (b) Normal pdf with $\mu = 0$ and $\sigma = 2$

Figure 2.5: Example of the normal distribution.

The Normal Distribution

The normal distribution is not anchored in a specific experiment (as the binomial distribution was) or in a mathematical property we want to express (such as the uniform and exponential distributions). Instead, the normal distribution is a distribution observed in nature: the height of a human population is normally distributed. Let X be a normally distributed variable and let $\mathtt{sd}[X] = \sigma$ and $\mathtt{E}[X] = \mu$, then its pdf is

$$\mathtt{Norm}_{\mu,\sigma}(x) = \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot e^{-\frac{(x-\mu)^2}{2 \cdot \sigma^2}}.$$

In Figure 2.5 is an example of the pdf and cdf shown with expected value 0 and standard deviation 2. Notice that Figure 2.5 is symmetric around 0. If $\mathtt{sd}[X] = 1$ and $\mathtt{E}[X] = 0$ then X is standard normally distributed and we let Φ

be its cdf. If $X \sim \texttt{Norm}_{\mu,\sigma}$ then $Y = \frac{X-\mu}{\sigma} \sim \texttt{Norm}_{0,1}$. X is said to have been normalised.

An important feature of the normal distribution is, that the sum of independently sampled and identically distributed random variables is normally distributed. A result generally known as the central limit theorem [96].

Theorem 1 (Central Limit Theorem)**.** Let $X_1, \ldots, X_n$ be independent and identically distributed random variables with expected value μ and standard deviation σ and let $S_n = \sum_{i=1}^{n} X_i$. Then for all $x \in \mathbb{R}$

$$\mathbb{P}\left(\frac{S_n - n\mu}{\sigma\sqrt{n}} \leq x\right) \to \Phi(x), \text{ as } n \to \infty$$

Remark 1. Recall that the binomial distribution was the result of counting the number of successes of n experiments and each experiment had success p. Let each experiment be represented by a random variable X_n that assumes 1 if the experiment is successful and 0 if it is not. All these variables has expected value p and standard deviation $\sqrt{p(1-p)}$. Technically, they are Bernoulli distributed. Let $Y = \sum_{i=1}^{n} X_i$ then $Y \sim \texttt{Binom}_{n,p}$. By the central limit theorem we have

$$\mathbb{P}\left(\frac{Y - n \cdot p}{\sqrt{p(1-p)}\sqrt{n}} \leq x\right) \to \Phi(x) \text{ as } n \to \infty.$$

Observe here that $\frac{Y-n\cdot p}{\sqrt{p\cdot(1-p)}\sqrt{n}} = \frac{Y-\texttt{E}[Y]}{\texttt{sd}[Y]}$ thus normalising Y gives a standard normally distributed variable hence with large enough n the binomial distribution can be approximated with a normal distribution.

2.3 Estimating Probability

A frequently occuring problem is that obtaining samples from a distribution is possible, but the exact distribution is unknown or that getting samples from a distribution is easy while computing probabilities is very difficult. The latter is for instance the case for SMC. Despite such difficulties we are nevertheless interested in the probability, p, of a property, thus we *estimate* it by the average of samples on which the property hold.

Let $X_1, X_2, \ldots X_n$ be independent and identically distributed random variables which are 1 with probability p and 0 with probability $1-p$. Let $Y = \sum_{i=1}^{n} X_i$ then $Y \sim \texttt{Binom}_{n,p}$ with $\texttt{E}[Y] = n \cdot p$ and $\texttt{Var}[Y] = n \cdot p \cdot (1-p)$. An *estimator* for p is $Z = \frac{Y}{n}$ with $\texttt{E}[Z] = \frac{\texttt{E}[Y]}{n}$ and $\texttt{Var}[Z] = \frac{\texttt{Var}[Y]}{n^2}$.

With large enough n we have $Z \sim \texttt{Norm}_{\texttt{E}[Z],\texttt{sd}[Z]}$ and by normalisation

$$Q = \frac{Z - \texttt{E}[Z]}{\texttt{sd}[Z]} = \frac{Z - \frac{\texttt{E}[Y]}{n}}{\sqrt{\frac{\texttt{Var}[Y]}{n^2}}} = \frac{Z - \frac{p\cdot n}{n}}{\sqrt{\frac{n\cdot(p)\cdot(1-p)}{n^2}}} = \frac{Z-p}{\sqrt{\frac{(p)\cdot(1-p)}{n}}} \sim \texttt{Norm}_{0,1}$$

Definition 7 (Confidence Interval)**.** Let $X_1, \ldots, X_n$ be random samples from the same distribution and p an unknown parameter of the distribution. If T_1 and T_2 are two functions of the sample such that

$$\mathbb{P}(T_1 \leq p \leq T_2) = \alpha,$$

we call the interval $[T_1, T_2]$ a confidence interval for p with confidence level α.

Now we construct a confidence interval for Q and "translates" this to a confidence interval for Z. First we find values Z_1 and Z_2 such that $\Phi(Z_1) = \frac{1-\alpha}{2}$ and $\Phi(Z_2) = 1 - \frac{1-\alpha}{2}$ which gives

$$\Phi(Z_2) - \Phi(Z_1) = 1 - \tfrac{1-\alpha}{2} - \tfrac{1-\alpha}{2} = 1 - 1 + \alpha = \alpha.$$

Since the normal distribution is symmetric around 0, $Z_1 = -Z_2$ thus

$$\begin{aligned}
\alpha \approx & \mathbb{P}(-Z_2 \leq Q \leq Z_2) = \\
& \mathbb{P}(-Z_2 \leq \frac{Z - p}{\sqrt{\frac{(p)\cdot(1-p)}{n}}} \leq Z_2) = \\
& \mathbb{P}\left(-Z_2 \cdot \sqrt{\tfrac{(p)\cdot(1-p)}{n}} \leq Z - p \leq Z_2 \cdot \sqrt{\tfrac{(p)\cdot(1-p)}{n}}\right) = \\
& \mathbb{P}\left(-Z_2 \cdot \sqrt{\tfrac{(p)\cdot(1-p)}{n}} - Z \leq -p \leq Z_2 \cdot \sqrt{\tfrac{(p)\cdot(1-p)}{n}} - Z\right) = \\
& \mathbb{P}\left(Z_2 \cdot \sqrt{\tfrac{(p)\cdot(1-p)}{n}} + Z \geq p \geq -Z_2 \cdot \sqrt{\tfrac{(p)\cdot(1-p)}{n}} + Z\right) = \\
& \mathbb{P}\left(-Z_2 \cdot \sqrt{\tfrac{(p)\cdot(1-p)}{n}} + Z \leq p \leq Z_2 \cdot \sqrt{\tfrac{(p)\cdot(1-p)}{n}} + Z\right)
\end{aligned}$$

This confidence interval depends on p which is the parameter we are estimating. We can replace p with the estimator Z [96] in the left and right hand side and get $p \in \left[Z - Z_2 \cdot \sqrt{\frac{(Z)\cdot(1-Z)}{n}},\ Z + Z_2 \cdot \sqrt{\frac{(Z)\cdot(1-Z)}{n}}\right]$ with confidence α.

Precomputing Sample Size

The above method of constructing the confidence interval relies on first obtaining the samples, then specify the confidence level and finally based on these construct the actual interval. It is more satisfying for a user to specify a width, $2 \cdot \delta$, of the confidence interval and the required confidence level, α, and let a tool determine the number, n, of samples. A nice result [78] (known as the Chernoff-Hoeffding bound) states that if $X_1, X_2, \ldots, X_n$ are identically distributed random variables with mean μ and $Y = \frac{\sum_{i=1}^{n} X_i}{n}$ then

$$\mathbb{P}(|Y - \mu| \geq \delta) \leq 2 \cdot e^{-2 \cdot n \cdot \delta^2} \Rightarrow \mathbb{P}(Y \in [\mu - \delta, \mu + \delta]) \geq 1 - 2 \cdot e^{-2 \cdot n \cdot \delta^2},$$

for some δ. If we want to ensure an interval with at least confidence α we should ensure that $\mathbb{P}(Y \in [\mu - \delta, \mu + \delta]) \geq 1 - 2 \cdot e^{-2 \cdot n \cdot \delta^2} \geq \alpha$. Now solving the inequation for n yields

$$\begin{aligned} & 1 - 2 \cdot e^{-2 \cdot n \cdot \delta^2} && \geq && \alpha \\ \Rightarrow & 1 - \alpha && \geq && 2 \cdot e^{-2 \cdot n \cdot \delta^2} \\ \Rightarrow & \frac{1 - \alpha}{2} && \geq && e^{-2 \cdot n \cdot \delta^2} \\ \Rightarrow & \ln\left(\frac{1 - \alpha}{2}\right) && \geq && -2 \cdot n \cdot \delta^2 \\ \Rightarrow & \frac{\ln(\frac{1-\alpha}{2})}{-2 \cdot \delta^2} && \leq && n \end{aligned} \quad ,$$

and thus we should generate at least $\frac{\ln(\frac{1-\alpha}{2})}{-2 \cdot \delta^2}$ samples.

2.4 Hypothesis Testing

In certain situations we are not interested in the exact probability of a desired property but wish to assert it is greater than a threshold value θ. Assume the real, unknown, probability is p and let $X_1, \ldots, X_n$ be random variables that are 1 with probability p and 0 with probability $1 - p$. Let $Y = \sum_{i=1}^{n} X_i$ then $Y \sim \texttt{Binom}_{n,p}$ and $\texttt{E}[Y] = n \cdot p$. If $p \geq \theta$, we expect $Y \geq n \cdot \theta$. If this is the result, we *accept* the hypothesis that $p \geq \theta$. On the other hand, if $Y < n \cdot \theta$ there is a possibility that the hypothesis is true and should only be *rejected* in the case Y is much smaller than $\theta \cdot n$.

The intuition is that a hypothesis should only be rejected if the result is unlikely given the hypothesis is true. In advance, one should decide how large the probability of rejecting a true hypothesis is allowed to be. This value is called the level of *significance* and denoted α. Having decided on α, we find the largest value c where

$$\mathbb{P}(X \leq c | p \geq \theta) \leq \alpha.$$

The hypothesis is rejected if the result is less than c. The area from 0 to c is called the *critical region*.

The significance bounds the probability of rejecting a true hypothesis but gives no way of bounding the probability of accepting a false hypothesis. The *power* of a test, denoted β, is the probability of acceptance while the hypothesis is wrong i.e. a test has power β if $\mathbb{P}(X > c | p < \theta) < \beta$. In the above setting this is uncontrollable. If we want to bound both α and β we need to introduce an *indifference region* of width $2 \cdot \delta$. Instead of testing whether $p \geq \theta$ we test if $p \geq \theta + \delta$ and choose c such that $\mathbb{P}(X > c | p < \theta - \delta) < \beta$ and $\mathbb{P}(X \leq c | p \geq \theta + \delta) \leq \alpha$. An example of an algorithm for finding fixed n and c, called a *single sampling plan*, for user specified α, β and δ is provided by Younes [123].

Sequential Probability Ratio Test

Assume we have concluded that 1000 samples are needed and that c should be 100 for having a significance level of α, power β and an indifference region of width $2 \cdot \delta$. If the first 101 samples satisfy the property then there is no need to generate the remaining 899 runs. Intuitively, we can do with a lot less samples than precalculated if we keep track of the successes during generation of samples.

By generalising this idea, Wald [118] introduced the sequential probability ratio test (SPRT) which is a hypothesis testing algorithm that can draw a conclusion with fewer samples than the precomputed n and c. Assume we want to test that $p \geq \theta$ and use an indifference region of width $2 \cdot \delta$ and let $p_0 = \theta + \delta$ and $p_1 = \theta - \delta$. The SPRT algorithm then iteratively generates a sample and validate if the new sample has the desired property. Let X count the number of samples with the property at iteration i, s be the actual number and let p be the real probability. Now, the quantity

$$r = \frac{\mathbb{P}(X = s | p = p_1)}{\mathbb{P}(X = s | p = p_0)} = \frac{\binom{i}{s}({p_1}^s) \cdot (1 - p_1)^{i-s}}{\binom{i}{s}({p_0}^s) \cdot (1 - p_0)^{i-s}} = \frac{({p_1}^s) \cdot (1 - p_1)^{i-s}}{({p_0}^s) \cdot (1 - p_0)^{i-s}}$$

is calculated and compared to two boundary conditions A and B defined by the algorithm. If $r \leq B$ then the hypothesis is accepted and if $r \geq A$ then the hypothesis is rejected. In any other case a new iteration is made. Finding A and B boundaries such that the test has significance α and power β is non-trivial but using $A = \frac{1-\beta}{\alpha}$ and $B = \frac{\beta}{1-\alpha}$ gives actual significance α' and power β' which guarantees $\alpha' + \beta' \leq \alpha + \beta$. In practice $\alpha' \leq \alpha$ and $\beta' \leq \beta$ holds most of the times [123].

Modelling Formalisms

3

This chapter gives an overview of various modelling formalisms. First it discusses how the discrete behaviour of and interaction between components is modelled. Then this is generalised to timed systems and the higher level formalism timed automata is introduced. After this, probabilistic choices are modelled using discrete time Markov chains and then it is shown how time is added with the continuous time Markov chains. Afterwards, a stochastic semantics for compositions of timed automata is given along with some syntactical restrictions to the syntax of timed automata and this is eventually generalised to stochastic hybrid automata.

3.1 Labelled Transition System

The most elementary view we can have on a system is that it consists of *states* and *transitions* between states. Transitions are *labelled* to tell observers what the system does and to synchronise multiple labelled transition systems (LTSs) [82].

Example 2. Figure 3.1 depicts a LTS for a vending machine. In the initial state, indicated by the incoming *start* arrow, the machine waits for a coin from a user. This is modelled by an input synchronisation `coin`?. Afterwards it awaits a selection from the user by waiting for an input synchronisation `coff`? or `tea`?. Finally, it provides coffee or tea through output synchronisations `gCoff`! or `gTea`!.

Let `Chan` be a set of channels over which components synchronise their behaviour, then the possible output actions over these channels are $\texttt{Chan}_o = \{\texttt{a}! \mid \texttt{a} \in \texttt{Chan}\}$ and equivalently the possible input actions $\texttt{Chan}_i = \{\texttt{a}? \mid \texttt{a} \in \texttt{Chan}\}$. In general we write `a`! for outputting on channel `a` and `a`? for receiving on `a`. For the remainder we abstract from the channels and merely specify the output and input actions of components. However, we adhere to the convention that the postfix ? is for input actions, ! is for output actions and that an output action `a`! has a single corresponding input action `a`? and vice versa. For a set of

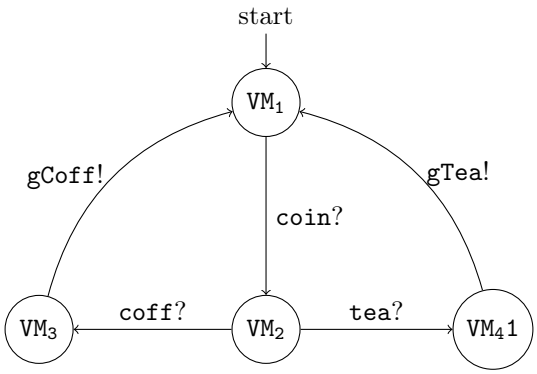


Figure 3.1: Coffee machine example

output actions $\Sigma_{\mathtt{o}}$ we denote by $\overline{\Sigma_{\mathtt{o}}}$ the set of corresponding input actions i.e. $\overline{\Sigma_{\mathtt{o}}} = \{\mathtt{a}? \mid \mathtt{a}! \in \Sigma_{\mathtt{o}}\}$.

Let S be a set of states, $\Sigma_{\mathtt{o}}$ a set of output actions and $\Sigma_{\mathtt{i}}$ a set of input actions. A transition relation $\rightarrow \subseteq S \times (\Sigma_{\mathtt{o}} \cup \Sigma_{\mathtt{i}}) \times S$ describes what actions can be performed to move between states. As per tradition we write $s \xrightarrow{a} s'$ if $(s, a, s') \in \rightarrow$. A transition relation, $\rightarrow$, is *input-enabled* with respect to a set of input actions $\Sigma_{\mathtt{i}}$ if for all $s \in S$ there exists $\mathtt{a}? \in \Sigma_{\mathtt{i}}$ and $s' \in S$ such that $s \xrightarrow{\mathtt{a}?} s'$ and it is *action-deterministic* with respect to a set of actions Σ if for any $a \in \Sigma$ whenever $s \xrightarrow{a} s'$ and $s \xrightarrow{a} s''$ then $s' = s''$.

Definition 8 (Labelled transition system)**.** An LTS is a tuple $(S, s_0, \Sigma_{\mathtt{o}}, \Sigma_{\mathtt{i}}, \mathtt{AP}, \mathtt{Pm}, \rightarrow)$ where

- S is a set of states,
- $s_0 \in S$ is the initial state,
- $\Sigma_{\mathtt{o}}$ is a set of output actions,
- $\Sigma_{\mathtt{i}}$ is a set of input actions,
- $\mathtt{AP}$ is a set of atomic propositions,
- $\mathtt{Pm} : S \rightarrow 2^{\mathtt{AP}}$ gives the propositions that are true in a state and
- $\rightarrow \subseteq S \times (\Sigma_{\mathtt{o}} \cup \Sigma_{\mathtt{i}}) \times S$ is a transition relation which is
  1. input-enabled with respect to $\Sigma_{\mathtt{i}}$ and
  2. action-deterministic with respect to $\Sigma_{\mathtt{i}} \cup \Sigma_{\mathtt{o}}$.

The required input-enabledness is easily guaranteed by adding self-loop on all states - these are not shown in pictures of LTSs. The possible executions of a

$$\frac{\exists i,\ s_i \xrightarrow{\mathtt{a}!} s_i' \qquad \forall j \neq i\ s_j \xrightarrow{\mathtt{a}?} s_j'}{(s_1, \ldots, s_n) \xrightarrow{\mathtt{a}!} (s_1', \ldots, s_n')}$$

Figure 3.2: Inference rule for parallel composition of n LTSs.

LTS $\mathcal{K} = (S, s_0, \Sigma_\mathtt{o}, \Sigma_\mathtt{i}, \mathtt{AP}, \mathtt{Pm}, \rightarrow)$ is a sequence of states and output actions $s_0 \mathtt{a_0}! s_1 \mathtt{a_1}! \ldots$, such that for all i $s_i \xrightarrow{\mathtt{a_i}!} s_{i+1}$.

Network of Labelled Transition Systems

Let $\Sigma_\mathtt{o}$ be a set of output actions partitioned into disjoint sets ${\Sigma_\mathtt{o}}^1, {\Sigma_\mathtt{o}}^2, \ldots, {\Sigma_\mathtt{o}}^n$ and let $\mathcal{K}_1, \ldots, \mathcal{K}_n$ where $\mathcal{K}_i = (S^i, {s_0}^i, {\Sigma_\mathtt{o}}^i, \overline{\Sigma_\mathtt{o}}, \mathtt{AP_i}, \mathtt{Pm}_i, \rightarrow^i)$ be LTSs. Then $\mathcal{K}_1, \ldots, \mathcal{K}_n$ can be composed into a network $\mathcal{J} = \mathcal{K}_1 | \ldots | \mathcal{K}_n$ which is a new LTS $\mathcal{K}_\mathcal{J} = (\mathbf{S}_\mathcal{J}, {\mathbf{s}_0}^\mathcal{J}, \Sigma_\mathtt{o}, \emptyset, \mathtt{AP}_\mathcal{J}, \mathtt{Pm}_\mathcal{J}, \rightarrow^\mathcal{J})$ with

- $\mathbf{S}_\mathcal{J} = S^1 \times \cdots \times S^n$,
- ${\mathbf{s}_0}^\mathcal{J} = ({s_0}^1, \ldots, {s_0}^n)$,
- $\mathtt{AP}_\mathcal{J} = \bigcup_{i=1}^n \mathtt{AP_i}$ and
- $\mathtt{Pm}_\mathcal{J}(s_1, \ldots, s_n) = \bigcup_{i=1}^n \mathtt{Pm}_i(s_i)$.

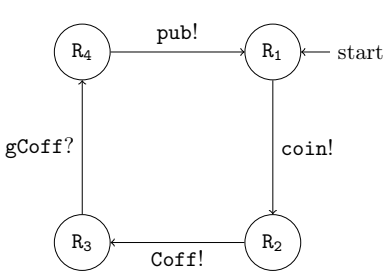
The transition relation $\rightarrow^\mathcal{J}$ is defined by the rule in Figure 3.2 and says that if one component wishes to make an output then all others must respond to that output with the corresponding input - we thus use *broadcast* as opposed to *handshake* communication. Hand-shake synchronisation has inter-dependencies between components because an output-action of one component can be blocked by no-one offering the corresponding input synchronisation. Our assumption of input-enabledness would guarantee this would never happen, but the receiver would be selected non-deterministically. For the later development of a stochastic semantics, we wish to only have non-determinism in selecting the output-actions thus the use of broadcast communication.

Example 3. Consider composing the LTS in Figure 3.3a with Figure 3.1 then we get the LTS depicted in Figure 3.3b. A possible execution of this composition is the sequence

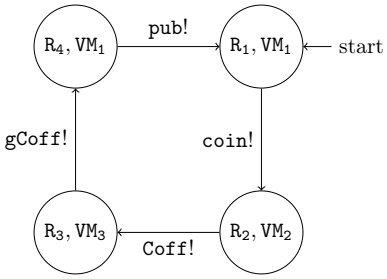
$$\mathtt{R}_1, \mathtt{M}_1\mathtt{coin!R}_2, \mathtt{M}_2\mathtt{coff!R}_3, \mathtt{M}_3 \ldots$$

3.2 Timed Labelled Transition System

Previously we mentioned the need for modelling timed systems. Timed labelled transition system (TLTS) is an extension of LTS that add delay transitions to the system.



(a) Researcher.



(b) Composed researcher and Vending machine. States that are unreacbable from the intial state $\mathtt{R}_1, \mathtt{VM}_1$ are omitted from the picture.

Figure 3.3: Composition of LTSs.

Definition 9 (Timed labelled transition system)**.** A TLTS is a tuple $(S, s_0, \Sigma_\mathtt{o}, \Sigma_\mathtt{i}, \mathtt{AP}, \mathtt{Pm}, \rightarrow)$ where

- S is a set of states,
- $s_0 \in S$ is the initial state,
- $\Sigma_\mathtt{o}$ is a set of output actions,
- $\Sigma_\mathtt{i}$ is a set of input actions,
- $\mathtt{AP}$ is a finite set of propositions,
- $\mathtt{Pm} : S \to 2^{\mathtt{AP}}$ maps a state to a set of propositions that are true in that state
- $\rightarrow \subseteq S \times (\Sigma_\mathtt{o} \cup \Sigma_\mathtt{i} \cup \mathbb{R}_{\geq 0}) \times S$ is a transition relation which is
  1. input-enabled with respect to $\Sigma_\mathtt{i}$ and
  2. action-deterministic with respect to $\Sigma_\mathtt{i} \cup \Sigma_\mathtt{o} \cup \mathbb{R}_{\geq 0}$.

If $(S, s_0, \Sigma_\mathtt{o}, \Sigma_\mathtt{i}, \mathtt{AP}, \mathtt{Pm}, \rightarrow)$ is a TLTS, $s \in S$ is a state, $d \in \mathbb{R}_{\geq 0}$ is a number and $s \xrightarrow{d} s'$ then one would naturally expect to be able to delay any $d' < d$ and from the resulting state be able to delay $d - d'$ to reach s'. We impose this requirement to any TLTS and say that time is additive. Another requirement we impose on the delay transition relation is that a zero-delay cannot change the state and we assume that time delays are deterministic. The requirements are formalised as follows [2]:

Time Additive If $s \xrightarrow{d} s'$ and $0 \leq d' \leq d$ then $s \xrightarrow{d'} s'' \xrightarrow{d-d'} s'$ for some s'',

No delay $s \xrightarrow{0} s$ and

Time deterministic If $s \xrightarrow{d} s'$ and $s \xrightarrow{d} s''$ then $s' = s''$.

Definition 10 (Timed Run)**.** An infinite timed run over a TLTS $\mathcal{K} = (S, s_0, \Sigma_{\mathtt{o}}, \Sigma_{\mathtt{i}}, \mathtt{AP}, \mathtt{Pm}, \rightarrow)$ is a sequence $s_0 d_0 \mathtt{a_0}! \, s_1 d_1 \mathtt{a_1}! \, s_2 d_2 \mathtt{a_2}! \ldots$ where for all i there exists s'_i such that $s_i \xrightarrow{d_i} s'_i \xrightarrow{\mathtt{a_i}!} s_{i+1}$.

We denote by $\Omega(\mathcal{K})$ all timed runs of $\mathcal{K}$ and if $\pi = s_0 d_0 \mathtt{a_0}! s_1 d_1 \mathtt{a_1}! \cdots \in \Omega(\mathcal{K})$ then we let $\pi^{:n} = s_0 d_0 \mathtt{a_0}! s_1 d_1 \mathtt{a_1}! \ldots d_{n-1} \mathtt{a_{n-1}}! s_n$ be its finite prefix, $\pi^{n:} = s_n d_n \mathtt{a_n}! s_{n+1} d_{n+1} \mathtt{a_{n+1}}! \ldots$ be its finite postfix and let $\pi^{:i:} = (s_i, d_i, \mathtt{i}!)$.

Definition 11 (Timed Propositional Run)**.** An infinite timed propositional run over the TLTS $\mathcal{K} = (S, s_0, \Sigma_{\mathtt{o}}, \Sigma_{\mathtt{i}}, \mathtt{AP}, \mathtt{Pm}, \rightarrow)$ is a sequence $\mathtt{P}_0 d_0 \, \mathtt{P}_1 d_1 \ldots$ where there exists $s_0 d_0 \mathtt{a_0}! \, s_1 d_1 \mathtt{a_1}! \cdots \in \Omega(\mathcal{K})$ such that for all i $\mathtt{P}_i = \mathtt{Pm}(s_i)$.

The set of all propostional runs over $\mathcal{K}$ is denoted $\Omega^{\mathtt{AP}}(\mathcal{K})$.

Network of Timed Labelled Transition Systems

Regarding a composition of TLTSs the transition semantics is the same as for LTSs with one added rule that for the composition to do a delay all TLTSs should do the delay i.e.

$$\frac{\forall j, s_j \xrightarrow{d} s'_j \qquad d \in \mathbb{R}_{\geq 0}}{(s_1, \ldots, s_n) \xrightarrow{d} (s'_1, \ldots, s'_n)}$$

The delay transition blocks the composition with respect to time if one TLTS is unable to do a delay $d > 0$ and no TLTS can perform an output action. This is counter-intuitive as a real system cannot stop time in this way hence we assume that for any state s of any TLTS either (1) for all $d \in \mathbb{R}_{\geq 0}$ there exists a s' s.t. $s \xrightarrow{d} s'$ or (2) there exists a $d \in \mathbb{R}_{\geq 0}$, an output action $\mathtt{a}!$ and states s' and s'' such that $s \xrightarrow{d} s' \xrightarrow{\mathtt{a}!} s''$. That is, either the system can do any delay or it can do some delay and afterwards perform an output. We call this property *time-lock-freedom*.

3.3 Timed Automata

Timed labelled transition systems are useful for giving semantics to timed systems but providing concise descriptions with TLTSs is tedious/impossible. What is needed is a finite syntax with semantics given by a TLTS. A timed automaton (TA) [6, 7] is a standard finite state machine equipped with real valued counters (called *clocks*) that keep track of the time flow in the system.

If $\mathtt{X}$ is a set of clocks then an upper (lower) bound over $\mathtt{x}$ is an element $\mathtt{x} \bowtie n$ where $\mathtt{x} \in \mathtt{X}$, $n \in \mathbb{N}$ and $\bowtie \in \{<, \leq\}$ ($\bowtie \in \{\geq, >\}$). The set of upper (lower) bounds over $\mathtt{X}$, $\mathtt{B}^{\leq}(\mathtt{X})$ ($\mathtt{B}^{\geq}(\mathtt{X})$), is the set of finite conjunctions of upper (lower) bounds and $\mathtt{B}(\mathtt{X})$ is the set of finite conjunctions of upper and lower bounds. Let $\mathtt{tt}$ denote the empty conjunction.

Example 4. In Figure 3.4 is shown a model of a smart lamp with three settings: Off, On and Bright. In the Off setting the lamp is turned to On by touching it. If the lamp is touched a second time within 5 seconds of being turned On it is turned Bright. After the five seconds in On touching the lamp will turn it Off. Touching the lamp while it is Bright always turn it Off.

The three settings are indicated by the three locations (the circles) of the model and the initial setting of Off is indicated by the double concentric circle. The switching between locations is governed by the edges and on edges are annotated synchronisations. The model has one clock x which is set to zero when entering On and the edges to Bright and Off are *guarded* by the expressions x>5 and x<=5 to ensure only one is enabled when the synchronisation touch? is given.

Definition 12 (Timed automaton)**.** A timed automaton is a tuple $(\mathtt{L}, \mathtt{l}_0, \Sigma_\mathtt{o}, \Sigma_\mathtt{i}, \mathtt{X}, \mathtt{E}, \mathtt{I})$ where

- $\mathtt{L}$ is a finite set of control locations,
- $\mathtt{l}_0 \in \mathtt{L}$ is the initial location,
- $\Sigma_\mathtt{o}$ is a set of output actions,
- $\Sigma_\mathtt{i}$ is a set of input actions,
- $\mathtt{X}$ is a set of clocks,
- $\mathtt{E} \subseteq \mathtt{L} \times \mathtt{B}(\mathtt{X}) \times (\Sigma_\mathtt{o} \cup \Sigma_\mathtt{i}) \times 2^\mathtt{X} \times \mathtt{L}$ is a set of edges between locations and
- $\mathtt{I} : \mathtt{L} \to \mathtt{B}(\mathtt{X})$ assigns an invariant to locations.

If $\mathtt{X}$ is a set of clocks, then a clock valuation is a function $\mathtt{v} : \mathtt{X} \to \mathbb{R}_{\geq 0}$ assigning a real value to all clocks. We let $\mathtt{V}(\mathtt{X})$ be the set of all valuations over $\mathtt{X}$. Let $g = (\mathtt{x} \bowtie n) \in \mathtt{B}(\mathtt{x})$ and let $\mathtt{v} \in \mathtt{V}(\mathtt{X})$ then we write $\mathtt{v} \vDash g$ iff $\mathtt{v}(\mathtt{x}) \bowtie n$ and

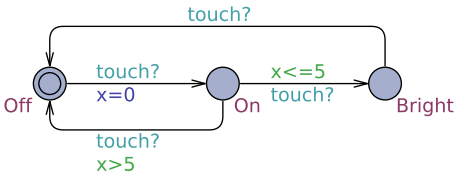


Figure 3.4: A smart lamp

say that $\mathtt{v}$ satisfies g. This is straightforwardly generalised to a conjunction of bounds.

For defining the semantics of TAs a few operations on valuations is needed: the *delay* operator increases the value of all clocks a given number of time units and the *reset* operator sets a set of clocks to zero. Let $\mathtt{v} \in \mathtt{V}(\mathtt{X})$, $d \in \mathbb{R}_{\geq 0}$ and

$\mathtt{Y} \subseteq \mathtt{X}$ then

$$(\mathtt{v}+d)(\mathtt{x}) = \mathtt{v}(\mathtt{x}) + d \quad \text{and} \quad \mathtt{v}[\mathtt{Y}=0](\mathtt{x}) = \begin{cases} 0 & \text{if } \mathtt{x} \in \mathtt{Y} \\ \mathtt{v}(\mathtt{x}) & \text{Otherwise.} \end{cases}$$

With these operations at our disposal we define the semantics of TA $S = (\mathtt{L}, \mathtt{l}_0, \Sigma_\mathtt{o}, \Sigma_\mathtt{i}, \mathtt{X}, \mathtt{E}, \mathtt{I})$ as the TLTS $\mathcal{K}_S = (\mathtt{L} \times \mathtt{V}(\mathtt{X}), (\mathtt{l}_0, \vec{0}), \Sigma_\mathtt{o}, \Sigma_\mathtt{i}, \mathtt{L}, \mathtt{Pm}_S, \rightarrow)$ where $\vec{0}(\mathtt{x}) = 0$ for any $\mathtt{x} \in \mathtt{X}$, $\mathtt{Pm}_S(\mathtt{l}, \mathtt{v}) = \mathtt{l}$ and $\rightarrow$ is defined as follows:

- $(\mathtt{l}, \mathtt{v}) \xrightarrow{d} (\mathtt{l}, (\mathtt{v}+d))$ if $d \in \mathbb{R}_{\geq 0}$ and $(\mathtt{v}+d) \vDash \mathtt{I}(\mathtt{l})$ and
- $(\mathtt{l}, \mathtt{v}) \xrightarrow{a} (\mathtt{l}', \mathtt{v}')$ if there exists $(\mathtt{l}, g, a, r, \mathtt{l}') \in \mathtt{E}$ such that $\mathtt{v} \vDash g$, $\mathtt{v}' = \mathtt{v}[r=0]$ and $\mathtt{v}' \vDash \mathtt{I}(\mathtt{l}')$.

For consistency we require that $\vec{0} \vDash \mathtt{I}(\mathtt{l}_0)$.

Remark 2. In this mapping we use the locations of the TA as propositions. Obviously, we can generalise this to more general set of propositions.
For instance, the extended formalism of TA used by UPPAAL has integer variables and hence propositions could be the result of comparison between variables or even the clock bounds that are true in a state.

Let $S_1, S_2, \ldots, S_n$ be TAs and $\mathcal{K}_1, \ldots, \mathcal{K}_n$ be their underlying TLTSs. Syntactically we compose the TAs with the operator $\|$. The semantics of a composition of TAs, $S_1 \| S_2 \| \ldots, \| S_n$, is the TLTS $\mathcal{J}_{S_1 \| S_2 \| \ldots, \| S_n} = \mathcal{K}_1 | \ldots | \mathcal{K}_n$.

Example 5. Returning to our smart lamp we show the underlying TLTS in Figure 3.5. In this drawing, the horizontal axis indicate the value of the clock x and the vertical position the location of the TA. Only a subset of the transitions is shown: solid lines correspond to discrete touch? synchronisations whereas the dashed lines correspond to delay transitions.

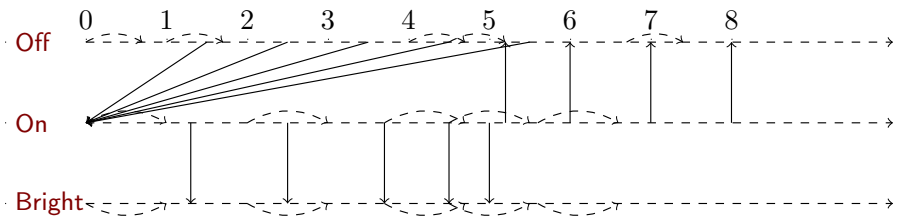


Figure 3.5: Timed labelled transition system for the smart lamp in Figure 3.4

3.4 Discrete Time Markov Chain

A discrete time Markov chain (DTMC) chain [12] consists of a set of states and a set of transitions between states. To each state is associated a probability distribution over the transitions i.e. the successor state is selected probabilistically. The important property of a DTMC is that the probabilities in the current state is independent from whatever happened in the past. As an example, consider the DTMC modelling a server in Figure 3.6. Initially, the server is in the `Idle` state and may accept a connection with probability 0.10 or stay idle with probability 0.90. Accepting a connection moves the server into `Work` from which it may complete the task with probability 0.99 or break down with probability 0.01. From here, the server may either be repaired and start idleing or require longer maintenance and stay in `Repair`.

Definition 13 (Discrete time Markov chain)**.** A discrete time Markov chain is a tuple $(S, s_0, \gamma, \mathtt{AP}, \mathtt{Pm})$ where

- S is a countable set of states,
- $s_0 \in S$ is the initial state,
- $\gamma : S \times S \to [0, 1]$ assigns probabilities to successor states with the requirement $\sum_{s' \in S} \gamma(s, s') = 1$ for all $s \in S$,
- $\mathtt{AP}$ is a finite set of propositions and
- $\mathtt{Pm} : S \to 2^{\mathtt{AP}}$ maps states to propositions.

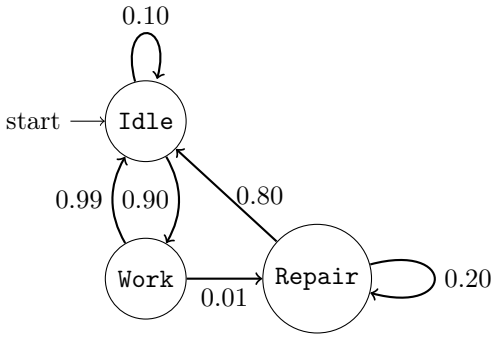


Figure 3.6: A discrete time Markov chain (DTMC)

An infinite *run* of a DTMC $\mathtt{M} = (S, s_0, \gamma, \mathtt{AP}, \mathtt{Pm})$ is an infinite sequence $\pi = s_0 s_1 s_2 \ldots$ where for all i, $\gamma(s_i, s_{i+1}) \neq 0$. Following notation as for LTSs we denote by $\Omega(\mathtt{M})$ all runs generated by $\mathtt{M}$ and let $\pi^{:n} = s_0 s_1 \ldots s_{n-1} s_n$. Let $\omega = s_0 s_1 \ldots s_n$ be a finite sequence of states, then the collection of paths of DTMC $\mathtt{M}$ with this prefix, $\mathcal{C}_{\mathtt{M}}(\omega) = \{\pi \in \Omega(\mathtt{M}) \,|\, \omega = \pi^{:n}\}$, is called the cylinder

of ω and a measure of the cylinder is $\mathbb{P}_{\mathtt{M}}(\omega) = \prod_{i=0}^{n-1} \gamma(s_i, s_{i+1})$, where the product signifies that transitions between states are selected independently from each other. A σ-algebra over the DTMC is constructed as the smallest σ-algebra containing all cylinder sets $\mathcal{C}_{\mathtt{M}}(\omega)$ where ω ranges over all possible finite prefixes [12]. As for LTSs we can also consider the set of propositional runs of a DTMC $(S, s_0, \gamma, \mathtt{AP}, \mathtt{Pm})$ being $\Omega^{\mathtt{AP}}(\mathtt{M}) = \{\mathtt{Pm}(s_0), \mathtt{Pm}(s_1) \ldots \mid s_0, s_1, \cdots \in \Omega(\mathtt{M})\}$.

Example 6. Consider the DTMC in Figure 3.6 and the probability that the server is idling for two steps then is working and afterwards is idling again. We are thus interested in the probability of getting a run with the prefix

$$\mathtt{Idle}, \mathtt{Idle}, \mathtt{Work}, \mathtt{Idle}.$$

If $(S, s_0, \gamma, \mathtt{AP}, \mathtt{Pm})$ is the DTMC then the probability is clearly

$$\gamma(\mathtt{Idle},\mathtt{Idle}) \cdot \gamma(\mathtt{Idle}, \mathtt{Work}) \cdot \gamma(\mathtt{Work}, \mathtt{Idle}) = 0.1 \cdot 0.90 \cdot 0.99 = 0.0891$$

3.5 Continuous Time Markov Chain

A continuous time Markov chain (CTMC) [96] extends the DTMC model by incorporating real time information into the states. In particular, each state is associated with an exponential distribution that gives the time the system will stay in that state.

Definition 14 (Continuous time Markov chain)**.** A CTMC is a tuple $(S, s_0, \gamma, \mathtt{R}, \mathtt{AP}, \mathtt{Pm})$ where

- S is a countable set of states,
- $s_0 \in S$ is the initial state,
- $\gamma : S \times S \to [0, 1]$ assigns probabilities to successor states with the requirement $\sum_{s' \in S} \gamma(s, s') = 1$ for all $s \in S$,
- $\mathtt{R} : S \to \mathbb{R}_{>0}$ is a function giving the rate parameter to an exponential distribution,
- $\mathtt{AP}$ is a set of propositions and
- $\mathtt{Pm} : S \to 2^{\mathtt{AP}}$ maps states to propositions

An infinite run of a CTMC $(S, s_0, \gamma, \mathtt{R}, \mathtt{AP}, \mathtt{Pm})$ is an infinite sequence $\pi = s_0 d_0 s_1 d_1 \ldots$ where for all i, $s_i \in S$, $d_i \in \mathbb{R}_{\geq 0}$ and $\gamma(s_i, s_{i+1}) \neq 0$. The d_i is called the *sojourn* time of s_i and is the time that the system is waiting in state s_i. As for TLTSs we let $\Omega(\mathtt{M})$ and $\Omega^{\mathtt{AP}}(\mathtt{M})$ denote all timed runs and timed

propositional runs respectively over the CTMC $\mathtt{M}$.
The stochastic semantics is given in the same way as for DTMCs but the cylinder construction also take the sojourn times into account. A *path* over a CTMC $\mathtt{M} = (S, s_0, \gamma, \mathtt{R}, \mathtt{AP}, \mathtt{Pm})$ is a finite sequence $\omega = s_0 I_0 s_1 I_1 \dots s_n$ where for all $i < n$, $s_i \in S$, $\gamma(s_i, s_{i+1}) \neq 0$ and I_i is an interval of $\mathbb{R}_{\geq 0}$. The cylinder of runs of ω is given by

$$\mathcal{C}_{\mathtt{M}}(\omega) = \{\pi \in \Omega(\mathtt{M}) \mid \forall i \leq n, \pi^{:i:} = (s_i, d_i) \text{ and } d_i \in I_i\}$$

and the probability is $\mathbb{P}_{\mathtt{M}}(\omega) = \prod_{i=0}^{n-1} \gamma(s_i, s_{i+1}) \cdot \int_{I_i} \mathtt{Exp}_{\mathtt{R}(s_i)}(t)\,\mathrm{d}t$.

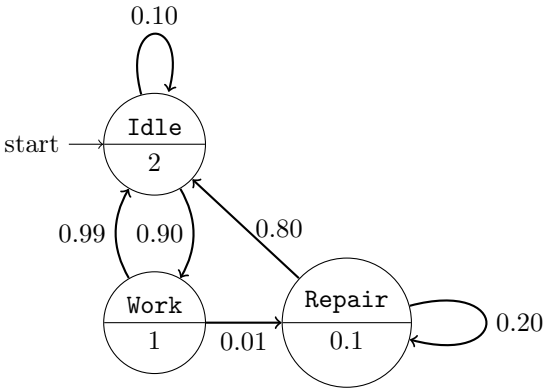


Figure 3.7: A CTMC

Example 7. In Figure 3.7 is depicted the DTMC as in the DTMC section but the states are annotated with rates hence creating a CTMC. For this CTMC we can calculate the probability that it will be idle two steps in a row, then be working and return to idle. We have previously seen in the DTMC case that this probability was 0.0891. If we add a time restriction, insisting the final idle should be reached before 2 time units, the set of runs we are interested in is captured by the set

$$\{\mathtt{Idle}\, d_0\, \mathtt{Idle}\, d_1\, \mathtt{Work}\, d_2\, \mathtt{Idle} \mid d_0 + d_1 + d_2 \leq 2\}.$$

The probability of these may be calculated by the integral[a]:

$$\int_0^2 \mathtt{Exp}_2(d_0) \cdot 0.1 \int_0^{2-d_1} \mathtt{Exp}_2(d_1) \cdot 0.90 \cdot \int_0^{2-d_0-d_1} \mathtt{Exp}_1(d_2) \cdot 0.99\,\mathrm{d}\dots =$$

$$0.0891 \cdot \int_0^2 \mathtt{Exp}_2(d_0) \cdot \int_0^{2-d_1} \mathtt{Exp}_2(d_1) \cdot \int_0^{2-d_0-d_1} \mathtt{Exp}_1(d_2)\,\mathrm{d}\dots \approx 0.052$$

[a] for conciseness the $\mathrm{d}d_1\,\mathrm{d}d_2\,\mathrm{d}d_3$ are replaced by $\mathrm{d}\dots$

Transition Rates Instead of assigning a probability distribution over the sojourn time and probabilistic chocies for the actual state transition, it is sometimes[1] more natural that each transition has an associated exponential distribution governing the time that transition should be taken. The actual state transition is governed by a race among the transitions.

Consider a system is located in state s_0 and may transit to $s_1, s_2, \ldots, s_n$ with rates $r_1, r_2, \ldots, r_n$ - we will now see that we can represent this by a CTMC. Let us consider the probability that the system transits to s_1 before time t, denoted $\mathbb{P}(\leq t, s_1|s_0)$.

$$\begin{aligned}
\mathbb{P}(\leq t, s_1|s_0) &= \int_0^t \mathtt{Exp}_{r_1}(x) \cdot \prod_{i=2}^{n} \int_x^\infty \mathtt{Exp}_{r_i}(\tau)\,\mathrm{d}\tau\,\mathrm{d}x \\
&= \int_0^t r_1 \cdot e^{-r_1 \cdot x} \cdot \prod_{i=2}^{n} e^{-r_i \cdot x}\,\mathrm{d}x \\
&= \int_0^t r_1 \cdot e^{-(\sum_{i=1}^{n} r_i)\cdot x}\,\mathrm{d}x \\
&= r_1 \cdot \left[\frac{-1}{(\sum_{i=1}^{n} r_i)} e^{-(\sum_{i=1}^{n} r_i)\cdot x}\right]_0^t \\
&= r_1 \left(\frac{-1}{(\sum_{i=1}^{n} r_i)} \cdot e^{-(\sum_{i=1}^{n} r_i)\cdot t} - \frac{-1}{(\sum_{i=1}^{n} r_i)}\right) \\
&= \frac{r_1}{(\sum_{i=1}^{n} r_i)} \cdot (1 - e^{-(\sum_{i=1}^{n} r_i)\cdot t})
\end{aligned}$$

Following equivalent reasoning

$$\mathbb{P}(\leq t, s_j|s_0) = \frac{r_j}{(\sum_{i=1}^{n} r_i)} \cdot (1 - e^{-(\sum_{i=1}^{n} r_i)\cdot t}) = \frac{r_j}{(\sum_{i=1}^{n} r_i)} \cdot \int_0^t \mathtt{Exp}_{\sum_{i=1}^{n} r_i}(\tau)\,\mathrm{d}\tau$$

for any $j \in \{1 \ldots n\}$ and thus we can "match" these probabilities by selecting a sojourn time from $\mathtt{Exp}_{\sum_{i=1}^{n} r_i}$ and afterwards probabilistically choose a succesor state s_j with probability $\frac{r_j}{\sum_{i=1}^{n} r_i}$. Thus exactly a CTMC.

3.6 Stochastic Semantics for Transitions Systems

In this section we develop the stochastic semantics for transition systems. In particular we present the stochastic semantics presented by David et al. [45]. Before this we try to relate the two existing stochastic/probabilistic semantics to transition systems. For an LTS with state space S and output actions Σ_o we can for each state s assume a pmf $\gamma_s : \Sigma_o \to [0, 1]$ that assigns probabilities to the individual output actions. This γ_s would correspond to the transition probabilities of a DTMC and in effect defining a DTMC. An LTS can thus

[1] For instance biochemical networks

be given a probabilistic semantics by assigning probabilities to output actions. In the case of real time systems modelled by a TLTS, a CTMC is not an adequate stochastic model since the CTMC model rely on choosing a delay in a given state and choose a discrete transition independent of the delay. A TLTS does not behave like this as the set of possible output-actions after a delay depends on the actual delay. Another problem is that CTMCs only select delays from exponential distributions which are unbounded whereas TLTS may define bounded delays.

Definition 15. A stochastic TLTS is a tuple $(S, s_0, \Sigma_\mathtt{o}, \Sigma_\mathtt{i}, \mathtt{AP}, \mathtt{Pm}, \rightarrow, \gamma, \mu)$ where

- $(S, s_0, \Sigma_\mathtt{o}, \Sigma_\mathtt{i}, \mathtt{AP}, \mathtt{Pm}, \rightarrow)$ is a TLTS,
- $\gamma : S \hookrightarrow \Sigma_\mathtt{o} \rightarrow \mathbb{R}_{\geq 0}$ gives a pmf for states that assigns probability mass to output actions with the requirements that (1) $\gamma(s)$ is defined if and only if there exists $\mathtt{a!} \in \Sigma_\mathtt{o}$ such that $s \xrightarrow{\mathtt{a!}}$ and (2) $\gamma(s)(\mathtt{a!}) \neq 0$ only if there exists s' s.t. $s \xrightarrow{\mathtt{a!}} s'$ and
- $\mu : S \rightarrow \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ gives a pdf for each state with the requirement that $\mu(s)(d) \neq 0$ only if there exists s' such that $s \xrightarrow{d} s'$ and $\gamma(s')$ is defined.

As a short-hand we write $\mu_s(d)$ instead of $\mu(s)(d)$ and $\gamma_s(\mathtt{a!})$ instead of $\gamma(s)(\mathtt{a!})$.

Stochastic Semantics for Networks

The semantics for a network is that each component chooses a delay d. The component with the smallest delay wins the race and performs an output action ($\mathtt{a!}$) after the delay. The entire network performs the delay d and the losing components perform the corresponding input action ($\mathtt{a?}$). Afterwards a new race commences.

As for DTMCs and CTMCs the σ-algebra is defined by a cylinder construction. Two different cylinder constructions can be made. The first [45] defines the cylinder by means of the actions performed by the system and the second [53] defines the cylinder by the propositions observed in the runs. Before defining the cylinders we introduce some short-hand notations: if $\mathbf{s}$ is a state of a network over output action $\Sigma_\mathtt{o}$ then $[\mathbf{s}]^x$ denotes the state $\mathbf{s}'$ for which $\mathbf{s} \xrightarrow{x} \mathbf{s}'$ where $x \in \mathbb{R}_{\geq 0} \cup \Sigma_\mathtt{o}$. For both delay and output actions $\mathbf{s}'$ is uniquely defined due to our assumptions about determinism and partitioning of output actions among the components. Another short-hand notation is indexing into the state vector: if $\mathbf{s} = (s_1, s_2, \ldots, s_n)$ then $\mathbf{s}^i = s_i$.

Action-Based Cylinders Let $\mathcal{J} = \mathcal{K}_1 | \mathcal{K}_2 | \ldots | \mathcal{K}_n$ be a network over the output actions $\Sigma_\mathtt{o}$ where for all $\mathcal{K}_i = (S^i, {s_0}^i, {\Sigma_\mathtt{o}}^i, \Sigma_\mathtt{i}, \mathtt{AP_i}, \mathtt{Pm}_i, \rightarrow^i, \gamma_i, \mu_i)$. An action-prefix over the network is an element of $\{\mathtt{a_1!}, \mathtt{a_2!}, \mathtt{a_3!}, \ldots, \mathtt{a_m!} \mid \forall i, \mathtt{a_i!} \in \Sigma_\mathtt{o}\}$. The

cylinder set for an action-prefix $\mathtt{a}_1!, \mathtt{a}_2!, \mathtt{a}_3!, \ldots, \mathtt{a}_\mathtt{m}!$ is the set defined as

$$\mathcal{C}^A_{\mathcal{J}}(\mathtt{a}_1!, \mathtt{a}_2!, \ldots, \mathtt{a}_\mathtt{m}!) = \{s_1 d_1 \mathtt{a}_1! s_2 d_2 \mathtt{a}_2! \ldots, \mathtt{a}_\mathtt{m}!, \cdots \in \Omega(\mathcal{J})\},$$

The probability of observing a run with the prefix $\mathtt{a}_1!, \mathtt{a}_2!, \ldots, \mathtt{a}_\mathtt{m}!$ from state $\mathbf{s}$ is recursively defined as

$$\mathbb{P}_{\mathbf{s}}(\mathtt{a}_1!, \mathtt{a}_2!, \ldots, \mathtt{a}_\mathtt{m}!) = \int_{t \geq 0} \mu_{\mathbf{s}^c}(t) \cdot \left(\prod_{i \neq c} \int_{\tau > t} \mu_{\mathbf{s}^i} \, \mathrm{d}\tau \right) \cdot \gamma_{([\mathbf{s}]^t)^i}(\mathtt{a}_1!) \cdot \mathbb{P}_{\left[[\mathbf{s}]^t\right]^{\mathtt{a}_1!}}(\mathtt{a}_2! \ldots, \mathtt{a}_\mathtt{m}!) \, \mathrm{d}t,$$

with base case $\mathbb{P}_{\mathbf{s}}(\epsilon) = 1$ and $\mathtt{a}_1! \in {\Sigma_\mathtt{o}}^c$.

On the outermost level we integrate over all possible delays the component responsible for performing the $\mathtt{a}_1!$ action is allowed to do. The other components independently choose a larger delay. Then the probability of performing the actual action is multiplied and finally, since the remaining steps are independent from the current one, the probability of seeing the remaining actions is multiplied.

Proposition-Based Cylinders A problem with defining the cylinder sets on the basis of actions is that most logics are defined on the basis of propositional runs thus the cylinders should contain the propositions. Furthermore, the action-based cylinders do not contain timing information which is important for some timed logics such as MTL [83].

For a network $\mathcal{J} = \mathcal{K}_1 | \mathcal{K}_2 | \ldots | \mathcal{K}_n$ where for all i $\mathtt{AP}_{\mathcal{K}_i}$ is the set of propositions for $\mathcal{K}_i$ we define a path to be a sequence $\omega = \mathtt{P}_1 I_1 \mathtt{P}_2 I_2 \ldots \mathtt{P}_m$, where for all i, $\mathtt{P}_i \subseteq \bigcup_{j=1 \ldots n} \mathtt{AP}_{\mathcal{K}_j}$ and I_i is a non-empty interval of $\mathbb{R}_{\geq 0}$. The cylinder set of ω is

$$\mathcal{C}^P_{\mathcal{J}}(\mathtt{P}_1 I_1 \mathtt{P}_2 I_2 \ldots \mathtt{P}_m) = \{\mathtt{P}_1 d_1 \mathtt{P}_2 d_2 \ldots \mathtt{P}_m \cdots \in \Omega^{\mathtt{AP}}(\mathcal{J}) \mid \forall i < m, d_j \in I_j\}$$

As for the action-based cylinders we can now define the probability of seeing a run matching ω from state $\mathbf{s}$ as:

$$\mathbb{P}_{\mathbf{s}}(\mathtt{P}_1 I_1 \mathtt{P}_2 I_2 \ldots \mathtt{P}_m) = (\mathtt{P}_1 \stackrel{?}{=} \mathtt{Pm}(\mathbf{s})) \cdot \sum_{i=1}^{n} \left(\int_{t \in I_1} \mu_{\mathbf{s}^i}(t) \cdot \left(\prod_{j \neq i} \int_{\tau > t} \mu_{\mathbf{s}^j}(\tau) \, \mathrm{d}\tau \right) \cdot \left(\sum_{a \in {\Sigma_\mathtt{o}}^i} \left(\gamma_{([\mathbf{s}]^t)^i}(a) \cdot \mathbb{P}_{\left[[\mathbf{s}]^t\right]^{\mathtt{a}!}}(\mathtt{P}_2 I_2 \ldots \mathtt{P}_m) \, \mathrm{d}t \right) \right) \right)$$

with base case $\mathbb{P}_{\mathbf{s}}(\mathtt{P}) = (\mathtt{P} \stackrel{?}{=} \mathtt{Pm}(\mathbf{s}))$ and $(\mathtt{P}_1 \stackrel{?}{=} \mathtt{P}_2) = 1$ whenever $(\mathtt{P}_1 = \mathtt{P}_2)$ and 0 otherwise.

In this expression it is first checked if the set of propositions true in $\mathbf{s}$ matches those required by the prefix. Then, because it is unknown which component is going to perform the next action we sum over all components. Inside the sum, the component winning the race chooses a delay in I_1 and all others a delay that is larger. Afterwards, we sum over all possible output actions of the winning component, take the probability of performing the action into account and multiply by the probability of observing the remaining prefix after performing the action.

3.7 Stochastic Timed Automata

In the previous section we saw how a stochastic semantics is given for a TLTS by associating a delay density to each state and a probability mass to each possible output action from a state. In this section we describe how these functions are defined for TA and get semantics for a structure we call stochastic timed automaton (STA). Syntactically an STA is almost equivalent to a TA with the additions that (1) each location is added a *rate* parameter that indicate how fast the STA wants to leave the location, (2) edges may only be guarded by lower bounds and (3) invariants can only be upper bounds. These syntactic restrictions ensure that if a guard is enabled at some point then it remains enabled until the invariant is violated.

Definition 16 (Stochastic timed automaton)**.** A stochastic timed automaton is a tuple $(\mathtt{L}, \mathtt{l}_0, \Sigma_\mathtt{o}, \Sigma_\mathtt{i}, \mathtt{X}, \mathtt{E}, \mathtt{I}, \mathtt{R})$ where

- $\mathtt{L}$ is a finite set of control locations,
- $\mathtt{l}_0 \in \mathtt{L}$ is the initial location of the STA,
- $\Sigma_\mathtt{o}$ is a set of output actions,
- $\Sigma_\mathtt{i}$ is a set of input actions,
- $\mathtt{X}$ is a set of clocks,
- $\mathtt{E} \subseteq \mathtt{L} \times \mathtt{B}^{\geq}(\mathtt{X}) \times (\Sigma_\mathtt{o} \cup \Sigma_\mathtt{i}) \times 2^{\mathtt{X}} \times \mathtt{L}$ is a set of edges between locations,
- $\mathtt{I} : \mathtt{L} \to \mathtt{B}^{\leq}(\mathtt{X})$ assign invariants to locations and
- $\mathtt{R} : \mathtt{L} \to \mathbb{R}_{>0}$ assign rates to location.

Let $s = (\mathtt{l}, \mathtt{v})$ be a state of the STA $S = (\mathtt{L}, \mathtt{l}_0, \Sigma_\mathtt{o}, \Sigma_\mathtt{i}, \mathtt{X}, \mathtt{E}, \mathtt{I}, \mathtt{R})$ then a delay is stochastically selected in s in the following way: if the possible delays are bounded i.e. $\mathtt{I}(\mathtt{l}) \neq \mathtt{tt}$ then the delay is selected uniformly between the minimal delay before a guard is satisfied and the maximal delay where the invariant is still satisfied. In case it is not bounded then the delay is selected from a shifted exponential distribution $\mathtt{Exp}_{\mathtt{R}(\mathtt{l}),min}$ where min is the minimal

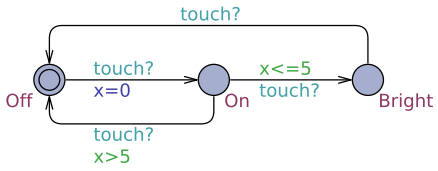
delay before any guard is satisfied. Formally, we define the functions $\texttt{min}$ and $\texttt{max}$ which extracts the minimal and maximal delay from $s = (\texttt{l}, \texttt{v})$ as

$$\texttt{min}(s) = \texttt{inf}\{d \mid \exists\, \texttt{a!} \text{ s.t. } s \xrightarrow{d} \xrightarrow{\texttt{a!}}\} \quad \text{and} \quad \texttt{max}(s) = \texttt{sup}\{d \mid s \xrightarrow{d}\}.$$

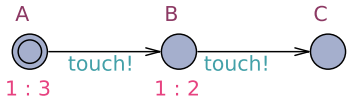
Given these two functions the delay density is defined as

$$\mu_s = \begin{cases} \texttt{Uni}^{\mu}_{\texttt{min}(s),\texttt{max}(s)} & \text{if } \texttt{I}(\texttt{l}) \neq \texttt{tt} \\ \texttt{Exp}_{\texttt{R}(\texttt{l}),\texttt{min}(s)} & \text{Otherwise} \end{cases}$$

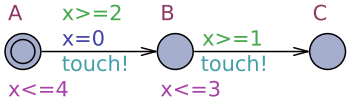
Regarding actions we use a uniform split among all possible actions. With a few synctactic additions to the definition of STA this is however easily generalised to user defined probability mass functions.



(a) Smart lamp from Figure 3.4.



(b) An exponential person.



(c) An uniform person.

Figure 3.8: Smart lamp composed with two different "persons".

Example 8. Consider putting the STA in Figure 3.8b in parallel with the TA of Figure 3.8a then we may consider "what is the probability the lamp is put into bright within 5 time units". Since Figure 3.8b has no invariants the delays in both locations are selected from exponential distributions with parameters $\frac{1}{3}$ and $\frac{1}{2}$. The timed propositional runs of interest are

$$\mathcal{C}_{\mathcal{J}} = \{\langle \mathsf{Off}, \mathsf{A}\rangle t \langle \mathsf{On}, \mathsf{B}\rangle \tau \langle \mathsf{Bright}, \mathsf{C}\rangle \ldots \mid t + \tau \leq 5\}.$$

The probability of these runs are

$$\mathbb{P}(\mathcal{C}_{\mathcal{J}}) = \int_0^5 \mathtt{Exp}_{\frac{1}{3}}(t) \int_0^{5-t} \mathtt{Exp}_{\frac{1}{2}}(\tau)\, \mathrm{d}\tau\, \mathrm{d}t \approx 0.59.$$

Consider inserting Figure 3.8c into the system instead of Figure 3.8b. The runs satisfying our property are then

$$\mathcal{C}_{\mathcal{J}} = \left\{ \langle \mathsf{Off}, \mathsf{A}\rangle t \langle \mathsf{On}, \mathsf{B}\rangle \tau \langle \mathsf{Bright}, \mathsf{C}\rangle \ldots \;\middle|\; \begin{array}{l} t \in [2,4],\ \tau \in [1,3] \\ \text{and } t + \tau \leq 5 \end{array} \right\}$$

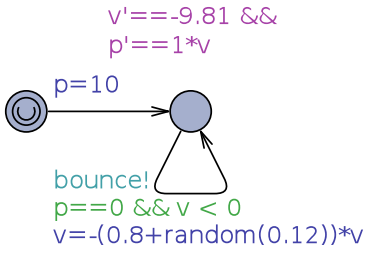
The probability is now calculated as

$$\begin{aligned}
\mathbb{P}(\mathcal{C}_{\mathcal{J}}) &= \int_2^4 \mathtt{Uni}^{\mu}_{2,4}(t) \int_{\tau=1}^{min(3,5-t)} \mathtt{Uni}^{\mu}_{1,3}(\tau)\, \mathrm{d}\tau\, \mathrm{d}t \\
&= \int_t^4 \frac{1}{2} \int_1^{5-t} \frac{1}{2}\, \mathrm{d}\tau\, \mathrm{d}t \\
&= \int_t^4 \frac{1}{2} \cdot \left(\frac{5-t}{2} - \frac{1}{2} \right) \mathrm{d}t \\
&= \frac{1}{2} \left[\frac{5 \cdot t}{2} - \frac{t^2}{4} - \frac{1}{2} \cdot t \right]_2^4 \\
&= \frac{1}{2} \left(\tfrac{5 \cdot 4}{2} - \frac{4^2}{4} - \frac{4}{2} - \left(\frac{5 \cdot 2}{2} - \frac{2^2}{4} - \frac{2}{2} \right) \right) \\
&= \frac{1}{2} \left(10 - 4 - 2 - (5 - 1 - 1) \right) \\
&= \frac{1}{2}
\end{aligned}$$

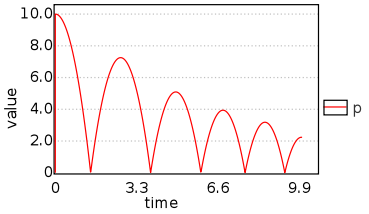
Remark 3. In Example 8, the guard from On to Bright in Figure 3.8a is guarded by an upper bound despite the syntax in Definition 16 disallowing it. This works in Uppaal SMC because the synchronisation on the edge is an input thus the guard is only used to ensure that the underlying TLTS is deterministic regarding the touch? synchronisation.

3.8 Stochastic Hybrid Systems

As stressed in the introduction future computer systems will be conditioning their behaviour on other physical quantities than just time. These physical quantities are state variables in the sense that in any state one can observe the current energy level or the heat in a room; but they also change as time progresses thus time delays affect their value



(a) A stochastic hybrid automaton modelling the ball in Uppaal SMC



(b) A simulation of the bouncing the ball

Figure 3.9: An stochastic hybrid automaton modelling a bouncing ball.

Figure 3.9a shows a stochastic hybrid automaton (SHA) modelling a bouncing ball. In this model there are two clocks, p and v, measuring the height of the ball and its vertical velocity respectively. Initially, p is set to ten and v is set to zero. Then the velocity changes with the acceleration constant -9.81 and the position with the rate given by the velocity. These rates can be seen by the invariant expressions v'==−9.81 and p'==1∗v in the figure. Whenever s is zero the velocity is multiplied by a random factor of 0.8 and negated - simulating the bouncing of the ball on the ground. In Figure 3.9b the vertical position of the ball is shown (y-axis) as time progresses (x-axis).

An stochastic hybrid automaton (SHA) may define the rate of a clock not only by the value of another clock or a number but also arbitrary arithmetic expressions over the clocks. This allows for expressing ordinary differential equations.

Definition 17 (Clock Expression)**.** Let $\mathtt{X}$ be a set of clocks then the set of expressions $\text{Expr}_{\mathtt{X}}$ is generated by the syntax

$$e, e_1, e_2 ::= \mathtt{x} \mid r \mid e_1 \mathsf{op} e_2 \mid (e)$$

where $\mathtt{x} \in \mathtt{X}$, $r \in \mathbb{R}$ and $\mathsf{op} \in \{+, -, \cdot, /\}$.

Definition 18. A stochastic hybrid automaton is a tuple $(\mathtt{L}, \mathtt{l}_0, \Sigma_\mathtt{o}, \Sigma_\mathtt{i}, \mathtt{X}, \mathtt{E}, \mathtt{I}, \mathtt{R}, \mathtt{X}^\mathtt{R})$ where

- $\mathtt{L}$ is a finite set of control locations,
- $\mathtt{l}_0 \in \mathtt{L}$ is the initial location of the SHA,
- $\Sigma_\mathtt{o}$ is a set of output actions,
- $\Sigma_\mathtt{i}$ is a set of input actions,
- $\mathtt{X}$ is a set of clocks,
- $\mathtt{E} \subseteq \mathtt{L} \times \mathtt{B}^{\geq}(\mathtt{X}) \times (\Sigma_\mathtt{o} \cup \Sigma_\mathtt{i}) \times 2^\mathtt{X} \times \mathtt{L}$ is a set of edges between locations,
- $\mathtt{I} : \mathtt{L} \to \mathtt{B}^{\leq}(\mathtt{X})$ assign invariants to locations and
- $\mathtt{R} : \mathtt{L} \to \mathbb{R}_{>0}$ assign rates to location,
- $\mathtt{X}^\mathtt{R} : \mathtt{L} \to \mathtt{X} \to \textsc{Expr}_\mathtt{X}$ assign rates to each clock in each location

Let $(\mathtt{l}, \mathtt{v})$ be a state of SHA $(\mathtt{L}, \mathtt{l}_0, \Sigma_\mathtt{o}, \Sigma_\mathtt{i}, \mathtt{X}, \mathtt{E}, \mathtt{I}, \mathtt{R}, \mathtt{X}^\mathtt{R})$ with $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_n\}$ and let

$$\begin{aligned} \tfrac{\mathrm{d}\mathtt{x}_1}{\mathrm{d}\tau} &= \mathtt{X}^\mathtt{R}(\mathtt{l})(\mathtt{x}_1) \\ \tfrac{\mathrm{d}\mathtt{x}_2}{\mathrm{d}\tau} &= \mathtt{X}^\mathtt{R}(\mathtt{l})(\mathtt{x}_2) \\ &\vdots \\ \tfrac{\mathrm{d}\mathtt{x}_n}{\mathrm{d}\tau} &= \mathtt{X}^\mathtt{R}(\mathtt{l})(\mathtt{x}_n), \end{aligned}$$

be the set of differentials equations induced by $\mathtt{X}^\mathtt{R}$. Let $\{\bar{\mathtt{x}} : \mathbb{R}_{\geq 0} \to \mathbb{R} \mid \mathtt{x} \in \mathtt{X} \wedge \bar{\mathtt{x}}(0) = \mathtt{v}(\mathtt{x})\}$ be the solution to the differential equations. Then we let $(\mathtt{v} + \mathtt{X}^\mathtt{R} \cdot d)(\mathtt{x}) = \bar{\mathtt{x}}(d)$. With this generalised delay operation on valuations we can straightforwardly generalise the semantics from STAs to SHAs and likewise the stochastic semantics is easily generalised.

Remark 4. Uppaal SMC does not solve the differential equations analytically. Instead, a step size, ϵ, is selected by the user and the SMC engine ensures that any delay step is below this ϵ. Uppaal SMC is essentially doing a fixed step Euler integration.

Remark 5 (Negative Clock Rates)**.**
The assumption driving the semantics of SHAs until now is that if a guard is enabled at some point, then it remains enabled for all delays afterwards. This assumption is true for SHAs with only positive clock rates but at the moment negative clock rates are present in the SHA this assumption fails. As an example of this, consider the SHA in Figure 3.10 where the clock `x` is 10 when entering location `B` and the edge towards `C` is enabled. The SHA therefore choose a delay between $[0, 12]$, say it chooses 2. Then after the delay, `x` is 8 and the edge is no longer enabled. Since `y` is now 12 the SHA cannot delay and exhibits a time-lock. In fact, the delay density assigns a probability density to a delay for which the SHA cannot do any action which violates the requirements from Definition 15.

The problem is that if a clock `x` has a negative rate, then a lower bound, `x <= n`, effectively becomes an upper bound. Similarly, upper bounds becomes lower bounds. One way to syntactically avoid this problem is to insist that guards over clocks with negative rate must be upper bound and likewise invariants must be lower bounds. In Uppaal SMC this syntactic check is not performed and t is the responsibility of the modeller to ensure models do not encounter situations as described.

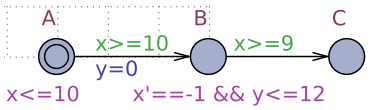


Figure 3.10: An SHA violating the model sanity property.

Specification Formalisms

4

Having covered the modelling side of software verification we turn to the specification side. Two different families of logics have been developed over the years: *branching-time* and *linear-time* logics. The latter considers the future of a single run to be fixed whereas the former considers the future of a single run to be undetermined when doing verification. This means that branching time logics can reason on the entire computation tree.

Linear time logics fits into the SMC framework as the satisfaction of a linear time property on a run can be determined by considering only that single run. For branching time logics resampling during a run is necessary. Younes [123] and Larsen and Skou [85] based their work on branching time logics.

4.1 Linear Temporal Logic

Linear temporal logic (LTL) is a logic that extends basic propositional logic with modalities for specifying requirements of the future. Definition 19 gives the grammar for LTL formulas.

Definition 19. A linear temporal logic formula over a set of propositions $\mathtt{AP}$ is generated by the grammar:

$$\varphi, \varphi_1, \varphi_2 ::= \mathsf{tt} \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U} \varphi_2$$

where $p \in \mathtt{AP}$.

Linear temporal logic is interpreted over propositional runs. The intuitive meaning of a formula $\varphi_1 \mathsf{U} \varphi_2$ is that at some point during a run φ_2 holds and φ_1 holds at all other places before this time. The formula $\mathsf{X}\varphi$ expresses that φ holds in the next state of the run.

Formally, let $\pi = \mathtt{P}_0, \mathtt{P}_1 \ldots$, be a propositional run and let $\pi^{i:} = \mathtt{P}_i, \mathtt{P}_{i+1} \ldots$ then we define a satisfaction relation between an LTL formula φ and π inductively as follows

- $\pi \vDash \mathtt{tt}$
- $\pi \vDash p$ if $p \in \mathtt{P}_0$
- $\pi \vDash \neg\varphi$ if $\pi \nvDash \varphi$
- $\pi \vDash \varphi_1 \wedge \varphi_2$ if $\pi \vDash \varphi_1$ and $\pi \vDash \varphi_2$
- $\pi \vDash \mathsf{X}\varphi$ if $\pi^{1:} \vDash \varphi$
- $\pi \vDash \varphi_1 \mathsf{U} \varphi_2$ if there exists $i \geq 0$ such that $\pi^{i:} \vDash \varphi_2$ and for all $0 \leq j < i$ we have $\pi^{j:} \vDash \varphi_1$

Common short hand notations are $\Diamond\varphi = \mathtt{tt}\,\mathsf{U}\,\varphi$, $\Box\varphi = \neg\Diamond\neg\varphi$ along with the classic boolean operators $\vee$, $\Longrightarrow$ and $\leftrightarrow$.

If $\mathtt{M}$ is a model that generates propositional runs and $\Omega(\mathtt{M})$ is the set containing all those runs then we say that $\mathtt{M}$ satisfies the LTL formula φ if for all $\pi \in \Omega(\mathtt{M})$, $\pi \vDash \varphi$ i.e. a model satisfies an LTL formula if all runs of the model satisfies the formula.

Example 9. Consider the server in Figure 3.6 that may break down and be repaired. One specification for this system could be $\Box\Diamond\mathtt{Idle}$ meaning that it is always the case that the server will return to idle. Clearly, this is not the case since there is a possibility of looping in `Repair`.

The standard model checking technique for LTL is to negate the formula φ to $\bar{\varphi} = \neg\varphi$, translate $\bar{\varphi}$ into a Büchi automaton A and construct the product $\mathtt{M} \times A$. If the language $\mathcal{L}(\mathtt{M} \times A) = \emptyset$ then $\mathtt{M} \vDash \varphi$ otherwise all $\pi \in \mathcal{L}(\mathtt{M} \times A)$ are counter examples i.e. $\pi \nvDash \varphi$ [12].

Consider now that $\mathtt{M}$ is a DTMC and assume we want to calculate the probability that a random run satisfies φ, denoted by $\mathbb{P}_{\mathtt{M}}(\varphi)$. Firstly, we need to assert that the set

$$\mathtt{Sat}_\phi = \{\pi \in \Omega(\mathtt{M}) \mid \pi \vDash \varphi\},$$

is measurable with respect to the σ-algebra over $\mathtt{M}$. This is for instance shown by Baier and Katoen [12].

For measuring the probability $\mathbb{P}_{\mathtt{M}}(\varphi)$ with PMC, an automata based approach similar to MC is adopted. First $\neg\varphi$ is converted into an automaton A. Let $\mathtt{Sat}_A$ be the of set runs accepted by A ,then $\mathbb{P}_{\mathtt{M}}(\varphi) = 1 - \mathbb{P}_{\mathtt{M}}(\mathtt{Sat}_A)$. To guarantee the measurability of $\mathtt{Sat}_A$ the product $\mathtt{M} \times A$ must be a DTMC and to guarantee this, A must be deterministic [12]. Deterministic Büchi automata are however strictly less expressive than their non-deterministic counterpart and

as LTL is characterised by non-deterministic Büchi automata another automata construction is necessary. Deterministic Rabin Automata fits this purpose. [12]

The satisfaction of an LTL formula on a single run can be established for LTL thus one could guess that LTL is a proper specification language for SMC. However, the semantics of LTL is not suitable for SMC since LTL is interpreted over infinite runs and the SMC approach can only provide finite prefixes. A common way to cope with this is to use the *infinite extension semantics* [17] where the last set of propositions is repeated infinitely often. An alternative is requiring that all U-operators are satisfied along the finite run and introduce two operators, $\overline{\mathsf{X}}$ and $\underline{\mathsf{X}}$, to replace the X operator. The $\overline{\mathsf{X}}\varphi$ operator is satisfied if there exists a next state that satisfies φ, whereas $\underline{\mathsf{X}}\varphi$ is satisfied if there exists a next state where φ is satisfied or there does not exist a next state. This semantics is called finite LTL [17]. Bauer and Haslum [17] showed that the infinite extension semantics and finite LTL coincide as long as we avoid using any of the X operators. A problem with using either of the above semantics is that they collapse runs that are for sure not satisfied and runs for which the run was just too short to draw a conclusion, with respect to the standard LTL semantics, into unsatisfaction. Bauer et al. [19] proposed the logic LTL_3 with verification outcomes in the set $\mathbb{B}_3 = \{\top, \bot, ?\}$ with $\top$ being satisfaction, $\bot$ not satisfied and ? meaning "don't know". This semantics distinguishes the two cases that were problematic for the other finite semantics but does not fit well into the SMC framework as the distribution is no longer binomial.

Zuliani et al. [124] introduced a logic called bounded LTL which is interpreted over infinite runs but for which there always exists a finite prefix after which the runs satisfies the property or not. The semantics is similar to standard LTL but the U and R operators are bounded which gives the ability to determine the satisfaction on a finite prefix. Bounded LTL has been applied in the SMC framework - although with a Bayesian interpretation of probabilities as opposed to our frequentist interpretation.

4.2 Metric Temporal Logic

Metric temporal logic (MTL) [83] is an extension of LTL that takes timing aspects of events into account. Where LTL can only assert an airbag deploys after a crash, MTL can assert that it takes at most 5 ms to deploy. On a syntactical level, Definition 20, LTL and MTL looks similar with the main difference being that the U operator is bounded by a time interval.

Definition 20. A metric temporal logic formula over the propositions $\mathtt{AP}$ is generated by the grammar:

$$\varphi, \varphi_1, \varphi_2 ::= \mathsf{tt} \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U}_I \varphi_2$$

where $p \in \mathtt{AP}$ and I is an open or closed interval of $\mathbb{R}$.

Two different semantics have been considered for MTL: the *point-wise* and

continuous semantics. The point-wise corresponds to the view that a system is observed at distinct points in times and only at those time points is it known what propositions are true. The continuous semantics assumes that a system can be observed continuously throughout its execution and thus one knows what propositions are true at any point in time.

In the point-wise semantics an MTL formula is evaluated over timed propositional runs. Here we only give the semantics of the U operator as the others are similar to LTL. Let φ be an MTL formula and $\pi = \mathtt{P}_0\, d_0\, \mathtt{P}_1\, d_1 \ldots$ be a timed propositional run then the satisfaction relation is defined as below.

- $\pi \vDash \varphi_1 \mathsf{U}_I \varphi_2$ if there exists $i \geq 0$ such that $\sum_{k=0}^{i-1} d_k \in I, \pi^{i:} \vDash \varphi_2$ and for all $0 \leq j < i$, $\pi^{j:} \vDash \varphi_1$

The continuous semantics is interpreted over *signals* which are mappings from the reals to sets of the propositions. Formally, a signal is a function

$$\mathsf{S} : \mathbb{R} \to 2^{\mathtt{AP}}.$$

Let S be a signal and $d \in \mathbb{R}_{\geq 0}$, then $(\mathsf{S} + d)$ is the signal S' defined as $\mathsf{S}'(x) = \mathsf{S}(x + d)$ i.e. it shifts the signal by d time units into the future.

Let S be a signal then we define the satisfaction relation as:

- $\mathsf{S} \vDash \mathsf{tt}$
- $\mathsf{S} \vDash p$ if $p \in \mathsf{S}(0)$
- $\mathsf{S} \vDash \neg\varphi$ if $\mathsf{S} \nvDash \varphi$
- $\mathsf{S} \vDash \varphi_1 \wedge \varphi_2$ if $\mathsf{S} \vDash \varphi_1$ and $\mathsf{S} \vDash \varphi_2$
- $\mathsf{S} \vDash \varphi_1 \mathsf{U}_I \varphi_2$ if there exists $i \in I$ such that $, (\mathsf{S} + i) \vDash \varphi_2$ and for all $0 \leq j < i$, $(\mathsf{S} + j) \vDash \varphi_1$

Notice that the X operator has been omitted because there is no next state in a continuous setting. Consider the timed propositional run $\pi = \mathtt{P}_0\, d_0\, \mathtt{P}_1\, d_1 \ldots$ and a signal constructed to mimic the timed propositional run

$$\mathsf{S}(x) = \begin{cases} \mathtt{P}_0 & \text{if } x < d_0 \\ \mathtt{P}_1 & \text{if } d_0 \leq x < d_0 + d_1 \\ \vdots & \end{cases}$$

With this construction one could guess that the two interpretations are equivalent. This is however not the case: consider the formula $\mathsf{tt}\mathsf{U}_{[2;5]}a$ and the timed propositional run $\pi = \emptyset\, 1\, \{a\}\, 7\, \emptyset \ldots$ and the signal

$$\mathsf{S}(x) = \begin{cases} \emptyset & \text{if } x < 1 \\ \{a\} & \text{if } 1 \leq x < 1 + 7 \\ \emptyset & \text{if } 1 + 7 < x \end{cases}$$

It is clear that $\pi \not\models \varphi$ because a is true after one time unit and the next set of propositions (where a is not true) is observed after 7 time units whereas $\mathsf{S} \models \varphi$ because a is true after 2 units as it has complete oberservability of the system. This is just one example where they do not coincide but in general the point-wise semantics is less expressive than the continuous semantics. The MC problem is known to be undecidable for the continuous semantics [71] but there are some decidable sublogics. Metric Interval Temporal Logic [9] is a decidable sub-logic where punctuality is not allowed i.e. all intervals are non-singular intervals. For many years it was thought that any kind of punctuality constraint would break the decidability result. Bouyer et al. [29] showed however that punctuality could be allowed as long as all intervals were bounded. For the point-wise semantics the MC problem is decidable over finite propositional runs with non-primitive recursive complexity and it is also decidable for the safety fragment of MTL over infinite words [97, 98]. Despite these undecidability and complexity issues of MTL we wish to use it for SMC anyways. The way we do this is by constructing observers that can recognise if a property is satisfied by a single timed propositional run. Nickovic and Piterman [95] constructed deterministic timed automata for MTL under a continuous semantics relying on a finite variability assumption, i.e. assumed that the system under evaluation can only change state for a known finite number of times within a time interval. Since the automata construction is deterministic this could prove useful for determining if a run satisfies a formula. The finite variability assumption is likely to hold because systems operate under a certain frequency but finding the bound on the number of changes is non-trivial. Maler et al. [89] proved that MTL without the finite variability is non-deterministic in the sense that no deterministic observer can be made which can check if a MTL formula is satisfied. This is surely a downfall as we want a real-time linear time specification formalism for SMC.

Dynamic Systems

5

The introduction already mentioned the need for modelling dynamic systems where components can be instantiated and cease to exist during the life span of a system. In this chapter TLTSs is generalised to dynamic network of TLTSs and the logic quantified dynamic metric temporal logic (QDTML) is created as an extension of MTL.

5.1 Modelling

A classic example of a system that relies on instantiating components on the fly is a client-server architecture as shown in Figure 5.1.

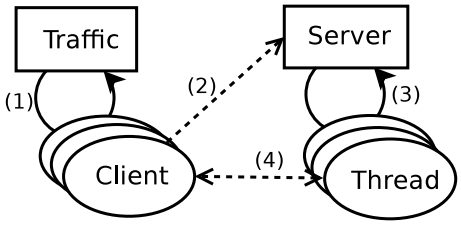


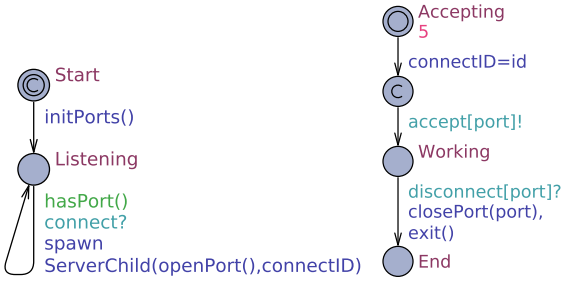
Figure 5.1: A client server

A main thread of computation is on the server side listening for incoming connections generated by the traffic on the network. In response to an incoming connection the main thread immediately *spawns* a thread for handling the connected client and returns to listening for connections. The spawned thread now communicates with the client and terminates when the client disconnects.

In Figure 5.2a and Figure 5.2b is shown a model of the main thread and the spawned threads respectively as UPPAAL SMC SHAs. Figure 5.2a initially performs some internal setup with `initPorts ()`, then it awaits an incoming connection by listening for a synchronisation `connect?` and checks it can support an extra connection by guarding that edge with `hasPort()`. If the connection

can be made a ServerChild is spawned to handle the connection by the update spawn ServerChild (openPort(),connectID). As part of this spawning a port id, returned by openPort(), is given to the spawned ServerChild and an id (connectID) to indicate what client the ServerChild should communicate with is forwarded.

When spawned, Figure 5.2b first has an arbitrary delay in the Accepting location to model that there is an initial setup time when spawning threads. Then it immediately accepts the connection on the port granted by the server by a synchronisation accept[port]!. Afterwards, in Working, it awaits a disconnect? request from the client during which the port is closed and the ServerChild itself cease to exist by the update exit().



(a) Server()

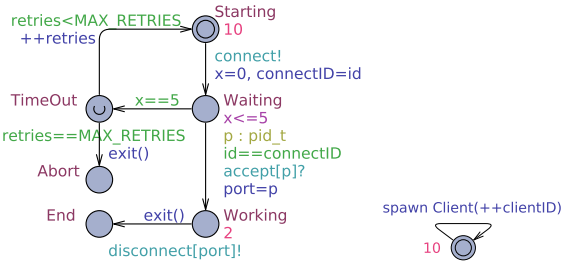
(b) ServerChild(const pid_t port, const int id)

Figure 5.2:

In Figure 5.3a and Figure 5.3b the two remaining SHA templates for the model is shown. Figure 5.3b models incoming traffic on the network by spawning a Client at random times and gives each client a unique id.

The client, Figure 5.3a, at the start tries to connect and while doing so write its id to the global variable connectID which is captured by the Server to link the Child to its designated ServerChild. Afterwards the client will be Waiting for an accept after which it will go into Working. When done working the client will disconnect? and terminate by exit(). If the accept is not granted, the client enters a TimeOut-retry loop.

In the framework developed in this thesis a dynamic network consist of a dynamically evolving number of components where components are running instances of a *template*. The set of templates a component can be instantiated from is given by a template collection $\mathcal{J} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)$ where each template defines a TLTS along with information about when another template should be spawned.



(a) Client (**const** int id) (b) Traffic ()

Figure 5.3:

Definition 21 (Template Collection)**.** A template collection over the output action Σ_o partitioned into n disjoint sets ${\Sigma_o}^1, \ldots, {\Sigma_o}^n$ is a tuple $\mathcal{J} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)$ where for all i $\mathcal{T}_i = (\mathcal{K}_i, \mathtt{spawn}^i)$ with

- $\mathcal{K}_i = (S^i, {s_0}^i, {\Sigma_o}^i, \overline{\Sigma_o} \setminus \overline{{\Sigma_o}^i}, \mathtt{AP_i}, \mathtt{Pm}_i, \rightarrow^i)$ being a TLTS and
- $\mathtt{spawn}^i : S^i \times {\Sigma_o}^i \rightarrow 2^{\{\mathcal{T}_1 \ldots \mathcal{T}_n\}}$ for each state-output- action pair giving a set of templates to be spawned while performing that action,

and for any $j \neq k$ $S^j \cap S^k = \emptyset$.

For statically encoded systems the index in the state vector was used for addressing individual TLTSs in the semantics. In a dynamic setting this is not possible, partly because we never know how many TLTSs are active in the system and partly because we cannot know what type of TLTS is located at any given index. Instead, we give names to the TLTSs and at the state level store the active names and map those to a TLTS and to a state. Let PNames be a set of names then a state of $\mathcal{J} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)$ is a tuple $(\mathsf{Active}, \mathtt{T}, \mathtt{Sm})$ where

- $\mathsf{Active} \subseteq \mathsf{PNames}$ is the names for the active components,
- $\mathtt{T} : \mathsf{Active} \rightarrow \{\mathcal{T}_1 \ldots \mathcal{T}_n\}$ maps the names to their given templates and
- $\mathtt{Sm} : \mathsf{Active} \rightarrow \bigcup_{i=1\ldots n} S^i$ maps the names to states.

Naturally, we require that if a name is mapped to a given template then that same name is mapped to a state of the TLTS of that template. Formally, if $\mathtt{T}(k) = \mathcal{T}$ and $\mathcal{T} = ((S, s_0, \Sigma_o, \overline{\Sigma_o}, \mathtt{AP}, \mathtt{Pm}, \rightarrow), \mathtt{spawn})$ then $\mathtt{Sm}(k) \in S$. For the actual spawning of a template we define an infix operator $\oplus$ that accepts a state and a template and *instantiates* that template to a name: let $(\mathsf{Active}, \mathtt{T}, \mathtt{Sm})$ be a state and $\mathcal{T} = ((S, s_0, \Sigma_o, \Sigma_i, \mathtt{AP}, \mathtt{Pm}, \rightarrow), \mathtt{spawn})$ a template then

$$(\mathsf{Active}, \mathtt{T}, \mathtt{Sm}) \oplus \mathcal{T} = (\mathsf{Active} \cup \{k\}, \mathtt{T}[k \mapsto \mathcal{T}], \mathtt{Sm}[k \mapsto s_0]),$$

where $k \in \mathsf{PNames} \setminus \mathsf{Active}$ and $f[k \mapsto i]$ updates f to map k to i. We generalise this spawn function to spawn a set (P) of templates recursively in the following

way

$$(\mathsf{Active}, \mathtt{T}, \mathtt{Sm}) \oplus P = ((\mathsf{Active}, \mathtt{T}, \mathtt{Sm}) \oplus p) \oplus (P \setminus p),$$

for $p \in P$ and base case $(\mathsf{Active}, \mathtt{T}, \mathtt{Sm}) \oplus \emptyset = (\mathsf{Active}, \mathtt{T}, \mathtt{Sm})$.

The transition relation of $\mathcal{J}$ is defined as

- $(\mathsf{Active}, \mathtt{T}, \mathtt{Sm}) \xrightarrow{d} (\mathsf{Active}, \mathtt{T}, \mathtt{Sm}')$ if for all $k \in \mathsf{Active}\ \mathtt{Sm}(k) \xrightarrow{d} \mathtt{Sm}'(k)$
- $(\mathsf{Active}, \mathtt{T}, \mathtt{Sm}) \xrightarrow[k]{\mathtt{a}!} (\mathsf{Active}, \mathtt{T}, \mathtt{Sm}') \oplus P$ where $k \in \mathsf{Active}$, $\mathtt{Sm}(k) \xrightarrow{\mathtt{a}!} \mathtt{Sm}'(k)$, for all $n \in (\mathsf{Active} \setminus \{k\})$ if $\mathtt{T}(n) \neq \mathtt{T}(k)$ then $\mathtt{Sm}(n) \xrightarrow{\mathtt{a}?} \mathtt{Sm}'(n)$ otherwise $\mathtt{Sm}(n) = \mathtt{Sm}'(n)$, $\mathtt{T}(k) = (\mathcal{K}, \mathtt{spawn})$ and $P = \mathtt{spawn}(\mathtt{Sm}(k), \mathtt{a}!)$.

The first rule expresses that the entire network may delay if all instantiated components can delay. The second rule says that for the component with name k to do an action $\mathtt{a}!$, that component should do $\mathtt{a}!$, all those with a different template should do the corresponding input and all those with the same template should just ignore the input. After, atomically, performing those transitions the set P of TLTSs is spawned as requested by the component with name k. For simplicity we let the initial state be defined as the state where each TLTS is instantiated with one instance i.e. $\mathbf{s}_0 = (\emptyset, \_, \_) \oplus \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$.

The set of runs over $\mathcal{J}$, $\Omega(\mathcal{J})$, is straightforwardly defined as for statically encoded networks but a slight modification for the propositional runs is needed. For statically encoded networks we defined the function $\mathtt{Pm}_{\mathcal{J}}$ that returned the union of all propositions that were true for each individual TLTS. For dynamic networks we redefine $\mathtt{Pm}_{\mathcal{J}}$ to also index what name satisfies a given proposition and to give the type of each active name. Formally,

$$\begin{aligned}\mathtt{Pm}_{\mathcal{J}}((\mathsf{Active}, \mathtt{T}, \mathtt{Sm})) = &\{(a, p) \mid a \in \mathsf{Active} \wedge p \in \mathtt{Pm}_{\mathtt{T}(a)}(\mathtt{Sm}(a))\} \bigcup \\ &\{(a, \mathtt{T}(a)) \mid a \in \mathsf{Active}\}\end{aligned}$$

and the set of propositional runs then given by

$$\Omega^{\mathtt{AP}}(\mathcal{J}) = \{\mathtt{Pm}_{\mathcal{J}}(\mathbf{s}_0) d_0 \mathtt{Pm}_{\mathcal{J}}](\mathbf{s}_1) d_1 \ldots | \mathbf{s}_0 d_0 \mathbf{s}_1 d_1 \cdots \in \Omega(\mathcal{J})\}.$$

Adding the above annotations to the propositional runs will prove useful when we define the semantics of quantified dynamic metric temporal logic (QDTML)

Stochastic Semantics

The stochastic semantics of a template collection is, on a conceptual level, identical to that of a static network. The semantics are still race based with the components selecting a delay independently and the winner selecting an action. Before proceeding to define the probability of a cylinder we need to take care of a few things though: the spawn operator defined so far is non-deterministically choosing a name for the spawned TLTS and when spawning a set of templates

it non-deterministically chooses a spawn ordering. Since this non-determinism result in possibly different propositional runs for the same sequence of states we need to determinise this. We do this by defining a total ordering among the elements in PNames and a total ordering among the templates and always extract the "smallest" element from $\mathsf{PNames} \setminus \mathsf{Active}$ and always spawn templates in the defined order.

We are now ready for defining the probability of a cylinder $\mathcal{C}_{\mathcal{J}} = \mathtt{P}_0 I_0 \mathtt{P}_1 I_1, \ldots, \mathtt{P}_m$ as

$$\mathbb{P}_{\mathbf{s}_0}(\mathcal{C}_{\mathcal{J}}) = (\mathtt{P}_0 \stackrel{?}{=} \mathtt{Pm}_{\mathcal{J}}(\mathbf{s}_0)) \cdot \sum_{k \in \mathsf{Active}} \cdot \left(\sum_{\mathsf{a}! \in {\Sigma_{\mathtt{o}}}^k} \left(\int_{I_0} \mu_{\mathtt{Sm}(k)}(t) \cdot \left(\prod_{k' \mathsf{Active} \setminus \{k\}} \int_{\tau > t} \mu_{\mathtt{Sm}(k')}(\tau)\, \mathrm{d}\tau \right) \cdot \gamma_{[\mathtt{Sm}(k)]^t}(\mathsf{a}!) \cdot \mathbb{P}_{\left[[\mathbf{s}_0]^t\right]^{\mathsf{a}!/k}}(\mathcal{C}^1_{\mathcal{J}})\, \mathrm{d}t \right) \right),$$

where $[\mathbf{s}]^{\mathsf{a}!/k} = \mathbf{s}'$ for $\mathbf{s} \xrightarrow[k]{\mathsf{a}!} \mathbf{s}'$ and ${\Sigma_{\mathtt{o}}}^k$ is the output alphabet associated to the TLTS with name k.

5.2 Specifying Properties

The logics developed for the statically described systems are not well suited for specifying properties of dynamically evolving system. The reason is that the logics would need to refer to specific process instantiations unknown when making the requirements. Our developed specification formalism, quantified dynamic metric temporal logic (QDTML), is MTL extended with constructs to quantify over all possible instances of a specific template and specify requirements to their future behaviour.

Definition 22 (Quantified Dynamic Metric Temporal Logic)**.** Let $\mathcal{T}_1, \ldots, \mathcal{T}_n$ be templates and let PVar be a set of identifiers where each $P \in \mathsf{PVar}$ has a type denoted by $(P : \mathcal{T})$. The set of QDTML formulas is then generated by the syntax

$$\varphi, \varphi_1, \varphi_2 ::= \mathsf{tt} \mid \mathsf{ff} \mid \neg\varphi \mid \mathsf{X}\,\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathsf{U}_I \varphi_2 \mid \mathsf{forall}(P : \mathcal{T})\varphi \mid P.p \mid \mathsf{sum}(P : \mathcal{T})(P.p) \bowtie m$$

where $p \in \mathtt{AP}$ with $(P : \mathcal{T})$ and $\mathcal{T} = ((S, s_0, \Sigma_{\mathtt{o}}, \overline{\Sigma_{\mathtt{o}}}, \mathtt{AP}, \mathtt{Pm}, \rightarrow), \mathtt{spawn})$, $m \in \mathbb{N}$, $\bowtie \in \{<, \leq, \geq, >\}$ and I is an interval of $\mathbb{R}_{\geq 0}$.

The operators in Definition 22 that QDTML has in common with MTL has the same semantical meaning. The construction $\mathsf{forall}(P : \mathcal{T})\varphi$ means that in the current state all the instantiations with type $\mathcal{T}$ should satisfy φ where P is bound to the name of instances of $\mathcal{T}$ while asserting that φ is satisfied. The

construct $P.p$ looks at what name P is bound to and asserts the state of that name satisfy p. In the client-server example a QDTML formula is for instance $\mathsf{forall}(P : \mathtt{ServerChild})P.\mathtt{Working}$ which means that at the current state all $\mathsf{ServerChild}$s should be working. A more interesting formula is perhaps

$$\mathsf{forall}(P : \mathtt{Client})(P.\mathtt{Waiting} \implies \Diamond_{[0;10]}(P.\mathtt{Working}),$$

which asserts that if a client is waiting for a connection, then it is granted a connection within 10 ten time units.

The final new construct is $\mathsf{sum}(P : \mathcal{T})(P.p) \bowtie m$ which sum together the number of instantiations of $\mathcal{T}$ satisfying a proposition and compares the result to a number m.

Syntactically the construction $P.p$ can appear anywhere in a QDTML formula but to have a meaning the variable P must be bound to an instantiation by a binding occurrence $\mathsf{forall}(P : \mathcal{T})\varphi$ thus $P.p$ should only be subformulas of $\mathsf{forall}(P : \mathcal{T})\varphi$. In the semantics a map $M : \mathsf{PVar} \to \mathsf{PNames} \cup \{\star\}$ binds process names to instantiations of the system where $\star$ means "not bound yet". The map is updated to point P to p by writing $M[P \mapsto p]$. With this we can define the satisfaction relation between a run $\pi = \mathtt{P}_0, d_0, \mathtt{P}_1, d_1, \ldots$, and a property as:

- $\pi \vDash^M \mathsf{tt}$
- $\pi \vDash^M \neg\varphi$ if $\pi \nvDash^M \varphi$
- $\pi \vDash^M \mathsf{X}\,\varphi$ if $\pi^{1:} \vDash^M \varphi$
- $\pi \vDash^M \varphi_1 \wedge \varphi_2$ if $\pi \vDash^M \varphi_1$ and $\pi \vDash^M \varphi_2$
- $\pi \vDash^M \varphi_1 \mathsf{U}_I \varphi_2$ if there exists i such that $\pi^{i:} \vDash^M \varphi_2$, $\left(\sum_{k<i} d_k\right) \in I$ and for all $j < i$ we have $\pi^{j:} \vDash^M \varphi_1$
- $\pi \vDash^M \mathsf{forall}(P : \mathcal{T})\varphi$ if for all $(a, \mathcal{T}) \in \mathtt{P}_0$ $\pi \vDash^{M[P \mapsto a]} \varphi$,
- $\pi \vDash^M P.p$ if $M(P) \neq \star$ and there exists $(M(P), p) \in \mathtt{P}_0$
- $\pi \vDash^M \mathsf{sum}(P : \mathcal{T})(P.p) \bowtie m$ if $|\{(a, \mathcal{T}) | (a, \mathcal{T}) \in \mathtt{P} \wedge \exists (a, p) \in \mathtt{P}\}| \bowtie m$.

We say a run π satisfies a property φ, written $\pi \vDash \varphi$ if $\pi \vDash^{M_\star} \varphi$ where $M_\star(P) = \star$ for all $P \in \mathsf{PVar}$.

Thesis Summary

6

This chapter presents the five papers in the second part of this thesis. For each of the papers the abstract is given, the publication history is given and its contribution is presented. Papers A and B are concerned with developing validation techniques for a weighted extension of MTL. Paper C extends the SMC approach to dynamic reconfigurable systems and Paper D presents the logic QDTML for specifying properties of these along with an extenstion of the validation techique from Paper B. Paper E shows the versatility of the SHA formalism by modelling biological systems and finally Paper F proposes how to distribute the SMC effort. The layout of the papers has been altered to fit this thesis and appendices have been moved to the main text.

Paper A - Monitor-Based Statistical Model Checking of WMTL$_{\leq}$

Peter E. Bulychev; Alexandre David; Kim Guldstrand Larsen; Axel Legay; Guangyuan Li; Danny Bøgsted Poulsen; Amelie Stainer

Abstract We present a novel approach and implementation for analysing weighted timed automata with respect to weighted metric temporal logic$_{\leq}$ (WMTL$_{\leq}$). Based on a stochastic semantics of weighted timed automata, we apply statistical model checking (SMC) to estimate and test probabilities of satisfaction with desired levels of confidence. Our approach consists in generation of deterministic monitors for formulae in weighted metric temporal logic$_{\leq}$ (WMTL$_{\leq}$), allowing for efficient SMC by run-time evaluation of a given formula. By necessity, the deterministic observers are in general approximate (over- or under-approximations), but are most often exact and experimentally tight. The technique is implemented in the new tool Casaal that we seamlessly connect to Uppaal SMC in a tool chain. We demonstrate the applicability of our technique and the efficiency of our implementation through a number of case-studies.

Contributions

- Translation from WMTL$_{\leq}$ to non-determistic monitoring automata,
- Tranlsation from WMTL$_{\leq}$ to determistic over- and under-approximating monitoring automata

Publication History The paper was accepted and presented at the 18th International Conference on Logic for Programming and Artifical Intelligence and Reasoning (LPAR) and published in Proceedings of LPAR LNCS 7180 pages 168-182.

Paper B - Rewrite-Based Statistical Model Checking of WMTL$_{[a,b]}$

Peter E. Bulychev; Alexandre David; Kim Guldstrand Larsen; Axel Legay; Guangyuan Li; Danny Bøgsted Poulsen

Abstract We present a new technique for verifying Weighted Metric Temporal Logic (WMTL) properties of weighted timed automata. Our approach relies on statistical model checking combined with a new monitoring algorithm based on rewriting rules. Contrary to existing monitoring approaches for WMTL ours is exact. The technique has been implemented in the statistical model checking engine of Uppaal and experiments indicate that the technique performs faster than existing approaches and leads to more accurate results.

Contributions

- Developed an exact on-the-fly- monitoring technique for WMTL$_{[a,b]}$
- Implementation of monitoring technique for MTL$_{[a,b]}$ subset of WMTL$_{[a,b]}$.

Publication History The paper was accepted and presented at the 3rd International Conference on Runtime Verification (RV) and published in Proceeding of RV LNCS 7687 pages 260-275.

Paper C - Statistical Model Checking of Dynamic Network of Stochastic Hybrid Automata

Alexandre David; Kim Guldstrand Larsen; Axel Legay; Danny Bøgsted Poulsen

Abstract In this paper we present a modelling formalism for dynamic networks of stochastic hybrid automata. In particular, our formalism is based on primitives for the dynamic creation and termination of hybrid automata components during the execution of a system. In this way we allow for natural modelling of concepts such as multiple threads found in various programming paradigms, as well as the dynamic evolution of biological systems.

We provide a natural stochastic semantics of the modelling formalism based on repeated output races between the dynamic evolving components of a system. As specification language we present a quantified extension of the logic metric temporal logic (MTL). As a main contribution of this paper, the statistical model checking engine of Uppaal has been extended to the setting of dynamic networks of hybrid systems and quantified MTL. We demonstrate the usefulness of the extended formalisms in an analysis of a dynamic version of the well-known Train Gate example, as well as in natural monitoring of a MTL formula, where observations may lead to dynamic creation of monitors for sub-formulas.

Contributions

- Introduces dynamic networks of stochastic hybrid automata that allows spawning of components during execution of system,
- introduces Dynamic Metric Temporal Logic (DTML),
- extended $\text{WMTL}_{[a,b]}$ monitoring technique to DMTL.

Publication History The paper was accepted and presented at the 13th International Workshop on Automated Verification of Critical Systems and published in Electronic Communication of the European Association of Software Science and Technology Volume 66.

Paper D - Quantified Dynamic Metric Temporal Logic for Dynamic Networks of Stochastic Hybrid Automata

Alexandre David; Kim Guldstrand Larsen; Axel Legay; Guangyuan Li; Danny Bøgsted Poulsen

Abstract Multiprocessing systems are capable of running multiple processes concurrently. By now such systems have established themselves as the defacto standard for operating systems. At the core of an operating system is the ability to execute programs and as such there must be a primitive for instantiating new processes - also programs are allowed to die/terminate. Operating systems may allow the executing programs to split (spawn) into more computational threads in order to let programs take advantage of concurrent execution as well. One of the most used modelling languages, Timed Automata, is based on multiple automata interacting thus they easily model the concurrent execution of programs. However, this language assumes a fixed size system in the sense that automata cannot be created at will but must be instantiated when the overall system is created. This is in contrast with the fact that developers are able to create threads when needed. In this paper we present our continued work to incorporate spawning of threads into Uppaal SMC. Our modelling language, *Dynamic Networks of Stochastic Hybrid Automata*, is essentially Timed Automata extended with a spawning primitive and a tear-down primitive. The dynamic creation of threads has the side-effect that it is no longer possible to use ordinary logics to specify behaviours of individual threads - because the threads no longer have unique names. In this paper we propose an extension of *Metric Temporal Logic* with means for quantcifying over the dynamically created threads. This makes it possible to zoom in on individual threads and specify requirements to their future behaviour. Furthermore, we present a monitoring procedure for the logic based on rewriting formulas. The presented modelling language and the specification language have been implemented in Uppaal SMC version 4.1.18.

Contributions

- Introduces Quantified Dynamic Metric Temporal Logic (QDMTL),
- extends monitoring technique to QDMTL.

Publication History The paper was accepted and presented at the 14th International conference on Application of Concurrency to System Design (ACSD) and appearded in the proceedings of ACSD.

Paper E - Statistical Model Checking for Biological Systems

Alexandre David; Kim Guldstrand Larsen; Axel Legay; Marius Mikocionis; Danny Bøgsted Poulsen; Sean Sedwards

Abstract Statistical Model Checking (SMC) is a highly scalable simulation-based verification approach for testing and estimating the probability that a stochastic system satisfy a given linear temporal property. The technique has been applied to (discrete and continuous time) Markov chains, stochastic timed automata and most recently hybrid systems using the tool Uppaal SMC. In this paper we enable the application of SMC to complex biological systems, by combining Uppaal SMC with ANIMO, a plugin of the tool Cytoscape used by biologists, as well as with SimBiology®, a plugin of Matlab to simulate reactions. ANIMO and SimBiology® are two domain specific tools that have their own user interfaces and formalisms specifically tailored towards the biological domain. However – though providing means for simulation – both tools lack the powerful analytic capabilities offered by SMC, which in previous work have proved very useful for identifying interesting properties of biological systems. Our aim is to offer the best of the two worlds: optimal domain specific interfaces and formalisms suited to biology combined with powerful SMC analysis techniques for stochastic and hybrid systems. This goal is obtained by developing translators from the XGMML and SBML formats used by Cytoscape and SimBiology® to stochastic and hybrid automata, allowing Uppaal SMC to be used as an efficient backend analysis tool, that we demonstrate can handle real-world biological systems by pitting it against the BioModels database. We present detailed analysis on two particular case-studies involving the ANIMO and SimBiology® tools.

Contributions

- Develops translation from SBML to Stochastic hybrid automata,
- develops alternative translation of ANIMO models to stochastic hybrid automata,
- Shows SHAs are useful for modelling biological systems.

Publication History The paper was submitted, accepted and published in the International Journal on Software Tools for Technology Transfer.

Paper F - Checking & Distributing Statistical Model Checking

Peter E. Bulychev; Alexandre David; Kim Guldstrand Larsen; Axel Legay; Danny Bøgsted Poulsen

Abstract In this paper we propose a general framework for distributed statistical model checking of networks of priced timed automata. The first contribution is a new algorithm to distribute sequential hypothesis testing without introducing bias in the results. The second contribution is an implementation of this algorithm in Uppaal. The major contribution is an experimental and analytical evaluation of the approach through case studies, including an analysis of the SMC algorithm itself.

Contributions

- Develops a distributed algorithm for SMC for use in Uppaal SMC,
- Verifies said algorithm using Uppaal SMC itself

Publication History The paper was accepted and presented at NASA Formal Methods NFM 4th International Symposium and was published NASA Formal Methods LNCS 7226.

Part II
Papers

Monitor-Based Statistical Model Checking of WMTL$_{\leq}$

Abstract *We present a novel approach and implementation for analysing weighted timed automata with respect to weighted metric temporal logic$_{\leq}$ (WMTL$_{\leq}$). Based on a stochastic semantics of weighted timed automata, we apply statistical model checking (SMC) to estimate and test probabilities of satisfaction with desired levels of confidence. Our approach consists in generation of deterministic monitors for formulae in weighted metric temporal logic$_{\leq}$ (WMTL$_{\leq}$), allowing for efficient SMC by run-time evaluation of a given formula. By necessity, the deterministic observers are in general approximate (over- or under-approximations), but are most often exact and experimentally tight. The technique is implemented in the new tool* Casaal *that we seamlessly connect to* Uppaal SMC *in a tool chain. We demonstrate the applicability of our technique and the efficiency of our implementation through a number of case-studies.*

1 Introduction

Model checking (MC) [39] is a widely used approach to guarantee correctness of a system by checking that its model satisfies a given property. A typical model checking algorithm explores the state space of a model and tries to prove or disprove the property holds on the model.

Despite a large and growing number of successful applications in industrial case studies, the MC approach still suffers from the state space explosion problem. This problem manifests itself in the form of unmanageable large state spaces of models with large number of parallel components or large number of variables. The situation is even worse when a system under analysis is hybrid, because a state space of such models may lack finite representation [5]. Another challenge for MC is to analyse stochastic systems, i.e. systems with probabilistic assumptions about their behaviour.

One of the ways to avoid these complexity and undecidability issues is to use statistical model checking (SMC) [123]. The main idea of SMC is to observe a number of simulations of a model and then use results from statistics (e.g.

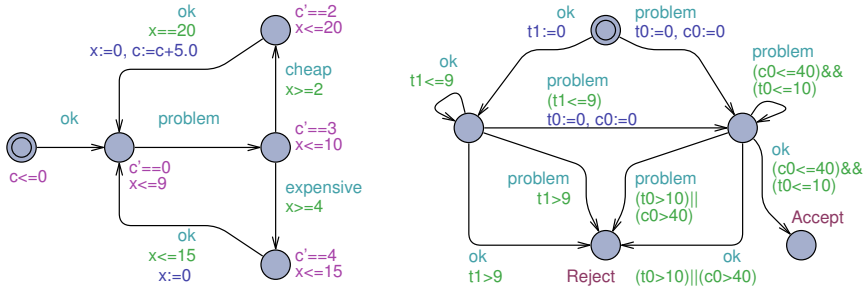


Figure A.1: A model (left) and deterministic monitor (right) for the repair problem

sequential analysis) to get an overall estimate of a system behaviour.

In the present paper we consider the problem of computing the probability that a random run of a given weighted timed automaton (WTA) satisfies a given weighted metric temporal logic$_{\leq}$ (WMTL$_{\leq}$). Solving this problem is of great practical interest since WTAs are as expressive as general linear hybrid automata [5], a formalism proved useful for modelling real-world hybrid and real-time systems. Moreover, WMTL$_{\leq}$ [28] is not only a weighted extension of the well established LTL but can also be seen as an extension of MTL [83] to hybrid systems. As the model checking problem for WMTL$_{\leq}$ is known to be undecidable [28], we propose an approximate approach that computes a confidence interval for the probability. In most of the cases this confidence interval can be made arbitrary small.

As an example consider a never-ending process of repairing problems [28], whose WTA model is depicted at Fig. A.1 (left). The repair of a problem has a certain cost, captured in the model by the clock `c`[1]. As soon as a problem occurs (modelled by the transition labelled by action `problem`) the value of `c` grows with rate 3, until actual `cheap` (rate 2) or `expensive` (rate 4) repair is taking place. Clock `x` grows with rate 1 (its default behaviour unless other rate is specified). Being a WTA, this model is equipped with a natural stochastic semantics [45] with a uniform choice on possible discrete transitions and uniformly selected delays in locations.

Now consider that we want to express the property that a path goes from `ok` back to itself in time less than 10 time units and cost less than 40. This can be formalised by the following WMTL$_{\leq}$ formula:

$$\texttt{ok}\,\mathsf{U}^{\tau}_{\leq 9}(\texttt{problem} \wedge (\neg\texttt{ok}\,\mathsf{U}^{\tau}_{\leq 10}\,\texttt{ok}) \wedge (\neg\texttt{ok}\,\mathsf{U}^{c}_{\leq 40}\,\texttt{ok}))$$

Here, the WMTL$_{\leq}$-formula $\varphi_1 \mathsf{U}^{c}_{\leq d} \varphi_2$ is satisfied by a run if φ_1 is satisfied on the run until φ_2 is satisfied, and this will happen before the value of the clock c increases with more than d starting from the beginning of the run (τ is a special clock that always grows with rate 1).

[1] we will (mis)use the term "clock" from timed automata, though in the setting of WTAs the clocks are really general real-valued variables

In order to estimate the probability that a random run of a model satisfies a given property, our approach first construct deterministic monitoring weighted timed automata for this property. In fact, it is not always possible to construct an *exact* deterministic observer for a property, thus our tool can result in deterministic under- and over-approximations. For our example, the tool constructed the exact deterministic monitor presented in Fig. A.1 (right). Here rates of a monitoring automaton are defined by the rates of the automaton being monitored, i.e. the rate of `c0` is equal to the rate of `c`.

The constructed monitoring WTA permits the SMC engine of Uppaal to use run-time evaluation of the property in order to efficiently estimate the probability that runs of the models satisfy the given property. In our example Uppaal SMC returns the 95% confidence interval $[0.215, 0.225]$. If none of the under- and over-approximation monitors are exact, then we use both of them to compute the confidence interval.

Our contribution is twofold. First, we are the first to extend statistical model checking to the $\text{WMTL}_{\leq}$ logic. The closest logic that has been studied so far is the strictly less expressive $MTL_{\leq}$, that does not allow using energy clocks in the U operator. Second, our monitor-based approach works on-the-fly and can terminate a simulation as soon as it may conclude that a formula is satisfied (or violated) by the simulation. Other statistical model checking algorithms that deal with linear-time properties (cf. [3, 107, 123, 124]) require a posterior (and expensive) check after a complete simulation of a fixed duration has been generated.

2 Weighted Timed Automata & Metric Temporal Logic

In this section we describe WTA and weighted metric temporal logic ($\text{WMTL}_{\leq}$) as our modelling and specification formalisms. A notion of monitoring weighted timed automaton (MWTA) is used to define automatically constructed (deterministic) observers for $\text{WMTL}_{\leq}$ properties.

2.1 Weighted Timed Automata

Let $\mathtt{X}$ be a set of clocks. A clock bound over $\mathtt{X}$ has the form $\mathtt{x} \bowtie n$ where $\mathtt{x} \in \mathtt{X}$, $\bowtie \in \{<, \leq, \geq, >\}$ and $n \in \mathbb{Z}_{\geq 0}$. We denote the set of all possible clock bounds over $\mathtt{X}$ by $\mathtt{B}(\mathtt{X})$. A valuation over $\mathtt{X}$ is a function $\mathtt{v} : \mathtt{X} \to \mathbb{R}_{\geq 0}$, and a rate vector is a function $r : \mathtt{X} \to \mathbb{Q}$. We let $\mathtt{V}(\mathtt{X})$ ($R(\mathtt{X})$, respectively) be all clock valuations (rates) over $\mathtt{X}$. If $d \in \mathbb{R}_{\geq 0}$, then we define $(\mathtt{v} + d)$ to be equal to the valuation $\mathtt{v}'$ such, that for all $\mathtt{x} \in \mathtt{X}$ we have $\mathtt{v}'(\mathtt{x}) = \mathtt{v}(\mathtt{x}) + d$. If r is a rate vector, then $(\mathtt{v} + r \cdot d)$ is the valuation $\mathtt{v}'$ such that for all clocks $\mathtt{x}$ in $\mathtt{X}$, $\mathtt{v}'(\mathtt{x}) = \mathtt{v}(\mathtt{x}) + r(\mathtt{x}) \cdot d$. The valuation assigning zero to all clocks is denoted $\vec{0}$. Given $\mathtt{Y} \subseteq \mathtt{X}$, $\mathtt{v}[\mathtt{Y} = 0]$ is the valuation equal to $\vec{0}$ over $\mathtt{Y}$ and equal to $\mathtt{v}$ over $\mathtt{X} \setminus \mathtt{Y}$.

Definition 23. A WTA[a] over alphabet Σ is a tuple $(\mathtt{L}, \mathtt{l}_0, \mathtt{X}^i, \mathtt{X}^o, \mathtt{E}, \mathtt{W}, \mathtt{I}, \mathtt{X}^{\mathtt{R}})$ where:

- $\mathtt{L}$ is a finite set of locations,
- $\mathtt{l}_0 \in \mathtt{L}$ is the initial location,
- $\mathtt{X}^i$ and $\mathtt{X}^o$ are finite set of real-valued variables called *internal* clocks and *observable* clocks, respectively,
- $\mathtt{E} \subseteq \mathtt{L} \times 2^{\mathtt{B}(\mathtt{X}^i \cup \mathtt{X}^o)} \times \Sigma \times 2^{\mathtt{X}^i} \times \mathtt{L}$ is a finite set of edges,
- $\mathtt{W} : \mathtt{E} \to R(\mathtt{X}^i \cup \mathtt{X}^o)$ assigns weights to edges, weights of observable clocks should be non-negative (i.e. $\mathtt{W}(e)(\mathtt{x}) \geq 0$ for any $e \in \mathtt{E}$ and $\mathtt{x} \in \mathtt{X}^o$),
- $\mathtt{I} : \mathtt{L} \to 2^{\mathtt{B}(\mathtt{X}^i \cup \mathtt{X}^o)}$ assigns an invariant to each location and
- $\mathtt{X}^{\mathtt{R}} : \mathtt{L} \to R(\mathtt{X}^i \cup \mathtt{X}^o)$ assigns rates to the clocks in each location, rates of observable clocks should be non-negative.

[a] In the classical notion of priced timed automata [10, 20] cost-variables (e.g. clocks where the rate may differ from 1) may not be referenced in guards, invariants or in resets, thus making e.g. optimal reachability decidable. This is in contrast to our notion of WTA, which is as expressive as linear hybrid systems [36].

We say, that a valuation $\mathtt{v}$ satisfies a clock bound $b = \mathtt{x} \bowtie n$ (denoted $\mathtt{v} \vDash b$), iff $\mathtt{v}(\mathtt{x}) \bowtie n$. A valuation satisfies a set of clock bounds if it satisfies all of them or this set is empty. A state $(\mathtt{l}, \mathtt{v})$ of a WTA consists of a location $\mathtt{l} \in \mathtt{L}$ and a valuation $\mathtt{v} \in \mathtt{V}(\mathtt{X}^i \cup \mathtt{X}^o)$. In particular, the initial state of the WTA is $(\mathtt{l}_0, \vec{0})$. From a state, a WTA can either delay for some time d or it can perform a discrete action a, the rules are given below:

- $(\mathtt{l}, \mathtt{v}) \xrightarrow{d} (\mathtt{l}, \mathtt{v}')$ if $\mathtt{v}' = (\mathtt{v} + \mathtt{X}^{\mathtt{R}}(\mathtt{l}) \cdot d)$ and $\mathtt{v}' \vDash \mathtt{I}(\mathtt{l})$.
- $(\mathtt{l}, \mathtt{v}) \xrightarrow{a} (\mathtt{l}', \mathtt{v}')$ if there exists an edge $e \in \mathtt{E}$ such that $e = (\mathtt{l}, g, a, \mathtt{Y}, \mathtt{l}')$, $\mathtt{v} \vDash g$, $\mathtt{v}' = (\mathtt{v}[\mathtt{Y} = 0] + \mathtt{W}(e) \cdot 1)$ and $\mathtt{v}' \vDash \mathtt{I}(\mathtt{l}')$.

Definition 24. An infinite weighted word over a finite set of propositions $\mathtt{AP}$ and clocks $\mathtt{X}$ is a sequence $\pi = (\mathtt{P}_0, \mathtt{v}_0)(\mathtt{P}_1, \mathtt{v}_1)\ldots$ where for all i, $\mathtt{P}_i \subseteq \mathtt{AP}$ and $\mathtt{v}_i \in \mathtt{V}(\mathtt{X})$.

For $i \geq 0$, we denote by $\pi^{i:}$ the weighted word $(\mathtt{P}_i, \mathtt{v}_i)(\mathtt{P}_{i+1}, \mathtt{v}_{i+1})\ldots$.

A WTA $S = (\mathtt{L}, \mathtt{l}_0, \mathtt{X}^i, \mathtt{X}^o, \mathtt{E}, \mathtt{W}, \mathtt{I}, \mathtt{X}^{\mathtt{R}})$ over Σ generates a weighted word $\pi = (\{a_0\}, \mathtt{v}_0)(\{a_1\}, \mathtt{v}_1)\ldots$ over Σ and $\mathtt{X}^o$, if there exists a sequence of transitions

$$(\mathtt{l}_0, \vec{0}) \xrightarrow{d_0} (\mathtt{l}_0, \mathtt{v}''_0) \xrightarrow{a_0} (\mathtt{l}_1, \mathtt{v}'_1) \xrightarrow{d_1} \ldots \xrightarrow{a_n} (\mathtt{l}_{n+1}, \mathtt{v}'_{n+1})\ldots,$$

s.t. for any i the valuation $\mathtt{v}_i$ is a projection of $\mathtt{v}''_i$ to $\mathtt{X}^o$, i.e. $\mathtt{v}_i(\mathtt{x})$ is equal to $\mathtt{v}''_i(\mathtt{x})$ for any observable clock $\mathtt{x} \in \mathtt{X}^o$. We let $\mathcal{L}(S)$ denote the set of all weighted words generated by an WTA S and refer to it as the language of S.

Note, that since *observable* clocks are never reset and grow only with positive rates, the values of observable clocks can not decrease in a word generated by a WTA. In fact, we restrict ourselves to WTAs that generate cost-divergent words (i.e. for any observable clock $\mathtt{x}$ and constant $k \in \mathbb{R}_{\geq 0}$ there is $\mathtt{v}_i$ such, that $\mathtt{v}_i(\mathtt{x}) > k$). If we consider that the WTA in Fig. A.1(left) has only one observable clock $\mathtt{c}$, then this WTA can generate a weighted word $(\mathtt{ok}, \{\mathtt{c} \mapsto 0.0\})$, $(\mathtt{problem}, \{\mathtt{c} \mapsto 0.0\})$, $(\mathtt{cheap}, \{\mathtt{c} \mapsto 10.2\}), \ldots$.

A network of Weighted Timed Automata is a parallel composition of several WTA with disjoint set of clocks and same set of actions Σ. The automata are synchronised regarding discrete transitions such that if one automaton performs a transition $\xrightarrow{a}$ all other also must perform a $\xrightarrow{a}$ transition. The notion of language recognised by WTA is naturally extended to the networks of Weighted Timed Automata.

In [45] we proposed a stochastic semantics for WTAs, i.e. a probability measure over the set of generated weighted words $\mathcal{L}(S)$. The non-determinism regarding discrete transitions for a single WTA is resolved using a uniform probabilistic choice among the possible transitions. Non-determinism regarding delays from a state $(\mathtt{l}, \mathtt{v})$ of a single WTA is resolved using a density function $\mu_{(\mathtt{l},\mathtt{v})}$ over delays in $\mathbb{R}_{\geq 0}$ being either a uniform or an exponential distribution depending on whether the invariant of $\mathtt{l}$ is empty or not.
The stochastic semantics for networks of WTAs is then given in terms of repeated races between the component WTAs of the network: before a discrete transition each acWTA chooses a delay according to its delay density function; then the WTA with the smallest delay wins the race and chooses probabilistically the action that the network must perform.

2.2 Monitoring Weighted Timed Automata

A monitoring weighted timed automaton (MWTA) S_M is used to define allowed behaviour of a system: a weighted word π over propositions $\mathtt{AP}$ and clocks $\mathtt{X}$ is fed as input to S_M. The run is read elementwise and S_M transits between its locations in response to each element. The clocks of S_M has a corresponding clock in π which allows S_M to measure the elapse of time with respect to a given clock of π. For accomplishing this, local clocks of S_M always grow with the same rate as their corrsponding clocks in π.

A literal is an element p or $\neg p$ with $p \in \mathtt{AP}$ and $\mathtt{AP}$ being a set of propositions. A conjunction over $\mathtt{AP}$ is a conjunction of literals. If α is a conjunction over $\mathtt{AP}$ and $\mathtt{P} \subseteq \mathtt{AP}$ then we define a satisfaction relation in the obvious manner and write $\mathtt{P} \vDash \alpha$. For a finite set of propositions $\mathtt{AP}$, we denote by $\mathtt{Conj}(\mathtt{AP})$ the set of all non-equivalent finite conjunctions over $\mathtt{AP}$. Also we denote by $\alpha_{\mathtt{tt}}$ the empty conjunction.

Definition 25. A monitoring weighted timed automaton over the clocks $\mathtt{X}$ and the propositions $\mathtt{AP}$ is a tuple $(\mathtt{L}, \mathtt{l}_0, \mathtt{l}_a, \mathtt{X}^{\mathtt{m}}, \mathtt{E}, \mathtt{m})$ where:

- $\mathtt{L}$ is a finite set of locations,
- $\mathtt{l}_0 \in \mathtt{L}$ is the initial location,
- $\mathtt{l}_a \in \mathtt{L}$ is the accepting location,
- $\mathtt{X}^{\mathtt{m}}$ is a finite set of *local* clocks,
- $\mathtt{E} \subseteq \mathtt{L} \times 2^{\mathtt{B}(\mathtt{X}^{\mathtt{m}})} \times \mathtt{Conj}(\mathtt{AP}) \times 2^{\mathtt{X}^{\mathtt{m}}} \times \mathtt{L}$ is a finite set of edges and
- $\mathtt{m} : \mathtt{X}^{\mathtt{m}} \to \mathtt{X}$ gives the correspondence of local clocks and $\mathtt{X}$.

An MWTA is called deterministic if for any location $\mathtt{l} \in \mathtt{L}$, set of propositions $\mathtt{P}$ and valuation $\mathtt{v} \in \mathtt{V}(\mathtt{X}^{\mathtt{m}})$ there exists only one edge $(\mathtt{l}, g, \alpha, \mathtt{Y}, \mathtt{l}') \in \mathtt{E}$ such that $\mathtt{v} \vDash g$ and $\mathtt{P} \vDash \alpha$.

An MWTA $S_M = (\mathtt{L}, \mathtt{l}_0, \mathtt{l}_a, \mathtt{X}^{\mathtt{m}}, \mathtt{E}, \mathtt{m})$ over clocks $\mathtt{X}$ and propositions $\mathtt{AP}$ accepts a weighted word $(\mathtt{P}_0, \mathtt{v}_0)(\mathtt{P}_1, \mathtt{v}_1)\ldots$ over the same $\mathtt{X}$ and $\mathtt{AP}$, iff there exists a finite sequence of states $(\mathtt{l}_0, \nu_0)(\mathtt{l}_1, \nu_1)\ldots(\mathtt{l}_n, \nu_n)$ of states of S_M such that:

- $\nu_0(\mathtt{x}) = \vec{0}$,
- for any $i < n$ there exists an edge $(\mathtt{l}_i, g_i, \alpha_i, \mathtt{Y}_i, \mathtt{l}_{i+1}) \in \mathtt{E}$ such, that:
  - $\nu_i \vDash g_i$,
  - $\mathtt{P}_i \vDash \alpha_i$,
  - $\nu_{i+1}(\mathtt{x}) = (\mathtt{v}_{i+1}(\mathtt{m}(\mathtt{x})) - \mathtt{v}_i(\mathtt{m}(\mathtt{x})))$ if $\mathtt{x} \in \mathtt{Y}_i$ and $\nu_{i+1}(\mathtt{x}) = \nu_i(\mathtt{x}) + (\mathtt{v}_{i+1}(\mathtt{m}(\mathtt{x})) - \mathtt{v}_i(\mathtt{m}(\mathtt{x})))$ otherwise
- $l_n = l_a$ is the accepting location of S_M.

Thus, after reading an element of an input weighted word, a *local* clock $\mathtt{x}$ of the MWTA grows with the same rate as the corresponding clock $(\mathtt{x})$ in the input word.

2.3 Weighted Metric Temporal logic WMTL$_{\leq}$

Definition 26. A WMTL$_{\leq}$ formula φ over atomic propositions $\mathtt{AP}$ and clocks $\mathtt{X}$ is defined by the grammar

$$\varphi ::= p \mid \neg p \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d} \varphi_2 \mid \varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d} \varphi_2$$

where $p \in \mathtt{AP}$, $d \in \mathbb{N}$, and $\mathtt{x} \in \mathtt{X}$.

Remark 6. We restrict WMTL$_{\leq}$ formulas to be in negative normal form. We need this for our monitoring technique and it provides no limitations as any standard WMTL$_{\leq}$ formula can be transformed to negative normal form.

Let $\mathtt{ff}$ be an abbreviation for $(p \wedge \neg p)$, and $\mathtt{tt}$ be an abbreviation for $\neg p \vee p$. The other commonly used operators in WMTL$_{\leq}$ can be defined by the following abbreviations: $\Diamond^{\mathtt{x}}_{\leq d}\varphi = \mathtt{tt}\,\mathsf{U}^{\mathtt{x}}_{\leq d}\varphi$, $\Box^{\mathtt{x}}_{\leq d}\varphi = \mathtt{ff}\,\mathsf{R}^{\mathtt{x}}_{\leq d}\varphi$. We assume that there always exists a special clock $\tau \in \mathtt{X}$ (that grows with rate 1).

For a given weighted word $\pi = (\mathtt{P}_0, \mathtt{v}_0)(\mathtt{P}_1, \mathtt{v}_1)(\mathtt{P}_2, \mathtt{v}_2)\ldots$ over $\mathtt{AP}$ and $\mathtt{X}$ and WMTL$_{\leq}$ formula φ over $\mathtt{AP}$ and $\mathtt{X}$, the satisfaction relation $\pi^{i:} \models \varphi$ is defined inductively:

1. $\pi^{i:} \vDash p$ iff $p \in \mathtt{P}_i$
2. $\pi^{i:} \vDash \neg p$ iff $p \notin \mathtt{P}_i$
3. $\pi^{i:} \vDash \varphi_1 \wedge \varphi_2$ iff $\pi^{i:} \models \varphi_1$ and $\pi^{i:} \models \varphi_2$
4. $\pi^{i:} \vDash \varphi_1 \vee \varphi_2$ iff $\pi^{i:} \models \varphi_1$ or $\pi^{i:} \models \varphi_2$
5. $\pi^{i:} \vDash \mathsf{X}\varphi$ iff $\pi^{i+1:} \models \varphi$
6. $\pi^{i:} \vDash \varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d}\varphi_2$ iff there exists j such that $j \geq i$, $\pi^{j:} \models \varphi_2$, $\mathtt{v}_j(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) \leq d$, and for all k, $i \leq k < j$, $\pi^{k:} \models \varphi_1$.
7. $\pi^{i:} \vDash \varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d}\varphi_2$ if for all j where $\mathtt{v}_j(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) \leq d$ $\pi^{j:} \vDash \varphi_2$ or there exists a k, $i \leq k < j$, where $\pi^{k:} \vDash \varphi_1$.

We say, that a weighted word π satisfies φ, iff $\pi^{0:} \vDash \varphi$, and denote by $\mathcal{L}(\varphi)$ the set of all weighted words satisfied by φ. Formulas φ_1 and φ_2 are equivalent if they are satisfied by the same weighted words, in which case we write $\varphi_1 \equiv \varphi_2$.

Given the stochastic semantics of a WTA S, and semantics of WMTL$_{\leq}$ formula φ, we can define $\mathbb{P}_S(\varphi)$ to be the probability that a random run of S satisfies φ. This probability is well-defined because $\mathcal{L}(S) \cap \mathcal{L}(\varphi)$ is a countable union and intersection of measurable sets and thus it is measurable itself.

3 From Formulas to Monitors

In this section we present a novel procedure for translating WMTL$_{\leq}$ formulas into equivalent MWTA monitors, providing an essential and efficient component of our tool-chain. However, to enable monitor-based SMC it is essential that the generated MWTA is deterministic. Unfortunately, this might not always be possible as there are WMTL$_{\leq}$ formulas for which no equivalent deterministic MWTA exists[2]. As a remedy, we describe how basic syntactic transformations prior to translation allow us to obtain deterministic over- and under-approximating MWTAs for any given formula φ. In Section 5, we shall see that these approximations are tight and often exact.

[2]For instance, $\Diamond^{\tau}_{\leq 1}(p \wedge \Box^{\tau}_{\leq 1}(\neg r) \wedge \Diamond^{\tau}_{\leq 1}(q))$ is an example of a formula not equivalent to any deterministic MWTA.

3.1 Closures & Extended Formulas

In this section, we assume that φ is a WMTL$_{\leq}$-formula over propositions $\mathtt{AP}$ and (observable) clocks $\mathtt{X}$. We use $\mathrm{Sub}(\varphi)$ to denote all the sub-formulas of φ.

In order to further expand φ into a disjunctive normal form, we introduce for each $\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d}\varphi_2 \in \mathrm{Sub}(\varphi)$ and each $\varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d}\varphi_2 \in \mathrm{Sub}(\varphi)$, one local clock $\mathtt{y}$ and two clock bounds $\mathtt{y} \leq d$ and $\mathtt{y} > d$ to express some timing information related to $\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d}\varphi_2$ and $\varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d}\varphi_2$. Also, we introduce auxiliary formulas $\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d-\mathtt{y}}\varphi_2$ and $\varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d-\mathtt{y}}\varphi_2$ to express some requirements that should be satisfied in the future when we try to guarantee $\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d}\varphi_2 \in \mathrm{Sub}(\varphi)$ or $\varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d}\varphi_2 \in \mathrm{Sub}(\varphi)$ is true in the current state.

We define $\mathtt{X}_\varphi = \{\mathtt{y}_{\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d}\varphi_2} | \varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d}\varphi_2 \in \mathrm{Sub}(\varphi)\} \cup \{\mathtt{y}_{\varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d}\varphi_2} | \varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d}\varphi_2 \in \mathrm{Sub}(\varphi)\}$ to be the set of all local clocks for φ, where $\mathtt{y}_{\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d}\varphi_2}$ is the clock assigned to $\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d}\varphi_2$ and $\mathtt{y}_{\varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d}\varphi_2}$ is the local clock assigned to $\varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d}\varphi_2$. We call $\mathtt{y}_{\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d}\varphi_2}$ a local clock of $\mathsf{U}_{\leq}$-type, $\mathtt{y}_{\varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d}\varphi_2}$ a local clock of $\mathsf{R}_{\leq}$-type and denote all $\mathsf{U}_{\leq}$ clocks by $\mathtt{X}^{\mathsf{U}}_{\varphi}$ and all $\mathsf{R}_{\leq}$ clock by $\mathtt{X}^{\mathsf{R}}_{\varphi}$. The mapping $\mathtt{m}$ from local clocks $\mathtt{X}_\varphi$ to observable clocks $\mathtt{X}$ is defined by $\mathtt{m}(\mathtt{y}_{\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d}\varphi_2}) = \mathtt{x}$ and $\mathtt{m}(\mathtt{y}_{\varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d}\varphi_2}) = \mathtt{x}$. The closure of φ, written as $\mathrm{CL}(\varphi)$, is now defined by the following rules:

1. $\mathtt{tt} \in \mathrm{CL}(\varphi)$, $\mathrm{Sub}(\varphi) \subseteq \mathrm{CL}(\varphi)$
2. If $\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d}\varphi_2 \in \mathrm{Sub}(\varphi)$ and $\mathtt{y} \in \mathtt{X}_\varphi$ is the local clock assigned to $\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d}\varphi_2$, then $\mathtt{y} \leq d$, $\mathtt{y} > d$, $\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d-\mathtt{y}}\varphi_2 \in \mathrm{CL}(\varphi)$,
3. If $\varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d}\varphi_2 \in \mathrm{Sub}(\varphi)$ and $\mathtt{y} \in \mathtt{X}_\varphi$ is the local clock assigned to $\varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d}\varphi_2$, then $\mathtt{y} \leq d$, $\mathtt{y} > d$, $\varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d-\mathtt{y}}\varphi_2 \in \mathrm{CL}(\varphi)$,
4. If φ_1, $\varphi_2 \in \mathrm{CL}(\varphi)$, then $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2 \in \mathrm{CL}(\varphi)$

Obviously, $\mathrm{CL}(\varphi)$ has only finitely many non-equivalent formulas.

For a local clock $\mathtt{y}$, we use $\mathsf{rst}(\mathtt{y})$ to represent that $\mathtt{y}$ will be reset at current step and $\mathsf{unch}(\mathtt{y})$ to represent that $\mathtt{y}$ will not be reset at current step. The set of extended formulas for φ, written as $\mathrm{Ext}(\varphi)$, is now defined by the following rules:

1. If $\psi \in \mathrm{CL}(\varphi)$, then ψ, $\mathsf{X}\psi \in \mathrm{Ext}(\varphi)$
2. If $\mathtt{y} \in \mathtt{X}^{\mathsf{U}}_{\varphi}$ then $\mathsf{unch}(\mathtt{y}) \in \mathrm{Ext}(\varphi)$
3. If $\mathtt{y} \in \mathtt{X}^{\mathsf{R}}_{\varphi}$ then $\mathsf{rst}(\mathtt{y}) \in \mathrm{Ext}(\varphi)$
4. If φ_1, $\varphi_2 \in \mathrm{Ext}(\varphi)$, then $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2 \in \mathrm{Ext}(\varphi)$

Extended formulas can be interpreted using extended weighted words. An *extended weighted word* $\pi = (\mathtt{P}_0, \mathtt{v}_0, \nu_0)(\mathtt{P}_1, \mathtt{v}_1, \nu_1)(\mathtt{P}_2, \mathtt{v}_2, \nu_2)\ldots$ is a sequence where $(\mathtt{P}_0, \mathtt{v}_0)(\mathtt{P}_1, \mathtt{v}_1)(\mathtt{P}_2, \mathtt{v}_2)\ldots$ is a weighted word over $\mathtt{AP}$ and $\mathtt{X}$, and for

every $i \in \mathbb{N}$, ν_i is a clock valuation over $\mathtt{X}_\varphi$ such that for all $\mathtt{y} \in \mathtt{X}_\varphi$, either $\nu_{i+1}(\mathtt{y}) = \mathtt{v}_{i+1}(\mathtt{m}(\mathtt{y})) - \mathtt{v}_i(\mathtt{m}(\mathtt{y}))$ or $\nu_{i+1}(\mathtt{y}) = \nu_i(\mathtt{y}) + \mathtt{v}_{i+1}(\mathtt{m}(\mathtt{y})) - \mathtt{v}_i(\mathtt{m}(\mathtt{y}))$.

The semantics for extended formulas is naturally induced by the semantics of WMTL$_\leq$ formulas:

Definition 27. Let $\pi = (\mathtt{P}_0, \mathtt{v}_0, \nu_0)(\mathtt{P}_1, \mathtt{v}_1, \nu_1)(\mathtt{P}_2, \mathtt{v}_2, \nu_2)\ldots$ be an extended weighted word and $\Phi \in \mathrm{Ext}(\varphi)$. The satisfaction relation $\pi^{i:} \vDash_e \Phi$ is inductively defined as follows:

1. $\pi^{i:} \vDash_e \mathtt{y} \leq d$ iff $\nu_i(\mathtt{y}) \leq d$
2. $\pi^{i:} \vDash_e \mathtt{y} > d$ iff $\nu_i(\mathtt{y}) > d$
3. $\pi^{i:} \vDash_e \mathsf{rst}(\mathtt{y})$ iff $\nu_{i+1}(\mathtt{y}) = \mathtt{v}_{i+1}(\mathtt{m}(\mathtt{y})) - \mathtt{v}_i(\mathtt{m}(\mathtt{y}))$
4. $\pi^{i:} \vDash_e \mathsf{unch}(\mathtt{y})$ iff $\nu_{i+1}(\mathtt{y}) = \nu_i(\mathtt{y}) + \mathtt{v}_{i+1}(\mathtt{m}(\mathtt{y})) - \mathtt{v}_i(\mathtt{m}(\mathtt{y}))$
5. $\pi^{i:} \vDash_e \varphi$ iff $\pi^{i:} \vDash \varphi$ and $\varphi \in \mathrm{Sub}(\varphi)$
6. $\pi^{i:} \vDash_e \varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d-\mathtt{y}} \varphi_2$ iff there exists j such that $j \geq i$, $\pi^{j:} \models \varphi_2$, $\mathtt{v}_j(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) \leq d - \nu_i(\mathtt{y})$, $\pi^{k:} \models \varphi_1$ for all k with $i \leq k < j$
7. $\pi^{i:} \vDash_e \varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d-\mathtt{y}} \varphi_2$ iff for all $j \geq i$ such that $\mathtt{v}_j(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) \leq d - \nu_i(\mathtt{y})$, either $\pi^{j:} \models \varphi_2$ or there exists k with $i \leq k < j$ and $\pi^{k:} \models \varphi_1$
8. $\pi^{i:} \vDash_e \varphi_1 \wedge \varphi_2$ iff $\pi^{i:} \vDash_e \varphi_1$ and $\pi^{i:} \vDash_e \varphi_2$
9. $\pi^{i:} \vDash_e \varphi_1 \vee \varphi_2$ iff $\pi^{i:} \vDash_e \varphi_1$ or $\pi^{i:} \vDash_e \varphi_2$
10. $\pi^{i:} \vDash_e \mathsf{X}\Phi$ iff $\pi^{i+1:} \vDash_e \Phi$

$\pi^{i:}$ is a *model* of φ if $\pi^{i:} \vDash_e \varphi$ and two extended WMTL$_\leq$-formulas are said *equivalent* if they have exactly the same models.

3.2 Constructing Non-deterministic Monitors

As in the construction of Büchi automata from LTL formulas, we break a formula into a disjunction of several conjunctions [43]. Each of the disjuncts corresponds to a transition of a resulting observer automaton and specifies the requirements to be satisfied in the current and in the next states. In the rest of this section, we use $\mathsf{rst}(\{\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_n\})$ and $\mathsf{unch}(\{\mathtt{y}_1, \mathtt{y}_2, \ldots, \mathtt{y}_n\})$ to denote the formula of $\mathsf{rst}(\mathtt{x}_1) \wedge \mathsf{rst}(\mathtt{x}_2) \wedge \ldots \wedge \mathsf{rst}(\mathtt{x}_n)$ and the formula of $\mathsf{unch}(\mathtt{y}_1) \wedge \mathsf{unch}(\mathtt{y}_2) \wedge \ldots \wedge \mathsf{unch}(\mathtt{y}_n)$ respectively. A *basic conjunction* is an extended formula of the form:

$$\alpha \wedge g \wedge \mathsf{rst}(\mathtt{X}) \wedge \mathsf{unch}(\mathtt{Y}) \wedge \mathsf{X}(\Psi),$$

where $\alpha \in \mathtt{Conj}(\mathtt{AP})$, g is a conjunction of clock bounds, $\mathtt{X}$ is a set of local clocks of R$_\leq$-type, $\mathtt{Y}$ is a set of local clocks of U$_\leq$-type, and Ψ is a formula in $\mathrm{CL}(\varphi)$. The $\alpha \wedge g \wedge \mathsf{rst}(\mathtt{X}) \wedge \mathsf{unch}(\mathtt{Y})$ part of a basic conjunction specifies requirements

to be satisfied in the current state and Ψ specifies the requirements in the next state. Note that we will often consider the conjunction of clock bounds, g, as a set of clock bounds.

Definition 28. Let φ be a WMTL$_{\leq}$ formula. Then we define the function $\mathtt{Rew} : \mathrm{CL}(\varphi) \to \mathrm{Ext}(\varphi)$ recursively as follows:

- $\mathtt{Rew}(p) = p$
- $\mathtt{Rew}(\neg p) = \neg p$
- $\mathtt{Rew}(\mathtt{y} \bowtie d) = \mathtt{y} \bowtie d$ where $\bowtie \in \{\leq, >\}$
- $\mathtt{Rew}(f \wedge g) = \mathtt{Rew}(f) \wedge \mathtt{Rew}(g)$
- $\mathtt{Rew}(f \vee g) = \mathtt{Rew}(f) \vee \mathtt{Rew}(g)$
- $\mathtt{Rew}(\mathsf{X}\varphi_1) = \mathsf{X}\varphi_1$
- $\mathtt{Rew}(\varphi_1 \,\mathsf{U}^{\mathtt{x}}_{\leq d}\, \varphi_2) = \mathtt{Rew}(\varphi_2) \vee (\mathtt{Rew}(\varphi_1) \wedge \mathsf{X}((\mathtt{y} \leq d) \wedge (\varphi_1 \,\mathsf{U}^{\mathtt{x}}_{\leq d-\mathtt{y}} \varphi_2)))$, where $\mathtt{y}$ is the clock assigned to $\varphi_1 \,\mathsf{U}^{\mathtt{x}}_{\leq d}\, \varphi_2$
- $\mathtt{Rew}(\varphi_1 \,\mathsf{U}^{\mathtt{x}}_{\leq d-\mathtt{y}}\, \varphi_2) = \mathtt{Rew}(\varphi_2) \vee (\mathtt{Rew}(\varphi_1) \wedge \mathsf{unch}(\mathtt{y}) \wedge \mathsf{X}((\mathtt{y} \leq d) \wedge (\varphi_1 \,\mathsf{U}^{\mathtt{x}}_{\leq d-\mathtt{y}} \varphi_2)))$
- $\mathtt{Rew}(\varphi_1 \,\mathsf{R}^{\mathtt{x}}_{\leq d}\, \varphi_2) = \mathtt{Rew}(\varphi_2) \wedge (\mathtt{Rew}(\varphi_1) \vee (\mathsf{rst}(\mathtt{y}) \wedge \mathsf{X}(((\mathtt{y} \leq d) \wedge (\varphi_1 \,\mathsf{R}^{\mathtt{x}}_{\leq d-\mathtt{y}} \varphi_2)) \vee (\mathtt{y} > d))))$, where $\mathtt{y}$ is the clock assigned to $\varphi_1 \,\mathsf{R}^{\mathtt{x}}_{\leq d}\, g$
- $\mathtt{Rew}(\varphi_1 \,\mathsf{R}^{\mathtt{x}}_{\leq d-\mathtt{y}}\, \varphi_2) = \mathtt{Rew}(\varphi_2) \wedge (\mathtt{Rew}(\varphi_1) \vee \mathsf{X}(((\mathtt{y} \leq d) \wedge (\varphi_1 \,\mathsf{R}^{\mathtt{x}}_{\leq d-\mathtt{y}} \varphi_2)) \vee (\mathtt{y} > d)))$

Lemma 3. Let $\pi = (\mathtt{P}_0, \mathtt{v}_0, \nu_0)(\mathtt{P}_1, \mathtt{v}_1, \nu_1)(\mathtt{P}_2, \mathtt{v}_2, \nu_2)\ldots$ be an extended weighted word and $\Phi \in \mathrm{CL}(\varphi)$. If $\pi^{i:} \vDash_e \mathtt{Rew}(\Phi)$ then $\pi^{i:} \vDash_e \Phi$.

Proof. Proof by induction in the structure of Φ. Let $\Phi =$

- p
  Trivial since $\mathtt{Rew}(p) = p$.
- $\neg p$
  Trivial since $\mathtt{Rew}(\neg p) = \neg p$.
- $\mathtt{y} \bowtie d$
  Trivial since $\mathtt{Rew}(\mathtt{y} \bowtie d) = \mathtt{y} \bowtie d$
- $\varphi_1 \wedge \varphi_2$
  Then $\mathtt{Rew}(\Phi) = \mathtt{Rew}(\varphi_1) \wedge \mathtt{Rew}(\varphi_2)$ and we need to show that $\pi^{i:} \vDash_e \varphi_1 \wedge \varphi_2$. Since $\pi^{i:} \vDash_e \mathtt{Rew}(\varphi_1)$ and $\pi^{i:} \vDash_e \mathtt{Rew}(\varphi_2)$ then by induction hypothesis $\pi^{i:} \vDash_e \varphi_1$ and $\pi^{i:} \vDash_e \varphi_2$ thus $\pi^{i:} \vDash_e \varphi_1 \wedge \varphi_2$.

- $\varphi_1 \vee \varphi_2$
  Similar to above

- $\mathsf{X}\varphi_1$
  Trivial since $\mathtt{Rew}(\mathsf{X}\varphi_1) = \mathsf{X}\varphi_1$.

- $\varphi_1 \,\mathsf{U}^{\mathtt{x}}_{\leq d}\, \varphi_2$
  Then $\mathtt{Rew}(\Phi) = \mathtt{Rew}(\varphi_2) \vee (\mathtt{Rew}(\varphi_1) \wedge \mathsf{X}((\mathtt{y} \leq d) \wedge (\varphi_1 \,\mathsf{U}^{\mathtt{x}}_{\leq d-\mathtt{y}} \varphi_2)))$. Since $\pi^{i:} \vDash_e \mathtt{Rew}(\Phi)$ then either:

  1. $\pi^{i:} \vDash_e \mathtt{Rew}(\varphi_2)$ and by induction hypothesis then $\pi^{i:} \vDash_e \varphi_2$ and so $\pi^{i:} \vDash_e \varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d} \varphi_2$, or
  2. $\pi^{i:} \nvDash_e \mathtt{Rew}(\varphi_2)$. Then by the fact that $\pi^{i:} \vDash_e \mathtt{Rew}(\Phi)$ we know that $\pi^{i:} \vDash \mathtt{Rew}(\varphi_1) \wedge \mathsf{X}((\mathtt{y} \leq d) \wedge (\varphi_1 \,\mathsf{U}^{\mathtt{x}}_{\leq d-\mathtt{y}} \varphi_2))$ thus $\pi^{i:} \vDash_e \mathtt{Rew}(\varphi_1)$ implying $\pi^{i:} \vDash_e \varphi_1$. It is not too difficult to see that $\pi^{i:} \vDash_e \varphi_1$ combined with that $\pi^{i:} \vDash_e \mathsf{X}((\mathtt{y} \leq d) \wedge (\varphi_1 \,\mathsf{U}^{\mathtt{x}}_{\leq d-\mathtt{y}} \varphi_2))$ implies that $\pi^{i:} \vDash_e \varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d} \varphi_2$.

- remaining cases are omitted. They are quite similar to the above.

□

Definition 29. Let $\pi = (\mathtt{P}_0, \mathtt{v}_0)(\mathtt{P}_1, \mathtt{v}_1)(\mathtt{P}_2, \mathtt{v}_2)\ldots$ be a weighted word and φ be WMTL$_{\leq}$ formula. We define an extended weighted word $\bar{\pi} = (\mathtt{P}_0, \mathtt{v}_0, \nu_0)(\mathtt{P}_1, \mathtt{v}_1, \nu_1)$ with $\nu_0 = \vec{0}$ and for all $i > 0$:

1. If $\mathtt{y}$ is the clock assigned to $(\varphi_1 \,\mathsf{U}^{\mathtt{x}}_{\leq d}\, \varphi_2) \in \mathrm{Sub}(\varphi)$, then

$$\nu_{i+1}(\mathtt{y}) = \begin{cases} \nu_i(\mathtt{y}) + \mathtt{v}_{i+1}(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) & \text{if } \nu_i(\mathtt{y}) \leq d, \pi^{i:} \vDash \varphi_1 \,\mathsf{U}^{\mathtt{x}}_{d-\nu_i(\mathtt{y})}\, \varphi_2 \\ & \text{and } \pi^{\nvDash:} \varphi_2 \\ \mathtt{v}_{i+1}(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) & \text{otherwise} \end{cases}$$

2. If $\mathtt{y}$ is the clock assigned to $(\varphi_1 \,\mathsf{R}^{\mathtt{x}}_{\leq d}\, \varphi_2) \in \mathrm{Sub}(\varphi)$ then

$$\nu_{i+1}(\mathtt{y}) = \begin{cases} \mathtt{v}_{i+1}(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) & \text{if } \pi^{i:} \vDash \varphi_1 \,\mathsf{R}^{\mathtt{x}}_{\leq d}\, \varphi_2 \text{ and } \pi^{i:} \nvDash \varphi_1 \\ \nu_i(\mathtt{y}) + \mathtt{v}_{i+1}(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) & \text{otherwise} \end{cases}$$

Lemma 4. Let $\pi = (\mathtt{P}_0, \mathtt{v}_0)(\mathtt{P}_1, \mathtt{v}_1)(\mathtt{P}_2, \mathtt{v}_2)\ldots$ be a weighted word and $\Phi \in \mathrm{CL}(\varphi)$. If $\bar{\pi}^{i:} \vDash_e \Phi$ then $\bar{\pi}^{i:} \vDash_e \mathtt{Rew}(\Phi)$

Proof. Induction in the structure of Φ. Let Φ be

- p.
  Trivial as $\mathtt{Rew}(p) = p$.

- $\neg p$.
  Trivial as $\texttt{Rew}(\neg p) = \neg p$.

- $\mathtt{y} \bowtie d$
  Trivial since $\texttt{Rew}(\mathtt{y} \bowtie d) = \mathtt{y} \bowtie d$

- $\varphi_1 \wedge \varphi_2$
  As $\bar{\pi}^{i:} \vDash \Phi$ then $\bar{\pi}^{i:} \vDash \varphi_1$ and $\bar{\pi}^{i:} \vDash \varphi_2$. By our induction hypothesis $\bar{\pi}^{i:} \vDash_e \texttt{Rew}(\varphi_1)$ and $\bar{\pi}^{i:} \vDash_e \texttt{Rew}(\varphi_2)$ thus $\bar{\pi}^{i:} \vDash_e \texttt{Rew}(\varphi_1) \wedge \texttt{Rew}(\varphi_2)$.

- $\varphi_1 \vee \varphi_2$
  Similar to above

- $\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d} \varphi_2$.
  Then $\texttt{Rew}(\Phi) = \texttt{Rew}(\varphi_2) \vee (\texttt{Rew}(\varphi_1) \wedge \mathsf{X}((\mathtt{y} \leq d) \wedge (\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d - \mathtt{y}} \varphi_2)))$ thus we need to argue either that $\bar{\pi}^{i:} \vDash_e \texttt{Rew}(\varphi_2)$ or that $\bar{\pi}^{i:} \vDash_e (\texttt{Rew}(\varphi_1) \wedge \mathsf{X}((\mathtt{y} \leq d) \wedge (\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d - \mathtt{y}} \varphi_2)))$

  Since $\bar{\pi}^{i:} \vDash_e \varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d} \varphi_2$ then either

  1. $\bar{\pi}^{i:} \vDash_e \varphi_2$.
     Then by our induction hypothesis $\bar{\pi}^{i:} \vDash_e \texttt{Rew}(\varphi_2)$, or
  2. $\bar{\pi}^{i:} \nvDash_e \varphi_2$.
     Then there exists $j > i$ s.t. $\mathtt{v}_j(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) \leq d$, $\bar{\pi}^{j:} \vDash_e \varphi_2$ and for all $k, i \leq k < j$, $\bar{\pi}^{k:} \vDash_e \varphi_1$. By induction hypothesis then $\bar{\pi}^{i:} \vDash_e \texttt{Rew}(\varphi_1)$ thus now we only need to argue that $\bar{\pi}^{i:} \vDash_e \mathsf{X}((\mathtt{y} \leq d) \wedge (\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d - \mathtt{y}} \varphi_2)))$ amounting to show that $\bar{\pi}^{i+1:} \vDash_e (\mathtt{y} \leq d)$ and that $\bar{\pi}^{i+1:} (\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d - \mathtt{y}} \varphi_2)$. To conclude this we need to consider the different values for $\nu_{i+1}(\mathtt{y})$. Let

     - $\nu_{i+1}(\mathtt{y}) = \mathtt{v}_{i+1}(\mathtt{x}) - \mathtt{v}_i(\mathtt{x})$
       Recall that $d \geq \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) \geq \mathtt{v}_{i+1} - \mathtt{v}_i(\mathtt{x}) = \nu_{i+1}(\mathtt{y})$ thus clearly $\bar{\pi}^{i+1:} \vDash_e (\mathtt{y} \leq d)$. For showing that $\bar{\pi}^{i+1:} \vDash_e \varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d - \mathtt{y}} \varphi_2$ we just need to show that $\mathtt{v}_j(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) \leq d - \nu_{i+1}(\mathtt{y})$ as the remaining requirements are implied from $\bar{\pi}^{i:} \vDash_e \varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d} \varphi_2$. Consider that

       $$\begin{aligned} & \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) + \nu_{i+1}(\mathtt{y}) \\ =& \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) + \mathtt{v}_{i+1}(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) \\ =& \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) \leq d \end{aligned}$$

       and thus

       $$\begin{aligned} & \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) + \nu_{i+1}(\mathtt{y}) \leq d \\ \Leftrightarrow & \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) \leq d - \nu_{i+1}(\mathtt{y}) \end{aligned}$$

– $\nu_{i+1}(\mathtt{y}) = \nu_i(\mathtt{y}) + \mathtt{v}_{i+1}(\mathtt{x}) - \mathtt{v}_i(\mathtt{x})$
Then by definition of $\bar{\pi}$, $\nu_i(\mathtt{y}) \leq d$ and $\pi^{i:} \vDash \varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d-\nu_i(\mathtt{y})} \varphi_2$. Then it must be the case that $\mathtt{v}_j(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) \leq d - \nu_i(\mathtt{y})$ thus $d \geq \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) + \nu_i(\mathtt{y}) \geq \mathtt{v}_{i+1}(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) + \nu_i(\mathtt{y}) = \nu_{i+1}(\mathtt{y})$ hence clearly $\bar{\pi}^{i+1:} \vDash_e (\mathtt{y} \leq d)$.
For showing that $\bar{\pi}^{i+1:} \vDash_e \varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d-\mathtt{y}} \varphi_2$ we just need to show that $\mathtt{v}_j(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) \leq d - \nu_{i+1}(\mathtt{y})$ as the remaining requirements are implied from $\bar{\pi}^{i:} \vDash_e \varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d} \varphi_2$. Consider now that $\nu_{i+1}(\mathtt{y}) + \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) = \nu_i(\mathtt{y}) + \mathtt{v}_{i+1}(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) + \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) = \nu_i(\mathtt{y}) - \mathtt{v}_i(\mathtt{x}) + \mathtt{v}_j(\mathtt{x}) \leq d \Leftrightarrow \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) \leq d - \nu_{i+1}(\mathtt{y})$

- Remaining cases are omitted as they are quite similar to the above.

□

The next Lemma 5 and main Theorem 2 provides the construction of a monitor from a formula.

Lemma 5. Let $\varphi \in \mathrm{CL}(\Phi)$. Then $\mathtt{Rew}(\varphi)$ can be transformed into a disjunction of several basic conjunctions.

Proof. Structural induction in the structure of φ. Let φ be

- p.
Since $\mathtt{Rew}(p) = p = p \wedge \mathsf{rst}(\emptyset) \wedge \mathsf{unch}(\emptyset) \wedge \mathsf{X}(\mathtt{tt})$ the lemma holds.

- $\neg p$.
Since $\mathtt{Rew}(\neg p) = \neg p = (\neg p \wedge \mathsf{rst}(\emptyset) \wedge \mathsf{unch}(\emptyset) \wedge \mathsf{X}(\mathtt{tt}))$ the lemma holds.

- $\mathtt{x} \bowtie n$.
Since $\mathtt{Rew}(\mathtt{x} \bowtie n) = \mathtt{x} \bowtie n = \alpha_{\mathtt{tt}} \wedge (\mathtt{x} \bowtie n) \wedge \mathsf{rst}(\emptyset) \wedge \mathsf{unch}(\emptyset)\mathsf{X}(\mathtt{tt}))$ where $\alpha_{\mathtt{tt}}$ denotes the empty conjunction of propositions the lemma holds.

- $\varphi_1 \wedge \varphi_2$.
Then by induction hypothesis $\mathtt{Rew}(\varphi_1) = \bigvee_{i=1}^{k_1} (\alpha_i^1 \wedge g_i^1 \wedge \mathsf{rst}(\mathtt{X}_i^1) \wedge \mathsf{unch}(\mathtt{Y}_i^1 \wedge \mathsf{X}(\psi_i^1)$ for some k_1 and $\mathtt{Rew}(\varphi_2) = \bigvee_{j=1}^{k_2} (\alpha_j^2 \wedge g_j^2 \wedge \mathsf{rst}(\mathtt{X}_j^2) \wedge \mathsf{unch}(\mathtt{Y}_j^2 \wedge \mathsf{X}(\psi_j^2)$ for some k_2

  Since

  $$\mathtt{Rew}(\varphi_1 \wedge \varphi_2) = \mathtt{Rew}(\varphi_1) \wedge \mathtt{Rew}(\varphi_2) =$$
  $$(\vee_{i=1}^{k_1} (\alpha_i^1 \wedge g_i^1 \wedge \mathsf{rst}(\mathtt{X}_i^1) \wedge \mathsf{unch}(\mathtt{Y}_i^1) \wedge \mathsf{X}(\psi_i^1)) \bigwedge$$
  $$(\vee_{j=1}^{k_2} (\alpha_j^2 \wedge g_j^2 \wedge \mathsf{rst}(\mathtt{X}_j^2) \wedge \mathsf{unch}(\mathtt{Y}_j^2) \wedge \mathsf{X}(\psi_j^2) =$$
  $$\bigvee_{i=1}^{k_1} \vee_{j=1}^{k_2} (\alpha_i^1 \wedge \alpha_j^2) \wedge (g_i^1 \wedge_j^2) \wedge \mathsf{rst}(\mathtt{X}_i^1 \cup \mathtt{X}_j^2) \wedge \mathsf{unch}(\mathtt{Y}_i^1 \cup \mathtt{Y}_j^2) \wedge \mathsf{X}(\psi_i^1 \wedge \psi_j^2),$$

  the lemma holds

- Remaining cases are similar to the above.

□

Theorem 2. Let φ be a WMTL$_{\leq}$-formula over the propositions $\mathtt{AP}$ and the clocks $\mathtt{X}$. Let the MWTA $S_{\varphi}= (\mathtt{L}, \mathtt{l}_0, \mathtt{l}_a, \mathtt{X}^{\mathtt{m}}, \mathtt{E}, \mathtt{m})$ over the clocks $\mathtt{X}$ and the actions $\Sigma = 2^{\mathtt{AP}}$ be defined as follows:

- $\mathtt{L} = \{\mathtt{l}_\psi \mid \psi \in \mathrm{CL}(\varphi)\ \}$ is a finite set of locations;
- $\mathtt{l}_0 = \mathtt{l}_\varphi$ is the initial location;
- $\mathtt{l}_a = \mathtt{l}_{\mathtt{tt}}$ is the accepting location;
- $\mathtt{X}^{\mathtt{m}} = \mathtt{X}_\varphi$ is the set of all local clocks,
- $(\mathtt{l}_\psi, g, \alpha, r, \mathtt{l}_{\psi'}) \in \mathtt{E}$ if $\alpha \wedge g \wedge \mathsf{rst}(\mathtt{X}) \wedge \mathsf{unch}(\mathtt{Y}) \wedge \mathsf{X}(\psi')$ is a basic conjunction of ψ, and that $\mathtt{X} \subseteq r \subseteq (\mathtt{X}^{\mathtt{m}} \setminus \mathtt{Y})$ and
- $\mathtt{m}$ is defined by $\mathtt{m}(\mathtt{y}_{\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d} \varphi_2}) = \mathtt{x}$ and $\mathtt{m}(\mathtt{y}_{\varphi_1 \mathsf{R}^{\mathtt{x}}_{\leq d} \varphi_2}) = \mathtt{x}$.

Then $\mathcal{L}(\varphi) = \mathcal{L}(S_\varphi)$.

Proof.

- $\mathcal{L}(\varphi) \subseteq \mathcal{L}(S_\varphi)$
  Let $\pi \in \mathcal{L}(\varphi)$ and let $\bar{\pi} = (\mathtt{P}_0, \mathtt{v}_0, \nu_0)(\mathtt{P}_1, \mathtt{v}_1, \nu_1)(\mathtt{P}_2, \mathtt{v}_2, \nu_2) \ldots$ be as defined in Definition 29 then by $\pi \in \mathcal{L}(\varphi)$ we know that $\bar{\pi} \vDash_e \varphi$.

  By structural induction in φ we now show that if $\bar{\pi}^{0:} \vDash_e \varphi$ then there exists accepting sequence of states $(\mathtt{l}_\varphi, \nu_0)(\mathtt{l}_{\psi_1}, \nu_1) \ldots (\mathtt{l}_{\psi_{m-1}}, \nu_{m-1})(\mathtt{l}_{\mathtt{tt}}, \nu_m)$ for some formulas $\psi_1, \psi_2 \ldots \psi_{m-1}$ and some m. Let φ be

  - p.
    Then a basic conjunction of $\mathtt{Rew}(\varphi)$ is $p \wedge \mathsf{rst}(\emptyset) \wedge \mathsf{unch}(\emptyset) \wedge \mathsf{X}(\mathtt{tt})$. Let $r = \{\mathtt{y} \in \mathtt{X}^{\mathtt{m}} \mid \nu_1(\mathtt{y}) = \mathtt{v}_1(\mathtt{m}(\mathtt{y})) - \mathtt{v}_0(\mathtt{m}(\mathtt{y}))\}$. Since $\emptyset \subseteq r \subseteq \mathtt{X}^{\mathtt{m}} \setminus \emptyset$ there exists an edge $(\mathtt{l}_\varphi, \emptyset, p, r, \mathtt{l}_{\mathtt{tt}})$ and thus $(\mathtt{l}_\varphi, \nu_0)(\mathtt{l}_{\mathtt{tt}}, \nu_1)$ is an accepting sequence of S_φ.
  - $\neg p$.
    Similar to above
  - $\mathtt{x} \bowtie n$.
    Similar to above
  - $\varphi_1 \wedge \varphi_2$.
    Then $\bar{\pi} \vDash \varphi_1$ and $\bar{\pi} \vDash \varphi_2$ and thus by induction hypothesis there exist accepting sequences of states

    $$(\mathtt{l}_{\varphi_1}, \nu_0)(\mathtt{l}_{\varphi_1^1}, \nu_1) \ldots (\mathtt{l}_{\varphi_1^{n-1}}, \nu_{n-1})(\mathtt{l}_{\mathtt{tt}}, \nu_n)$$

and

$$(\mathtt{l}_{\varphi_2}, \nu_0)(\mathtt{l}_{\varphi_2^1}, \nu_1) \dots (\mathtt{l}_{\varphi_2^{k-1}}, \nu_{k-1})(\mathtt{l}_{\mathtt{tt}}, \nu_k),$$

for some k and n. Assume without loss of generality that $n < k$, then it can be shown by induction in n that there exists a sequence

$$(\mathtt{l}_{\varphi_1 \wedge \mathtt{l}_{\varphi_2}}, \nu_0)(\mathtt{l}_{\varphi_1^1 \wedge \varphi_2^1}, \nu_1) \dots (\mathtt{l}_{\varphi_1^{n-1} \wedge \varphi_2^{n-1}}, \nu_{n-1})(\mathtt{l}_{\varphi_2^n}, \nu_n)$$

of states of S_φ.

Thus an accepting sequence of states is

$$(\mathtt{l}_{\varphi_1 \wedge \mathtt{l}_{\varphi_2}}, \nu_0)(\mathtt{l}_{\varphi_1^1 \wedge \varphi_2^1}, \nu_1) \dots (\mathtt{l}_{\varphi_1^{n-1} \wedge \varphi_2^{n-1}}, \nu_{n-1})(\mathtt{l}_{\varphi_2^n}, \nu_n)$$
$$(\mathtt{l}_{\varphi_2^{n+1}}, \nu_{n+1}) \dots (\mathtt{l}_{\varphi_2^{k-1}}, \nu_{k-1})(\mathtt{l}_{\mathtt{tt}}, \nu_n)$$

– $\varphi_1 \vee \varphi_2$.
Assume without loos of generality that $\pi \vDash_e \varphi_1$ then by induction hypothesis there exists an accepting sequence of states

$$(\mathtt{l}_{\varphi_1}, \nu_0)(\mathtt{l}_{\varphi_1^1}, \nu_1) \dots (\mathtt{l}_{\varphi_1^{n-1}}, \nu_{n-1})(\mathtt{l}_{\mathtt{tt}}, \nu_n)$$

and a basic conjunction $\alpha \wedge \mathsf{rst}(\mathtt{X}) \wedge \mathsf{unch}(\mathtt{Y}) \wedge \mathsf{X}(\varphi_1^1)$ of $\mathtt{Rew}(\varphi_1)$. Since $\alpha \wedge \mathsf{rst}(\mathtt{X}) \wedge \mathsf{unch}(\mathtt{Y}) \wedge \mathsf{X}(\varphi_1^1)$ is also a basic conjunction of $\mathtt{Rew}(\varphi)$ clearly

$$(\mathtt{l}_{\varphi_1 \vee \varphi_2}, \nu_0)(\mathtt{l}_{\varphi_1^1}, \nu_1) \dots (\mathtt{l}_{\varphi_1^{n-1}}, \nu_{n-1})(\mathtt{l}_{\mathtt{tt}}, \nu_n)$$

is a valid sequence of states of S_φ.

– $\varphi_1 \mathsf{U}^{\mathtt{x}}_{\leq d} \varphi_2$.
Then either

* $\bar{\pi} \vDash_e \varphi_2$.
  Using similar reasoning as in the $\vee$ case it is very easy to construct an accepting sequence using the induction hypothesis.
* there exists an n where $\bar{\pi}^{n:} \vDash_e \varphi_2$ and for all $j < n$, $\bar{\pi}^{j:} \vDash_e \varphi_1$ and $\mathtt{v}_j(\mathtt{x}) - \mathtt{v}_0(\mathtt{x}) \leq d$.

  Firstly consider that since $\bar{\pi}^{0:} \vDash_e \varphi$, then $\bar{\pi}^{0:} \vDash_e \mathtt{Rew}(\varphi)$ and consequently a basic conjunction of $\mathtt{Rew}(\varphi)$ is satisfied. As $\mathtt{Rew}(\varphi) = \mathtt{Rew}(\varphi_2) \vee (\mathtt{Rew}(\varphi_1) \wedge \mathsf{X}((\mathtt{y} \leq d) \wedge \varphi_1 \mathsf{U}^{\mathtt{y}}_{d-\mathtt{y}} \varphi_2))$ and $\bar{\pi}^{0:} \nvDash_e \varphi_2$, clearly $\bar{\pi}^{0:} \vDash_e (\mathtt{Rew}(\varphi_1) \wedge \mathsf{X}((\mathtt{y} \leq d) \wedge \varphi_1 \mathsf{U}^{\mathtt{y}}_{d-\mathtt{y}} \varphi_2)))$. Then it must be the case that a basic conjunction of $\mathtt{Rew}(\varphi_1)$ is true. Let that be

  $$\alpha_{\varphi_1} \wedge g_{\varphi_1} \wedge \mathsf{rst}(\mathtt{X}_{\varphi_1}) \wedge \mathsf{unch}(\mathtt{Y}_{\varphi_1}) \wedge \mathsf{X}(\psi_1').$$

A basic conjunction of $\mathtt{Rew}(\varphi)$ which is satisfied is then

$$\alpha_{\varphi_1} \wedge g_{\varphi_1} \wedge \mathsf{rst}(\mathtt{X}_{\varphi_1}) \wedge \mathsf{unch}(\mathtt{Y}_{\varphi_1}) \wedge \mathsf{X}(\psi_1' \wedge (\mathtt{y} \leq d) \wedge (\varphi_1 \mathsf{U}^{\mathtt{x}}_{d-\mathtt{y}} \varphi_2))$$

and thus the state $(\mathtt{l}_{\psi_1' \wedge (\mathtt{y} \leq d) \wedge (\varphi_1 \mathsf{U}^{\mathtt{x}}_{d-\mathtt{y}} \varphi_2)}, \nu_1)$ is reachable.

We now show that there exists a sequence of states

$$(\mathtt{l}_{\psi_1}, \nu_1) \dots (\mathtt{l}_{\psi_n^n}, \nu_n),$$

where for all $i \geq 1$, $\psi_i = \psi_i' \wedge (\mathtt{y} \leq d) \wedge (\varphi_1 \mathsf{U}^{\mathtt{x}}_{d-\mathtt{y}} \varphi_2)$ for some formula ψ_i' and $\bar{\pi}^{i:} \vDash_e \psi_i$.

base case, $i = 1$

Trivial.

Inductive step, i<n

Since $\bar{\pi}^{i:} \vDash_e \psi_i$ then $\bar{\pi}^{i:} \vDash_e \psi_i'$ and thus one basic conjunction of $\mathtt{Rew}(\psi_i')$, say $\alpha_{\psi_i'} \wedge g_{\psi_i'} \wedge \mathsf{rst}(\mathtt{X}_{\psi_i'}) \wedge \mathsf{unch}(\mathtt{Y}_{\psi_i'}) \wedge \mathsf{X}(\psi_i'')$, must be satisfied. Also, since $i \neq n$ we know $\bar{\pi}^{i:} \vDash_e \varphi_1$ and hence one basic conjunction of $\mathtt{Rew}(\varphi_1)$, say $\alpha_{\varphi_1} \wedge g_{\varphi_1} \wedge \mathsf{rst}(\mathtt{X}_{\varphi_1}) \wedge \mathsf{unch}(\mathtt{Y}_{\varphi_1}) \wedge \mathsf{X}(\varphi_1')$ is satisfied.

A basic conjunction of $\mathtt{Rew}(\psi_i)$ is

$$\alpha_{\psi_i'} \wedge \alpha_{\varphi_1} \wedge (\mathtt{y} \leq d) \wedge g_{\psi_i'} \wedge g_{\varphi_1} \wedge \mathsf{rst}(\mathtt{X}_{\psi_i'} \cup \mathtt{X}_{\varphi_1}) \wedge \mathsf{unch}(\mathtt{Y}_{\psi_i'} \cup \mathtt{Y}_{\varphi_1} \cup \{\mathtt{y}\}) \wedge \mathsf{X}(\psi_i'' \wedge \varphi_1' \wedge (\mathtt{y} \leq d) \wedge \varphi_1 \mathsf{U}^{\mathtt{x}}_{d-\mathtt{y}} \varphi_2).$$

thus clearly the state $(\mathtt{l}_{\psi_i'' \wedge \varphi_1' \wedge (\mathtt{y} \leq d) \wedge \varphi_1 \mathsf{U}^{\mathtt{x}}_{d-\mathtt{y}} \varphi_2)}, \nu_{i+1})$ is reachable in S_φ.

In $(\mathtt{l}_{\psi_n}, \nu_n)$ of S_φ we know that $\bar{\pi}^{n:} \vDash_e \varphi_2$ and we know that $\psi_n = \psi_n' \wedge (\mathtt{y} \leq d) \wedge (\varphi_1 \mathsf{U}^{\mathtt{x}}_{d-\mathtt{y}} \varphi_2)$ for some ψ_n'. By definition

$$\begin{aligned}\mathtt{Rew}(\psi_n) =& \mathtt{Rew}(\psi_n') \wedge (\mathtt{y} \leq d) \wedge (\mathtt{Rew}(\varphi_2) \vee (\mathtt{Rew}(\varphi_1) \wedge \mathsf{unch}(\{\mathtt{y}\}) \wedge \\ & \mathsf{X}((\mathtt{y} \leq d) \wedge (\varphi_1 \mathsf{U}^{\mathtt{x}}_{d-\mathtt{y}} \varphi_2)))) \\ =& (\mathtt{Rew}(\psi_n') \wedge (\mathtt{y} \leq d) \wedge \mathtt{Rew}(\varphi_2)) \vee \\ & (\mathtt{Rew}(\psi_n') \wedge (\mathtt{y} \leq d) \wedge \mathtt{Rew}(\varphi_1) \wedge \mathsf{unch}(\mathtt{y}) \wedge \\ & \mathsf{X}((\mathtt{y} \leq d) \wedge (\varphi_1 \mathsf{U}^{\mathtt{x}}_{d-\mathtt{y}} \varphi_2))).\end{aligned}$$

Using arguments as in the $\wedge$ case it can easily be seen that $\bar{\pi}^{n:} \vDash_e \mathtt{Rew}(\psi_n') \wedge ((\mathtt{y} \leq d) \wedge \mathtt{Rew}(\varphi_2)$ implies there exists an accepting sequence from $(\mathtt{l}_{\psi_n}, \nu_n)$.

– Remaining cases omitted.

- $\mathcal{L}(S_\varphi) \subseteq \mathcal{L}(\varphi)$
  Let $\pi = (\mathtt{P}_0, \mathtt{v}_0)(\mathtt{P}_1, \mathtt{v}_1)(\mathtt{P}_2, \mathtt{v}_2) \dots$ and let $\pi \in \mathcal{L}(S_\varphi)$. Then there exists

edges $(\mathtt{l}_{\psi_0}, g_0, \alpha_0, r_0, \mathtt{l}_{\psi_1})(\mathtt{l}_{\psi_1}, g_1, \alpha_1, r_1, \mathtt{l}_{\psi_2}) \dots (\mathtt{l}_{\psi_{n-1}}, g_{n-1}, \alpha_{n-1}, r_{n-1}, \mathtt{l}_{\psi_n})$ with $\psi_0 = \varphi$, $\psi_n = \mathtt{tt}$ and valuations $\nu_0, \nu_1, \dots, \nu_n$ such that $\nu_0 = \vec{0}$ and for all i, $0 < i < n$, there exists a basic conjunction $\alpha_i \wedge g_i \wedge \mathsf{rst}(\mathtt{X}_i) \wedge \mathsf{unch}(\mathtt{Y}_i) \wedge \mathsf{X}(\psi_{i+1})$ of $\mathtt{Rew}(\psi_i)$ where $\nu_i \vDash g_i$, $\mathtt{P}_i \vDash \alpha_i$, $\mathtt{X}_i \subseteq r_i \subseteq \mathtt{X}^{\mathtt{m}} \setminus \mathtt{Y}_i$ and for all $\mathtt{y} \in \mathtt{X}$, $\nu_{i+1}(\mathtt{y}) = \nu_i[r_i = 0](\mathtt{y}) + \mathtt{v}_{i+1}(\mathtt{m}(\mathtt{y})) - \mathtt{v}_i(\mathtt{m}(\mathtt{y}))$.

We now consider the extended weighted word $\vec{\pi} = (\mathtt{P}_0, \mathtt{v}_0, \nu_0)(\mathtt{P}_1, \mathtt{v}_1, \nu_1)(\mathtt{P}_2, \mathtt{v}_2, \nu_2) \dots$ and show that for all k, $\vec{\pi}^{k:} \vDash_e \psi_k$. We do induction in the distance to $\mathtt{l}_{\psi_n}$ and use as induction hypothesis that for some i, $\vec{\pi}^{n-i:} \vDash_e \psi_{n-i}$ and use this to show that $\vec{\pi}^{n-(i+1):} \vDash_e \psi_{n-(i+1)}$.

Base case, $i = 0$ Since $\psi_{n-0} = \psi_n = \mathtt{tt}$ this is trivially true.

Inductive step We need to show that $\vec{\pi}^{n-(i+1):} \vDash_e \psi_{n-(i+1)}$ which amounts to showing that one basic conjunct of $\mathtt{Rew}(\psi_{n-(i+1)})$ is satisfied. One basic conjunct of $\mathtt{Rew}(\psi_{n-(i+1)})$ is

$$\alpha_{n-(i+1)} \wedge g_{n-(i+1)} \wedge \mathsf{rst}(\mathtt{X}_{n-(i+1)}) \wedge \mathsf{unch}(\mathtt{Y}_{n-(i+1)}) \wedge \mathsf{X}(\psi_{n-(i)}).$$

Since there was en edge in the automaton clearly $\mathtt{P}_{n-(i+1)} \vDash \alpha_{n-(i+1)}$ and equivalently we know $\nu_{n-(i+1)} \vDash g_{n-(i+1)}$. By induction hypothesis we know that $\vec{\pi}^{n-i:} \vDash_e \psi_{n-i}$ thus we only need to prove that the $\mathsf{rst}(\mathtt{X}_{n-(i+1)})$ and $\mathsf{unch}(\mathtt{Y}_{n-(i+1)})$ subformulae are satisfied. By definition of S_φ then for any $\mathtt{x} \in \mathtt{X}_{n-(i+1)}$, $\nu_{n-i}(\mathtt{x}) = \mathtt{v}_{n-i}(\mathtt{m}(\mathtt{x})) - \mathtt{v}_{n-(i+1)}(\mathtt{m}(\mathtt{x}))$. This is the requirement for satisfying $\mathsf{rst}(\mathtt{X}_{n-(i+1)})$. Similarly by definition we know that for any clock in $\mathtt{x} \in \mathtt{Y}_{n-(i+1)}$ $\nu_{n-i}(\mathtt{x}) = \nu_{n-(i+1)}(\mathtt{x}) + \mathtt{v}_{n-i}(\mathtt{m}(\mathtt{x})) - \mathtt{v}_{n-(i+1)}(\mathtt{m}(\mathtt{x}))$.

□

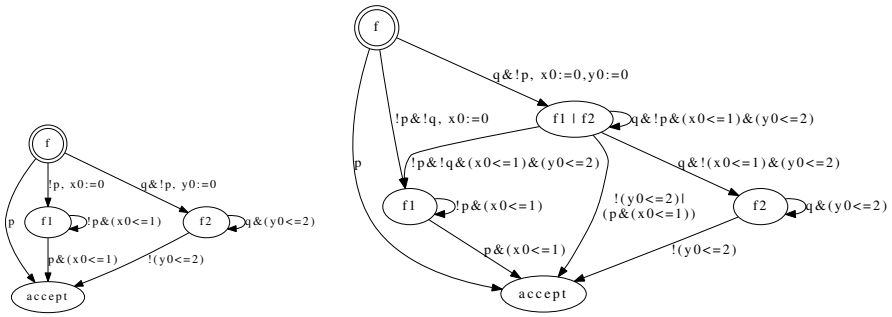
Example 10. Fig.A.2a is a MWTA obtained with our approach for $f = (\Diamond^x_{\leq 1} p) \vee (\Box^y_{\leq 2} q) = (\mathsf{tt}\, \mathsf{U}^x_{\leq 1} p) \vee (\mathsf{ff}\, \mathsf{R}^y_{\leq 2} q)$.

3.3 Constructing Deterministic Monitors

The construction of section 3.2 might produce non-deterministic automata. In fact, as stated earlier, there exist $\mathrm{WMTL}_{\leq}$ formulas for which no equivalent deterministic MWTA exists. To get deterministic MWTA for $\mathrm{WMTL}_{\leq}$-formulas, we further translate formulas in disjunctive form into the following *deterministic* form by repeated use of the logical equivalence $p \Leftrightarrow (p \wedge q) \vee (p \wedge \neg q)$.

$$F = \bigvee_{i=1}^{n} \left(\alpha_i \wedge g_i \wedge \bigvee_{k=1}^{m_i} (\mathsf{rst}(\mathtt{X}_{ik}) \wedge \mathsf{unch}(\mathtt{Y}_{ik}) \wedge \mathsf{X}(\Psi_{ik})) \right)$$

where for all $i \in \{1, \dots, n\}$: m_i is a positive integer, $\mathtt{X}_{ik} \subseteq \mathtt{X}_\varphi$ is a set of local clocks of $\mathsf{R}_{\leq}$-type and $\mathtt{Y}_{ik} \subseteq \mathtt{X}_\varphi$ is a set of local clocks of $\mathsf{U}_{\leq}$-type, and for all $i \neq j$: $\alpha_i \wedge g_i \wedge \alpha_j \wedge g_j$ is ff.



(a) Non-deterministic monitor (b) Deterministic under-approximation monitor

Figure A.2: Monitoring weighted timed automaton for $f \equiv (\Diamond^{x}_{\leq 1} p) \vee (\Box^{y}_{\leq 2} q)$, with
$f1 \equiv (x_0 \leq 1) \wedge (\mathsf{ttU}^{x}_{\leq 1-x_0} p)$ and $f2 \equiv ((y_0 \leq 2) \wedge (\mathsf{ffR}^{y}_{\leq 2-y_0} q)) \vee (y_0 > 2)$.

Using the facts that X distributes over $\vee$, and $\mathsf{rst}(X)$ and $\mathsf{unch}(X)$ are monotonic in X, the following formulas are obviously strengthened (F^u) respectively weakened (F^o) versions of F:

$$F^u = \bigvee_{i=1}^{n} \left(\alpha_i \wedge g_i \wedge \mathsf{rst}\left(\bigcup_{k=1}^{m_i} \mathtt{X}_{ik} \right) \wedge \mathsf{unch}\left(\bigcup_{k=1}^{m_i} \mathtt{Y}_{ik} \right) \wedge \mathsf{X}\left(\bigvee_{k=1}^{m_i} \Psi_{ik} \right) \right)$$

$$F^o = \bigvee_{i=1}^{n} \left(\alpha_i \wedge g_i \wedge \mathsf{rst}\left(\bigcap_{k=1}^{m_i} \mathtt{X}_{ik} \right) \wedge \mathsf{unch}\left(\bigcap_{k=1}^{m_i} \mathtt{Y}_{ik} \right) \wedge \mathsf{X}\left(\bigvee_{k=1}^{m_i} \Psi_{ik} \right) \right)$$

Interestingly, by simply applying the construction of Theorem 2 to F^u (F^o) we immediately obtain a deterministic under-approximating (over-approximating) MWTA S^u_φ (S^o_φ) for φ. Moreover, if during the construction of S^u_φ we see that F^u is always semantically equivalent to F, then S^u_φ is an exact determinisation of φ, i.e. $\mathcal{L}(S^u_\varphi) = \mathcal{L}(\varphi)$ (the same is true for over-approximation).

Example 11. (continued) Fig.A.2b is the under-approximation deterministic MWTA for $f = (\Diamond^{x}_{\leq 1} p) \vee (\Box^{y}_{\leq 2} q)$.

4 The Tool Chain

Figure A.3 provides an architectural view of our tool chain. The tool chain takes as input a WMTL$_{\leq}$ formula φ, a WTA model S, as well as statistical parameters ϵ, α for controlling precision and confidence level. As a result a confidence interval for the probability $\mathbb{P}_S(\varphi)$ with the desired precision and confidence level is returned.

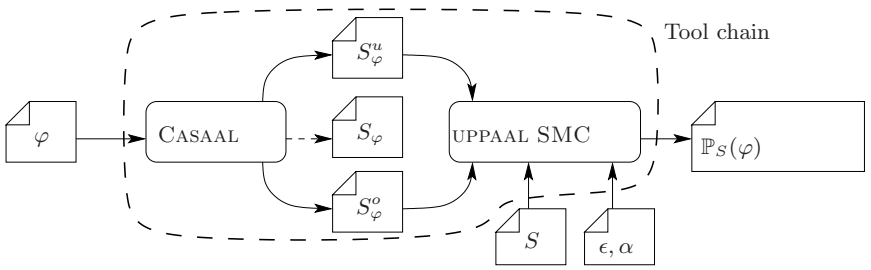


Figure A.3: Tool chain architecture

Casaal The tool chain includes the new tool component Casaal for generating monitors. The tool is implemented in C++ and is build on top of the Spot[3] open-source library for LTL to Büchi automata translation. We also use Buddy[4] BDD package to handle operations over Boolean formulas. Given a WMTL$_\leq$ formula φ, Casaal may construct an exact MWTA S_φ, as well as two – possibly approximating – MWTAs, S^u_φ and S^o_φ. The tool also reports if one of these approximations is exact (i.e. recognises exactly the language of φ). Table A.1 demonstrates some experimental results for Casaal. The formulas were also used in [61] and for comparison we list their results as well.

formula	automaton	states	trans	time(s)
$p\mathsf{U}^\tau_{\leq 1}(q\mathsf{U}^\tau_{\leq 1}(r\mathsf{U}^\tau_{\leq 1}s))$	nondet	5	14	0.02
	under	9	58	0.02
	over	9	56	0.04
	Geilen	14	30	
$(p \to \diamond^\tau_{\leq 5} q)\mathsf{U}^\tau_{\leq 100}\Box^\tau_{\leq 5}\neg p$	nondet	7	19	0.01
	under	9	32	0.01
	over	9	32	0.01
	Geilen	21	64	
$(((p\mathsf{U}^\tau_{\leq 4}q)\mathsf{U}^\tau_{\leq 3}r)\mathsf{U}^\tau_{\leq 2}s)\mathsf{U}^\tau_{\leq 1}t)$	nondet	17	121	0.02
	under	17	121	0.03
	over	17	121	0.03
	Geilen	60	271	

Table A.1: Experimental results for WMTL$_\leq$ formulas.

Uppaal SMC [45, 46] is a tool that allows estimating and test $\mathbb{P}_S(\phi)$, i.e. the probability that a random run of a given WTA model M satisfies ϕ, where ϕ is a WMTL$_\leq$ formula restricted to the form $\Diamond^c_{\leq d}\psi$ and ψ is a state predicate. Estimation is performed by generating a number of random simulations of S, where each simulation stops when either it reaches a state when ψ is satisfied, or $c \leq d$ is violated.

Combining Casaal and Uppaal SMC Casaal can generate monitors for general weighted words thus if $\mathtt{AP}$ is the set of propositions and $\mathtt{X}$ the set of clocks then a weighted run is $\pi = (\mathtt{P}_0, \mathtt{v}_0)(\mathtt{P}_1, \mathtt{v}_1)\ldots$ where for all i, $\mathtt{P}_i \subseteq \mathtt{AP}$

[3] http://spot.lip6.fr/wiki/
[4] http://sourceforge.net/projects/buddy/develop

and $\mathtt{v}_i \in \mathtt{V}(\mathtt{X})$. As a result of this, the edges of the created monitors may contain labels of the form $p \wedge q$ and thus require two propositions to be true simultaneously. The weighted word generated by a network of WTAs over the actions Σ is however simpler as here, for all i, $\mathtt{P}_i \in \{\{a\} | a \in \Sigma\}$ and thus only one proposition is true at any given time in these weighted words. With this in mind, an edge $\mathtt{l}_\psi \xrightarrow{g,\alpha,r} \mathtt{l}_{\psi'}$ where $\alpha = p \wedge q$ can clearly be removed from the generated monitor as p and q can never be satisfied simultaneously. Also, an edge $\mathtt{l}_\psi \xrightarrow{g,\alpha,r} \mathtt{l}_{\psi'}$ where $\alpha = \neg p_1 \wedge \neg p_2 \ldots \neg p_n$ can be transformed into a set of edges $\{\mathtt{l}_\psi \xrightarrow{g,q,r} \mathtt{l}_{\psi'} | q \in \Sigma \setminus \{p_1, p_2, \ldots p_n\}\}$. These transformations ensures the MWTA can be synchronised with a WTA.

Given the above transformation let us describe how we use Uppaal SMC together with Casaal to estimate the probability that a random run of a WTA model S satisfies a general WMTL$_{\leq}$ property φ, i.e. $\mathbb{P}_S(\varphi)$.

Let us first assume, that one of two deterministic approximations for φ returned by Casaal is exact. This means, that we have MWTA $S_\varphi^{det} = (\mathtt{L}, \mathtt{l}_0, \mathtt{l}_a, \mathtt{X}^{\mathtt{m}}, \mathtt{E}, m)$ such that $\mathcal{L}(S_\varphi^{det}) = \mathcal{L}(\varphi)$. First, we turn S_φ^{det} into input-enabled automaton by introducing a *rejecting* location $\mathtt{l}_r$ and adding complementary transitions to $\mathtt{l}_r$ from all other locations. Then we augment MWTA S_φ^{det} with a clock $c^\dagger$ that will grow with rate 1 in rejecting location $\mathtt{l}_r$, and with rate 0 in all other locations. Additionally, for every clock $c \in \mathtt{X}^{\mathtt{m}}$ we duplicate all rates and transition weights from the corresponding clock $m(c)$ to make sure, that the clocks of S_φ^{det} grow with the same rate as the corresponding clocks of the automaton S being monitored. Forming a parallel composition of S and S_φ^{det}, we may now use Uppaal SMC to estimate the probability $p = \mathbb{P}_{S||S_\varphi^{det}}(\Diamond_{\leq 1}^{c^\dagger}(\mathtt{l}_a))$. This can be done because of the following theorem:

Theorem 3. If S produces cost-divergent runs only, then each simulation of $S||S_\varphi^{det}$ will end up in accepting or rejecting location of S_φ^{det} after finite number of steps.

If none of the two MWTAs S_φ^o and S_φ^u are exact determinisation of S_φ (i.e. $\mathcal{L}(A_\varphi^u) \subsetneq \mathcal{L}(\varphi) \subsetneq \mathcal{L}(A_\varphi^o)$), then we use both of them to compute upper (using S_φ^o) and lower (using S_φ^u) bounds for $\mathbb{P}_S(\varphi)$. Indeed, if n_1 (n_2, correspondingly) out of m random simulations of $S||S_\varphi^u$ ($S||S_\varphi^o$, correspondingly) ended in accepting location l_a^u (l_a^o, correspondingly), then with significance level of α we can accept a hypothesis H_1 (H_2, correspondingly) that $\mathbb{P}_S(\varphi) \geq n_1/m - \varepsilon$ ($\mathbb{P}_S(\varphi) \leq n_2/m + \varepsilon$). By combining hypothesis H_1 and H_2 we can obtain a confidence interval $[n_1/m - \varepsilon, n_2/m + \varepsilon]$ for $\mathbb{P}_S(\varphi)$ with significance level of $1 - (1-\alpha)^2 = 2\alpha - \alpha^2$.

5 Case Studies

We performed several case studies to demonstrate the applicability of our tool chain. In the first case study we analyse the performance of Casaal on a set of randomly generated WMTL$_{\leq}$ formulae. In the second case study we

use a model of a robot moving on a two-dimensional grid, this model was first analysed in [14] using the manually constructed monitoring timed automaton.

5.1 Automatically Generated Formulae

In the first case study we analyse the performance of Casaal on a set of randomly generated WMTL$_{\leq}$ formulae. We generated 1000 formulae with 2, 3 and 4 actions, and created deterministic over and approximations for these formulae. Each of the formulae have 15 connectives (release, until, conjunction or disjunction) and four clocks.

For the formulae where only one or none of the approximations was exact (i.e. $\mathcal{L}(S^u_\varphi) \neq \mathcal{L}(S_\varphi)$ or $\mathcal{L}(S^o_\varphi) \neq \mathcal{L}(S_\varphi)$), we measured the "stochastic difference" between approximations by generating a number of random weighted words and estimating the probability that the over approximation accepts a random word, when the under approximation does not.

Table A.2 reports the amount of formulae for which the under or over approximation was exact and the amount of formulae where none of them was exact. It also contains the average time spent for generating the monitors and the average number of locations, and the stochastic difference.

5.2 Robot Control

We consider the case of a robot moving on a two-dimensional grid that was explored in e.g. [14]. Each field of the grid is either `normal`, on `fire`, cold as `ice` or it is a wall which cannot be passed. Also, there is a `goal` field that the robot must reach. The robot is moving randomly i.e. it stays in a field for some time, and then randomly moves to one of the neighbouring fields (if it is not a wall). Figure A.5 shows a robot controller implementing this along with the grid we use.

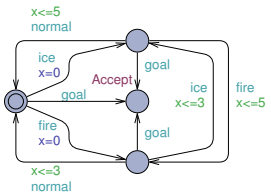


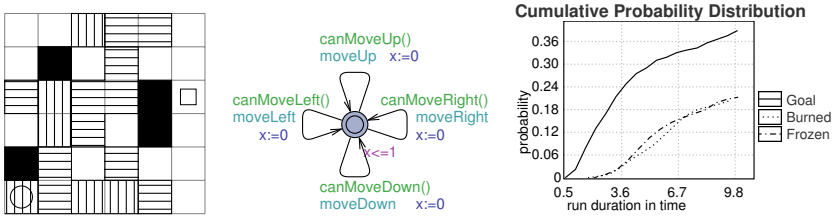
Figure A.4: Observer automaton used in [14]

We are interested in the probability that the robot reaches its goal location without staying on consecutive fire fields for more than one time units and on consecutive ice fields for more than two time units.

In [14] the authors solved this problem by manually constructing a monitoring automaton to operate in parallel with the model of the robot. The automaton they used is depicted in Figure A.4. Using WMTL$_{\leq}$ we can express the same requirement more easily as $\varphi \equiv (\varphi_1 \wedge \varphi_2)\mathsf{U}^{\tau}_{\leq 10}\texttt{goal}$, where:

	# exact				Avg. time (s)		Avg. size		Stochastic difference	
Actions	under	over	none	one	under	over	under	over	no exact	one exact
2	831	542	169	289	0.24	1.01	6.35	6.35	0.27	0.15
3	706	370	294	336	1.42	2.75	12.29	12.29	0.05	0.03
4	586	233	414	353	8.66	13.05	22.97	22.97	0.01	0.02

Table A.2: Results for the random generated formula test.



(a) (b) (c)

Figure A.5: (a) A 6×6 grid. The black fields are walls, the fields with vertical lines are on fire and the fields with horizontal lines contain ice. The circle indicates the robot's starting position and the square the goal.
(b) WTA implementing the random movement of the robot.
(c) Cumulative distribution of the robot reaching the goal, staying too long in the fire or too long on the ice.

$$\varphi_1 \equiv \texttt{ice} \implies \Diamond^{\tau}_{\leq 2}(\texttt{fire} \vee \texttt{normal} \vee \texttt{goal})$$
$$\varphi_2 \equiv \texttt{fire} \implies \Diamond^{\tau}_{\leq 1}(\texttt{ice} \vee \texttt{normal} \vee \texttt{goal})$$

Casaal produces an MWTA (6 locations, 55 edges) that is an exact under-approximation for φ. Based on this MWTA, our tool chain estimates the probability that the random behaviour of the robot satisfies φ to lie in the interval $[0.373, 0.383]$ with a confidence of 95%. Fig. A.5c shows how we can visualise and compare the different distributions using the plot composer of Uppaal SMC.

Energy

We extend the model by limiting the energy of the robot that will stop moving when it runs out of energy. Furthermore, it can regain energy while staying on fire fields and use additional energy while staying on ice fields. Let $\texttt{x}$ be the clock accumulating the amount of consumed energy. Now, we can express the property $\varphi \equiv (\varphi_1 \wedge \varphi_2 \wedge \neg\texttt{noEnergy})\mathsf{U}^{\texttt{x}}_{\leq 5}\texttt{goal}$ that the robot should not use more than 5 units of energy while obeying the requirements from before. The tool chain estimates the probability that the robot satisfies this requirement to lie in $[0.142; 0.152]$ with a confidence of 95%.

6 Related and Future Work

To our knowledge, we are the first to propose and implement an algorithm for translation of WMTL$_{\leq}$ formulae into monitoring automata. However, if we level down to MITL$_{\leq}$, there are several translation procedures described in the literature that are dealing with this logic. First, Alur et al. [9] presents

a procedure that is mostly theoretical and is not intended to be practically implemented. Second, Maler et al. [90] proposed a procedure to translate MITL into temporal testers (not the classic timed automata), their procedure also has not been implemented. Nickovic and Piterman [95] proposed an approach how to translate MTL to deterministic timed automata under *finite variability* assumption (this assumption is not valid for the WTA stochastic semantics that we use). Finally, Geilen [60] has implemented a procedure to translate $\text{MITL}_{\leq}$ to timed automata, but his approach works in the continuous semantics.

For future work we aim at extending our monitor- and approximate determinization constructions to $\text{WMTL}_{[a,b]}$ with (non-singleton) cost interval-bounds on the U modality in order to allow for SMC for this more expressive logic. Here a challenge will be how to bound the length of the random runs to be generated.

Rewrite-Based Statistical Model Checking of WMTL$_{[a,b]}$

B

Abstract *We present a new technique for verifying Weighted Metric Temporal Logic (WMTL) properties of weighted timed automata. Our approach relies on statistical model checking combined with a new monitoring algorithm based on rewriting rules. Contrary to existing monitoring approaches for WMTL ours is exact. The technique has been implemented in the statistical model checking engine of* Uppaal *and experiments indicate that the technique performs faster than existing approaches and leads to more accurate results.*

1 Introduction

Runtime verification (RV) [18, 68] is an emerging paradigm used to design a series of techniques whose main objective is to instrument the specification of a system (code, ...) in order to prove/disprove potentially complex properties at the execution level. Over the last years, RV has received a lot of interest and has been implemented in several toolsets. Such tools have successfully been applied on several real-life case studies.

The main problem with RV is that, contrary to classical verification techniques, it does not permit assessing the overall correctness of the entire system. Statistical model checking (SMC) [22, 107, 108, 120] extends runtime verification capabilities by exploiting statistical algorithms to get evidence that a given system satisfies some property. The core idea of the approach is to monitor several executions of the system. The results are then used together with algorithms from statistics to decide whether the system satisfies the property with a probability greater than some threshold. Statistical model checking techniques can also be used to estimate the probability that a system satisfies a given property [75]. In contrast to classical exhaustive formal verification approaches, a simulation-based solution does not guarantee a result with 100% confidence. However, it is possible to bound the probability of making an error. Simulation-based methods are known to be far less memory and time intensive than exhausative ones, and are sometimes the only option [122]. Statistical

model checking, which clearly complements RV, is widely accepted in various research areas such as software engineering, in particular for industrial applications, or even for solving problems originating from systems biology [64, 81].

To get a more accurate intuition, Fig. B.1 provides a schematic view of a statistical model checker and its interaction with RV procedures.

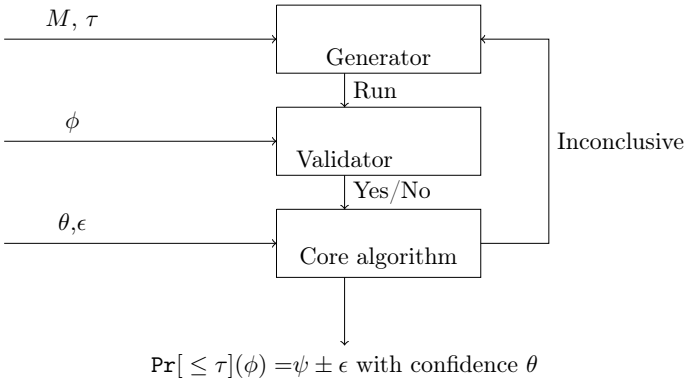


Figure B.1: A statistical model checker. The run generator first generates a run of M, which is propagated into the run validator. The run validator then validates if the run satisfies the property φ and returns Yes or No to the core algorithm. Afterwards the core algorithm decides if another run is needed or if it, based on the accumulated knowledge, can draw a conclusion.

The run generator is responsible for generating runs of the model under verification and the run validator, which corresponds to the RV part of the effort, validates if a run satisfy the property or not. The core algorithm collects the simulation results until sufficient samples has been obtained to provide an overall result. The core algorithm is computationally lightweight compared to the remaining two. An optimisation of SMC is therefore most easily obtained by optimising either the run generation or the run validation. In this paper, we focus on the run validation part.

In our work, we consider combining RV and SMC techniques in order to verify complex quantitative properties (performance evaluation, scheduling, ..) over rich systems. More precisely, we are interested in computing the probability that a random run of a weighted timed automaton (WTA) [20] satisfies a formula written in Weighted Metric Temporal Logic (WMTL) [28]. Weighted timed automaton is a rich formalism capable of capturing (quantitative) non-linear hybrid systems, while WMTL corresponds to the real-time extension of the linear temporal logic equipped with cost operators. In this paper, due to the use of SMC, we assume that the scope of the temporal operators is bounded, i.e., that one can decide whether a run satisfies a formula by only looking at a finite prefix. Unfortunately, it is known that, due to the expressivity of the automata-based model, the problem of verifying WMTL with respect to WTA

is undecidable [27] – hence it cannot be tackled with existing formal techniques such as model checking. Another drawback is that it is known that, even for the case where temporal operators are bounded, WMTL is more expressive than the class of deterministic timed automata [89]. This latter result implies that there is no automata-based runtime monitoring procedure for WMTL, even for the case where the scope of the temporal operators is finitely bounded. A first solution to the above problems could be to use a three-valued logic [19]. However, the absence of decision results is often unsatisfactory from an engineering point of view, especially when dealing with performance analysis.

In [45], we proposed the first SMC-based verification procedures for the eventually and always fragments of WMTL. Our work relies on a natural stochastic semantic for WTA. The work was implemented in Uppaal SMC and applied to a wide range of case studies. However, our original work does not consider nested temporal operators for which a solution was first proposed by Clarke et al. in [124]. While the approach in [124] is of clear interest, it only works for a subset of MTL where the temporal operators can only be upwards bounded, i.e., the lower bound is 0. In [35] we proposed another approach for this fragment, called $\text{WMTL}_{\leq}$, that relies on monitoring automata representing over and under approximations of solutions to the $\text{WMTL}_{\leq}$ formula. This approach, which has been implemented in Casaal and Uppaal SMC, exploits confidence levels obtained on both approximations in order to estimate the probability to satisfy the formula. The first drawback with the approach in [35] is that both the under and over approximation depend on some precision that has an influence on the confidence level returned by the SMC algorithms. The second drawback is that automata-based monitors may be of large size, hence intractable.

In this paper, we propose a new monitoring approach for WMTL formulas with upper and lower bounds, called weighted metric temporal $\text{logic}_{[a,b]}$ ($\text{WMTL}_{[a,b]}$). Contrary to existing approaches that work by first constructing a monitor for the property, ours exploit a graph-grammar procedure that rewrite the formula on-the-fly until a decision can be taken. The approach extends that of [104] to a timed logic. Contrary to existing off-line monitoring approaches [55], ours stops as soon as the formula is proved/disproved, which allows saving computation time and hence drastically improve both memory and time performances. Our approach has been implemented in Uppaal SMC and evaluated on several case studies, from random large-size formulas to concrete applications. As expected, there are many situations where we clearly outperform [35] while being more precise!

Outline In section 2 we introduce our modelling formalism Networks of WTAs. Later in section 3 we define the $\text{WMTL}_{[a,b]}$ logic, and section 4 describes our rewrite-based algorithm for monitoring of $\text{WMTL}_{[a,b]}$ properties. The experiments are described in section 5.

2 Networks of Weighted Timed Automata

In this section, we briefly recap the formalism of networks of WTAs [20]. Let $\mathtt{X}$ be a set of variables called clocks. A clock valuation over $\mathtt{X}$ is a function $\mathtt{v} : \mathtt{X} \to \mathbb{R}$ that assigns a real-valued number to each clock. We let $\mathtt{V}(\mathtt{X})$ denote all possible valuations over $\mathtt{X}$ and let $\vec{0}$ be the valuation that assign zero to all clocks. An *upper* (resp. *lower*) bound over $\mathtt{X}$ is of the form $\mathtt{x} \bowtie m$ where $\mathtt{x} \in \mathtt{X}$, $m \in \mathbb{N}$, and $\bowtie \in \{<, \leq\}$ (resp. $\bowtie \in \{>, \geq\}$). We denote by $\mathtt{B}^{\leq}(\mathtt{X})$ (resp. $\mathtt{B}^{\geq}(\mathtt{X})$) the set of upper (resp. lower) bounds over $\mathtt{X}$. We let $\mathtt{B}(\mathtt{X}) = \mathtt{B}^{\leq}(\mathtt{X}) \cup \mathtt{B}^{\geq}(\mathtt{X})$. Let $\mathtt{v}$ be a valuation over $\mathtt{X}$ and let $g \subseteq \mathtt{B}(\mathtt{X})$ then we write $\mathtt{v} \vDash g$ if for all $(\mathtt{x} \bowtie m) \in g$, $\mathtt{v}(\mathtt{x}) \bowtie m$. For a valuation $\mathtt{v} \in \mathtt{V}(\mathtt{X})$, a function $r : \mathtt{X} \to \mathbb{Q}$ and a $d \in \mathbb{R}$ we let $(\mathtt{v} + r \cdot d)$ be the valuation over $\mathtt{X}$ such that $(\mathtt{v} + r \cdot d)(\mathtt{x}) = \mathtt{v}(\mathtt{x}) + r(\mathtt{x}) \cdot d$ for every clock $\mathtt{x} \in \mathtt{X}$. Let $\mathtt{Y} \subseteq \mathtt{X}$ then $\mathtt{v}[\mathtt{Y} = 0]$ is the valuation that assigns zero to every clock in $\mathtt{Y}$ and agrees with $\mathtt{v}$ on all other clocks. For two valuations $\mathtt{v}_1$ and $\mathtt{v}_2$ we let $\mathtt{v}_2 - \mathtt{v}_1$ be the valuation $\mathtt{v}'$ where $\mathtt{v}'(\mathtt{x}) = \mathtt{v}_2(\mathtt{x}) - \mathtt{v}_1(\mathtt{x})$.

Definition 30. A WTA over the finite set of actions Σ and the set of propositions $\mathtt{AP}$ is a tuple $(\mathtt{L}, \mathtt{l}_0, \mathtt{X}, \mathtt{X}^o, \mathtt{E}, \mathtt{I}, \mathtt{R}, \mathtt{X}^{\mathtt{R}}, \mathtt{Pm})$, where

- $\mathtt{L}$ is a finite set of locations,
- $\mathtt{l}_0 \in \mathtt{L}$ is the initial location,
- $\mathtt{X}$ is a finite set of clocks,
- $\mathtt{X}^o \subseteq \mathtt{X}$ is a finite set of observable clocks
- $\mathtt{E} \subseteq \mathtt{L} \times 2^{\mathtt{B}^{\geq}(\mathtt{X})} \times \Sigma \times 2^{\mathtt{X}} \times \mathtt{L}$ is a set of edges,
- $\mathtt{I} : \mathtt{L} \to 2^{\mathtt{B}^{\leq}(\mathtt{X})}$ assigns invariants to the locations,
- $\mathtt{R} : \mathtt{L} \to \mathbb{Q}$ assigns transition rates to locations,
- $\mathtt{X}^{\mathtt{R}} : \mathtt{L} \to \mathtt{X} \to \mathbb{Q}$ assign rates to the clocks of the WTA and
- $\mathtt{Pm} : \mathtt{L} \to 2^{\mathtt{AP}}$ assign propositions to the locations of the WTA.

The semantics of a WTA $S = (\mathtt{L}, \mathtt{l}_0, \mathtt{X}, \mathtt{X}^o, \mathtt{E}, \mathtt{I}, \mathtt{R}, \mathtt{X}^{\mathtt{R}}, \mathtt{Pm})$ is given as a timed transition system with state space $\mathtt{L} \times \mathtt{V}(\mathtt{X})$ (denoted $(\mathtt{SP}(S))$ and initial state $(\mathtt{l}_0, \vec{0})$ (denoted $\mathtt{init}(S)$). For consistency, we require $\vec{0} \vDash \mathtt{I}(\mathtt{l}_0)$. Furthermore, we require that the rates of the observable clocks in any location is greater than 0. The transition rules are given below:

Delay Transition $(\mathtt{l}, \mathtt{v}) \xrightarrow{d} (\mathtt{l}, \mathtt{v}')$ where $d \in \mathbb{R}_{\geq 0}$, if $\mathtt{v}' = (\mathtt{v} + \mathtt{X}^{\mathtt{R}}(\mathtt{l}) \cdot d)$ and $\mathtt{v}' \vDash \mathtt{I}(\mathtt{l})$

Discrete Transition $(\mathtt{l}, \mathtt{v}) \xrightarrow{a} (\mathtt{l}', \mathtt{v}')$ if there exists $(\mathtt{l}, g, a, \mathtt{Y}, \mathtt{l}') \in \mathtt{E}$ such that $\mathtt{v} \vDash g$, $\mathtt{v}' = \mathtt{v}[\mathtt{Y} = 0]$ and $\mathtt{v}' \vDash \mathtt{I}(\mathtt{l}')$.

To prepare for composition of WTAs we assume that the set of actions Σ is partitioned into a set of input actions $\Sigma_{\mathtt{i}}$ and output actions $\Sigma_{\mathtt{o}}$. Also, we assume the WTA is input-enabled for the input actions $\Sigma_{\mathtt{i}}$, i.e. that for any $a \in \Sigma_{\mathtt{i}}$ and any state $(\mathtt{l}, \mathtt{v})$ there exists a transition $(\mathtt{l}, \mathtt{v}) \xrightarrow{a} (\mathtt{l}', \mathtt{v}')$. A WTA is deterministic for $\Sigma' \subseteq \Sigma$ if there exists at most one transition for each $a \in \Sigma$. In the paper, we let $S^i = (\mathtt{L}_i, \mathtt{l}_0^i, \mathtt{X}_i, \mathtt{X}_i^o, \mathtt{E}_i, \mathtt{I}_i, \mathtt{R}_i, \mathtt{X}_i^{\mathtt{R}}, \mathtt{Pm}_i)$

Network of WTAs A network of WTAs (NWTA) is a set of WTAs executing in parallel. The automata communicate via *broadcast synchronisation*.

Let $S^1, S^2, \ldots, S^n$ be WTAs over the common set of actions Σ. Furthermore, let $\Sigma^1, \Sigma^2 \ldots \Sigma^n$ be mutually disjoint subsets of Σ and for all i let S^i be deterministic and input-enabled with respect to $\Sigma \setminus \Sigma^i$ and deterministic with respect to Σ^i. Then we call $\mathcal{N} = S^1 \| S^2 \| \ldots \| S^n$ a network of WTAs over Σ where Σ^i is the output actions of S^i and $\Sigma \setminus \Sigma^i$ is its input actions.

The semantics of the network of WTAs is a timed transition system with the state space $\mathtt{SP}(\mathcal{N}) = \mathtt{SP}(S^1) \times \mathtt{SP}(S^2) \times \cdots \times \mathtt{SP}(S^n)$ and the initial state $(\mathtt{init}(S^1), \mathtt{init}(S^2), \ldots, \mathtt{init}(S^n))$. We refer to an element $\mathbf{s} = (s_1, s_2, \ldots, s_n) \in \mathtt{SP}(\mathcal{N})$ as a state vector of the network and let $\mathbf{s}^i = s_i$. The transition rules of a network is given as

- $(\mathbf{s}) \xrightarrow{d} (\mathbf{s}')$ if for all $i, 1 \leq i \leq n$, $\mathbf{s}^i \xrightarrow{d} \mathbf{s}'^i$, and $d \in \mathbb{R}_{\geq 0}$
- $(\mathbf{s}) \xrightarrow{a} (\mathbf{s}')$ if for all $i, 1 \leq i \leq n$ $\mathbf{s}^i \xrightarrow{a} \mathbf{s}'^i$, and $a \in \Sigma$.

Consider the WTAs given in Fig. B.2. WTAs (a) and (b) are competing to force (c) to go either to location Left or to location Right. Initially both competitors are waiting for between 3 and 5 time units where after one of them moves the (c) to either Left or Right. Afterwards both competitors have a period where time progresses and nothing occurs. Indeed, when one of the competitors returns it must wait for between 3 and 5 time units again and choose to either move (c) or let it be and enter a waiting period again. The primary difference between (a) and (b) is that (b) rushes to return to a position from which it can change (c) and (a) returns within 5 time units.

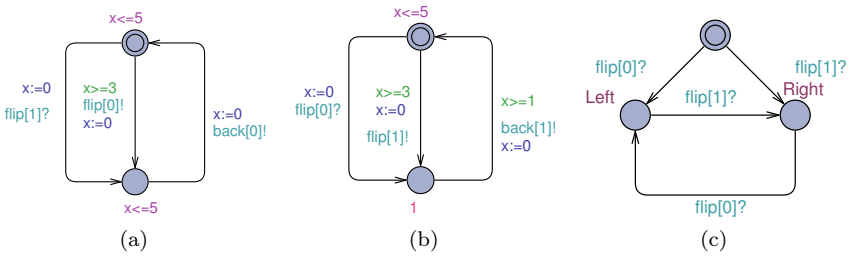


Figure B.2: Network of Timed Automata.

Let $\mathbf{s} = ((\mathtt{l}_1, \mathtt{v}_1), (\mathtt{l}_2, \mathtt{v}_2), \ldots, (\mathtt{l}_n, \mathtt{v}_n))$ be a state vector for $S^1 \| S^2 \| \ldots \ S^n$. Then we let $\mathtt{Pm}(\mathbf{s}) = \bigcup_{i=1}^{n} \mathtt{Pm}_i(\mathtt{l}_i)$. Let $\mathtt{x} \in \mathtt{X}_i$ for som i then $V(\mathbf{s}, \mathtt{x}) = \mathtt{v}_i(\mathtt{x})$.

Definition 31 (Run)**.** Let $S^1 \| S^2 \| \ldots \| S^n$ be a network of WTAs. A run of the network is an infinite weighted word $(\mathtt{P}_0, \mathtt{v}_0)(\mathtt{P}_1, \mathtt{v}_1) \ldots$ where for all $i, \mathtt{v}_i$ is a valuation over $\mathtt{Y} = \bigcup_{i \in \{1,2,\ldots,n\}} \mathtt{X}_i^o$ and

- $\mathtt{v}_0 = \vec{0}$,
- there exists an alternating sequence of delays and discrete transitions $\mathbf{s}_0 \xrightarrow{d_0} \mathbf{s}'_0 \xrightarrow{a_0} \mathbf{s}_1 \xrightarrow{d_1} \ldots$, where for all i, $0 < i$, and for all $\mathtt{x} \in \mathtt{Y}$ $\mathtt{v}_i(\mathtt{x}) = \mathtt{v}_{i-1}(\mathtt{x}) + (V(\mathbf{s}'_{i-1}, \mathtt{x}) - V(\mathbf{s}_{i-1}, \mathtt{x}))$.
- $\mathbf{s}_0 = (\mathtt{init}(S^1), \mathtt{init}(S^2), \ldots, \mathtt{init}(S^n))$ and
- for all $j, j \geq 0, \mathtt{P}_j = \mathtt{Pm}(\mathbf{s}_j)$.

For a run $\pi = (\mathtt{P}_0, \mathtt{v}_0)(\mathtt{P}_1, \mathtt{v}_1) \ldots$, we let $\pi^{i:} = (\mathtt{P}_i, \mathtt{v}_i)(\mathtt{P}_{i+1}, \mathtt{v}_{i+1}) \ldots$. A run π is called *diverging* for clock $\mathtt{x}$ if for any i there exists a j such that $\mathtt{v}_j(\mathtt{x}) > \mathtt{v}_i(\mathtt{x}) + 1$. A run is diverging if it is diverging for all clocks. In what follows, we assume that there always exists a clock τ in a WTA, and this clock always have a rate of 1 and is never reset, i.e. τ measures the time length of a run.

Stochastic Semantics In [45] we introduced the stochastic semantics for NWTAs, i.e. proposed a probability measure on the set of all runs of a network and described an algorithm for generating *a random run*. Roughly speaking, the stochastic semantics of WTA components associates probability distributions on both the delays one can spend in a given state and on the transition between states. In Uppaal SMC uniform distributions are applied for bounded delays and exponential distributions for the case where a component has unbounded delay. In a network of WTAs the components repeatedly race against each other, i.e. they independently and stochastically decide on their own how much to delay before outputting, with the "winner" being the component that chooses the minimum delay.

Statistical Model Checking As said in the introduction, we use SMC [22, 107, 108, 120, 123] to compute the probability for a network of WTAs to satisfy a given property. Given a program B and a trace-based property[1] ϕ , SMC refers to a series of simulation-based techniques that can be used to answer two questions: (1) *qualitative:* is the probability for B to satisfy ϕ greater or equal to a certain threshold θ (or greater or equal to the probability to satisfy another property ϕ') [120]? And (2) *quantitative:* what is the probability for B to satisfy ϕ [75]? In both cases, the answer is correct up to some confidence level, i.e., probability that the algorithm does not make mistake, whose value

[1] i.e. a property with semantics defined on traces.

can be configured by the user. For the quantitative approach, which we will intensively use in this paper, the method computes a confidence interval that is an interval of probabilities that contains the true probability to satisfy the property. The confidence level is interpreted as the probability for the algorithm to compute a confidence interval that indeed contains the probability to satisfy the property.

Our UPPAAL SMC toolset implements a wide range of SMC algorithms for WTAs. In addition, the tool offers several features to visualise and reason on the results. Until now, the monitoring procedure for WMTL relies on a technique that computes over and under approximation monitors for the formulas. In this paper, we go one big step further and propose a more efficient and precise monitoring procedure.

3 Weighted Metric Temporal Logic

In this section we review the syntax and semantics of weighted metric temporal logic$_{[a,b]}$ (WMTL$_{[a,b]}$). The syntax is defined as follows.

Definition 32. A WMTL$_{[a,b]}$ formula over the propositions $\mathtt{AP}$ and the clocks $\mathtt{X}$ is generated by the grammar:

$$\varphi, \varphi_1, \varphi_2 ::= \mathsf{tt} \mid \mathsf{ff} \mid p \mid \neg p \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U}^{\mathtt{x}}_{[a,b]} \varphi_2 \mid \varphi_1 \mathsf{R}^{\mathtt{x}}_{[a,b]} \varphi_2$$

where $a, b \in \mathbb{Q}$, $a \leq b$, $p \in \mathtt{AP}$ and $\mathtt{x} \in \mathtt{X}$.

As one can see in the syntax, we restrict to a fragment of WMTL where temporal operators are bounded. As stated in the introduction, this fragment is sufficient to break any decidability results. Observe that WMTL$_{[a,b]}$ is an extension of MTL [83] in which U and R can also be bounded for arbitrary clocks. As an example, bounding U and R over arbitrary clock allows one to express that a communication device should recover from a state without spending more than x units of energy. This can be accomplished by adding an observable clock, that measures the energy consumption, to the model and bound the U and R modalities over this clock.

We interpret WMTL$_{[a,b]}$ formulas over runs of WTAs. Informally, the WMTL$_{[a,b]}$ formula $\varphi_1 \mathsf{U}^{\mathtt{x}}_{[a,b]} \varphi_2$ is satisfied by a run if φ_1 is satisfied on the run until φ_2 is satisfied, and this should happen before the value of the clock $\mathtt{x}$ increases with more than b units starting from the beginning of the run, and after it increases for more than a units. Formula $\mathsf{X}\varphi$ means that φ should be satisfied starting from the next observation of the run. The logical operators are defined as usual, and the *release* operator R is dual to U i.e. $\varphi_1 \mathsf{R}^{\mathtt{x}}_{[a,b]} \varphi_2 \equiv \neg(\neg\varphi_1 \mathsf{U}^{\mathtt{x}}_{[a,b]} \neg\varphi_2)$.

Formally, let $\pi = (\mathtt{P}_0, \mathtt{v}_0)(\mathtt{P}_1, \mathtt{v}_1) \ldots$ be a timed run. The satisfaction relation is inductively defined as

- $\pi \models \mathsf{tt}$

- $\pi \vDash p$ if $p \in \mathsf{P}_0$
- $\pi \vDash \neg p$ if $p \notin \mathsf{P}_0$
- $\pi \vDash \varphi_1 \vee \varphi_2$ if $\pi \vDash \varphi_1$ or $\pi \vDash \varphi_2$
- $\pi \vDash \varphi_1 \wedge \varphi_2$ if $\pi \vDash \varphi_1$ and $\pi \vDash \varphi_2$
- $\pi \vDash \mathsf{X}\varphi$ if $\pi^{1:} \vDash \varphi$
- $\pi \vDash \varphi_1 \mathsf{U}^{\mathtt{x}}_{[a,b]} \varphi_2$ if there exists i such that $a \leq \mathtt{v}_i(\mathtt{x}) - \mathtt{v}_0(\mathtt{x}) \leq b, \pi^{i:} \vDash \varphi_2$ and for all $j < i$ we have $\pi^{j:} \vDash \varphi_1$
- $\pi \vDash \varphi_1 \mathsf{R}^{\mathtt{x}}_{[a,b]} \varphi_2$ if for all i where $a \leq \mathtt{v}_i(\mathtt{x}) - \mathtt{v}_0(\mathtt{x}) \leq b, \pi^{i:} \vDash \varphi_2$ or there exists $j < i$ where $\pi^{j:} \vDash \varphi_1$.

In the rest of the paper, we use the following equivalences: $\Diamond^{\mathtt{x}}_{[a,b]}\varphi = \mathsf{tt}\mathsf{U}^{\mathtt{x}}_{[a,b]}\varphi$ and $\Box^{\mathtt{x}}_{[a,b]}\varphi = \mathsf{ff}\mathsf{R}^{\mathtt{x}}_{[a,b]}\varphi$. We also use $\Box_{[a,b]}\varphi$ instead of $\Box^{\tau}_{[a,b]}\varphi$ for the case where τ always grows with rate 1.

Example 12. Consider again the WTAs in Fig. B.2 and assume that the winner of the competition is the one who managed to have (c) located in its designated location for 8 consecutive time units. To express that (a) wins within 100 time units we need to state that (c) stays in Left for 8 consecutive time units at some point and that it has not stayed in Right for 8 consecutive time units before that point. Using WMTL this can be expressed like

$$(\neg \mathsf{Right} \vee \Diamond_{[0,8]} \mathsf{Left}) \mathsf{U}_{[0,92]} (\Box_{[0,8]} \mathsf{Left}).$$

We now focus on deciding a WMTL$_{[a,b]}$ formula φ on a finite prefix of an infinite diverging run $\pi = (\mathsf{P}_0, \mathtt{v}_0)(\mathsf{P}_1, \mathtt{v}_1)\ldots$. We first define the bound function $N(\pi, \varphi)$ inductively as follows:

$$\begin{aligned}
N(\pi, \mathsf{tt}) = N(\pi, \mathsf{ff}) = N(\pi, p) &= 0 \\
N(\pi, \neg p) &= 0 \\
N(\pi, \varphi_1 \wedge \varphi_2) &= \max\{N(\pi, \varphi_1), N(\pi, \varphi_2)\} \\
N(\pi, \varphi_1 \vee \varphi_2) &= \max\{N(\pi, \varphi_1), N(\pi, \varphi_2)\} \\
N(\pi, \mathsf{X}(\varphi)) &= 1 + N(\pi^{1:}, \varphi) \\
N(\pi, \varphi_1 \mathsf{U}^{\mathtt{x}}_{[a;b]} \varphi_2) &= \max(\{i + N(\pi^{i:}, \varphi_2), j + N(\pi^{j:}, \varphi_1) \mid \\
&\quad a \leq \mathtt{v}_i(\mathtt{x}) - \mathtt{v}_0(\mathtt{x}) \leq b \wedge j < i\}) \\
N(\pi, \varphi_1 \mathsf{R}^{\mathtt{x}}_{[a;b]} \varphi_2) &= \max(\{i + 1, i + N(\pi^{i:}, \varphi_2), j + N(\pi^{j:}, \varphi_1) \mid \\
&\quad a \leq \mathtt{v}_i(\mathtt{x}) - \mathtt{v}_0(\mathtt{x}) \leq b \wedge j \leq i\})
\end{aligned}$$

The bound function characterises the maximal prefix of π that one needs to observe to decide φ. Observe that, contrary to [124], the bound depends not only on the formula but also on the run itself. The latter is due to the introduction of the next operator that is absent in [124].

We say that two infinite runs $\pi_1 = (\mathtt{P}_0^1, \mathtt{v}_0^1)(\mathtt{P}_1^1, \mathtt{v}_1^1)\ldots$ and $\pi_2 = (\mathtt{P}_0^2, \mathtt{v}_0^2)$ $(\mathtt{P}_1^2, \mathtt{v}_1^2)\ldots$ are n-equivalent, denoted $\pi_1 \equiv_n \pi_2$, if for all $i \leq n$ $\mathtt{P}_i^1 = \mathtt{P}_i^2$ and $\mathtt{v}_i^1 = \mathtt{v}_i^2$. We say that π n-boundly satisfies φ, denoted $\pi \models^n \varphi$, iff for all π' where $\pi \equiv_n \pi'$, $\pi' \models \varphi$. We say that π run n-boundly violate φ if for all π' where $\pi \equiv_n \pi'$, $\pi' \not\models \varphi$. It is easy to see that $\pi \models^n \varphi \implies \pi \models \varphi$ and $\pi \not\models^n \varphi \implies \pi \not\models \varphi$. We can now conclude with the following theorem that shows that any $\mathrm{WMTL}_{[a,b]}$ property can be decided on a finite prefix of the run.

Theorem 4. Let $\pi = (\mathtt{P}_0, \mathtt{v}_0)(\mathtt{P}_1, \mathtt{v}_1)$ be an infinite run and φ be a $\mathrm{WMTL}_{[a,b]}$ formula. Then $\pi \models \varphi$ if and only if $\pi \models^{N(\pi,\varphi)} \varphi$.

4 Monitoring WMTL Properties

We present an efficient online monitoring algorithm for checking if a given infinite run π of a WTA satisfies a given $\mathrm{WMTL}_{[a,b]}$ property φ.

Algorithm B.1: WMTL formula satisfiability checking

```
1  // Input: WMTL_[a,b] formula φ and weighted word ω
2  // Output: tt iff π ⊨ φ, ff otherwise
3  i:=0
4  while φ ≠ tt ∧ φ ≠ ff do
5  |  φ:=Simp(Rew(φ, P_i, v_{i+1} − v_i))
6  |  i:=i+1
7  end
8  if φ == tt then
9  |  return tt
10 end
11 if φ == ff then
12 |  return ff
13 end
```

The pseudo code of our algorithm is presented in Algorithm B.1. Intuitively, the algorithm reads the elements of the input run one-by-one and rewrites the formula after reading each new element. The algorithm stops when the formula becomes tt or ff meaning any continuation of the finite prefix read so far will be accepted (or rejected) by the original formula φ. The rewriting step is performed by first applying the function `Rew` that *updates* the formula according to a new observation, and then applying `Simp` function that *simplifies* the formula and tries to reduce it to tt or ff.

The *rewrite* function $\texttt{Rew}$ is defined by the following recursive rules where $\texttt{v}$ is a function that gives the change of the clock variables since the last element of the run:

- $\texttt{Rew}(p, \texttt{P}, \texttt{v}) = \begin{cases} \texttt{tt}, & \text{if } p \in \texttt{P} \\ \texttt{ff}, & \text{if } p \notin \texttt{P} \end{cases}$
- $\texttt{Rew}(\neg p, \texttt{P}, \texttt{v}) = \begin{cases} \texttt{ff}, & \text{if } p \in \texttt{P} \\ \texttt{tt}, & \text{if } p \notin \texttt{P} \end{cases}$
- $\texttt{Rew}(\varphi_1 \wedge \varphi_2, \texttt{P}, \texttt{v}) = \texttt{Rew}(\varphi_1, \texttt{P}, \texttt{v}) \wedge \texttt{Rew}(\varphi_2, \texttt{P}, \texttt{v})$
- $\texttt{Rew}(\varphi_1 \vee \varphi_2, \texttt{P}, \texttt{v}) = \texttt{Rew}(\varphi_1, \texttt{P}, \texttt{v}) \vee \texttt{Rew}(\varphi_2, \texttt{P}, \texttt{v})$
- $\texttt{Rew}(\mathsf{X}\,\varphi, \texttt{P}, \texttt{v}) = \varphi$
- $\texttt{Rew}(\varphi_1 \mathsf{U}^{\texttt{x}}_{[a,\,b]} \varphi_2, \texttt{P}, \texttt{v}) =$
$$\begin{cases} \texttt{Rew}(\varphi_1, \texttt{P}, \texttt{v}) \wedge \varphi_1 \mathsf{U}^{\texttt{x}}_{[max(a-\texttt{v}(\texttt{x}),\,0),\,b-\texttt{v}(\texttt{x})]} \varphi_2\,, & \text{if } a > 0 \wedge \texttt{v}(\texttt{x}) \leq b \\ \texttt{Rew}(\varphi_2, \texttt{P}, \texttt{v}) \vee (\texttt{Rew}(\varphi_1, \texttt{P}, \texttt{v}) \wedge \varphi_1 \mathsf{U}^{\texttt{x}}_{[0,\,b-\texttt{v}(x)]} \varphi_2)\,, & \text{if } a = 0 \wedge \texttt{v}(\texttt{x}) \leq b \\ \texttt{Rew}(\varphi_2, \texttt{P}, \texttt{v}), & \text{if } a = 0 \wedge \texttt{v}(\texttt{x}) > b \\ \texttt{ff}, & \text{if } a > 0 \wedge \texttt{v}(\texttt{x}) > b \end{cases}$$
- $\texttt{Rew}(\varphi_1 \mathsf{R}^{\texttt{x}}_{[a,\,b]} \varphi_2, \texttt{P}, \texttt{v}) =$
$$\begin{cases} \texttt{Rew}(\varphi_1, \texttt{P}, \texttt{v}) \vee \varphi_1 \mathsf{R}^{\texttt{x}}_{[max(a-\texttt{v}(\texttt{x}),\,0),\,b-\texttt{v}(\texttt{x})]} \varphi_2\,, & \text{if } a > 0 \wedge \texttt{v}(\texttt{x}) \leq b \\ \texttt{Rew}(\varphi_2, \texttt{P}, \texttt{v}) \wedge (\texttt{Rew}(\varphi_1, \texttt{P}, \texttt{v}) \vee \varphi_1 \mathsf{R}^{\texttt{x}}_{[0,\,b-\texttt{v}(x)]} \varphi_2)\,, & \text{if } a = 0 \wedge \texttt{v}(\texttt{x}) \leq b \\ \texttt{Rew}(\varphi_2, \texttt{P}, \texttt{v}), & \text{if } a = 0 \wedge \texttt{v}(\texttt{x}) > b \\ \texttt{tt} & \text{if } a > 0 \wedge \texttt{v}(\texttt{x}) > b \end{cases}$$

The omitted cases are all rewritten into themselves. The *simplify* function $\texttt{Simp}$ is defined by the following recursive rules:

- $\texttt{Simp}(\varphi_1 \wedge \varphi_2) = \begin{cases} \texttt{ff}, & \text{if } \texttt{Simp}(\varphi_1) = \texttt{ff} \text{ or } \texttt{Simp}(\varphi_2) = \texttt{ff} \\ \texttt{Simp}(\varphi_1), & \text{if } \texttt{Simp}(\varphi_2) = \texttt{tt} \\ \texttt{Simp}(\varphi_2), & \text{if } \texttt{Simp}(\varphi_1) = \texttt{tt} \\ \texttt{Simp}(\varphi_1) \wedge \texttt{Simp}(\varphi_2), & \text{otherwise.} \end{cases}$
- $\texttt{Simp}(\varphi_1 \vee \varphi_2) = \begin{cases} \texttt{tt}, & \text{if } \texttt{Simp}(\varphi_1) = \texttt{tt} \text{ or } \texttt{Simp}(\varphi_2) = \texttt{tt} \\ \texttt{Simp}(\varphi_1), & \text{if } \texttt{Simp}(\varphi_2) = \texttt{ff} \\ \texttt{Simp}(\varphi_2), & \text{if } \texttt{Simp}(\varphi_1) = \texttt{ff} \\ \texttt{Simp}(\varphi_1) \vee \texttt{Simp}(\varphi_2), & \text{otherwise.} \end{cases}$
- $\texttt{Simp}(\varphi) = \varphi$ in rest of the cases.

The *simplify* function $\texttt{Simp}$ takes into account only the logical equivalences, namely $\varphi \wedge \mathsf{tt} \equiv \varphi$, $\varphi \wedge \mathsf{ff} \equiv \mathsf{ff}$, $\varphi \vee \mathsf{tt} \equiv \mathsf{tt}$, $\varphi \vee \mathsf{ff} \equiv \varphi$.

The correctness and termination of our algorithm is proved by the following lemmas:

Lemma 6. Let $\pi = (\mathtt{P}_0, \mathtt{v}_0), (\mathtt{P}_1, \mathtt{v}_1), \ldots$ be an infinite weighted word, and φ be a $\text{WMTL}_{[a,b]}$ formula. If $\pi^{i+1:} \vDash \mathtt{Rew}(\varphi, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i)$ then $\pi^{i:} \vDash \varphi$.

Proof. Induction in the structure of φ. Let $\varphi =$

- p.
  Then since $\pi^{i+1:} \vDash \mathtt{Rew}(\varphi, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i)$ we know $\mathtt{Rew}(\varphi, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i) = \mathsf{tt}$ thus by the definition $p \in \mathtt{P}_i$ hence $\pi^{i:} \vDash p$.

- $\neg p$.
  Similar to above.

- $\varphi_1 \wedge \varphi_2$.
  Then $\mathtt{Rew}(\varphi, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i) = \mathtt{Rew}(\varphi_1, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i) \wedge \mathtt{Rew}(\varphi_2, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i)$ thus $\pi^{i+1:} \vDash \mathtt{Rew}(\varphi_1, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i)$ and $\pi^{i+1:} \vDash \mathtt{Rew}(\varphi_2, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i)$. By induction hypothesis then $\pi^{i:} \vDash \varphi_1$ and $\pi^{i:} \vDash \varphi_2$.

- $\varphi_1 \vee \varphi_2$.
  Similar to above.

- $\mathsf{X}\varphi_1$.
  Then $\mathtt{Rew}(\varphi, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i) = \varphi_1$ and $\pi^{i+1:} \vDash \varphi_1$ thus $\pi^{i:} \vDash \mathsf{X}(\varphi_1)$.

- $\varphi_1 \mathsf{U}^{\mathtt{x}}_{[a,\,b]} \varphi_2$.
  In the remainder let $\mathtt{v} = \mathtt{v}_{i+1} - \mathtt{v}_i$. We divide into cases and let $\mathtt{Rew}(\varphi_1 \mathsf{U}^{\mathtt{x}}_{[a,\,b]} \varphi_2) =$

  - $\mathtt{Rew}(\varphi_1, \mathtt{P}_i, \mathtt{v}) \wedge \varphi_1 \mathsf{U}^{\mathtt{x}}_{[max(a - \mathtt{v}(\mathtt{x}),\, 0),\, b - \mathtt{v}(\mathtt{x})]} \varphi_2$
    Let us consider the case where $max(a - \mathtt{v}(\mathtt{x}), 0) = a - \mathtt{v}(\mathtt{x})$. Then since $\pi^{i+1:} \vDash \varphi_1 \mathsf{U}^{\mathtt{x}}_{[max(a-\mathtt{v}(\mathtt{x}),\, 0),\, b-\mathtt{v}(\mathtt{x})]} \varphi_2$ there exists a $k \geq i+1$ such that $\pi^{k:} \vDash \varphi_2$, $a - \mathtt{v}(\mathtt{x}) \leq \mathtt{v}_k(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) \leq b - \mathtt{v}(\mathtt{x})$ and for all $j < k$, $\pi^{j:} \vDash \varphi_1$. Also, by induction hypothesis $\pi^{i:} \vDash \varphi_1$ thus to prove that $\pi^{i:} \vDash \varphi_1 \mathsf{U}^{\mathtt{x}}_{[a,\,b]} \varphi_2$ we should only prove that $a \leq \mathtt{v}_k(\mathtt{x}) - \mathtt{v}_i(k) \leq b$. Recall that $\mathtt{v} = (\mathtt{v}_{i+1} - \mathtt{v}_i)$ thus

    $$\begin{aligned} & a - \mathtt{v}(\mathtt{x}) \leq \mathtt{v}_k(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) \leq b - \mathtt{v}(\mathtt{x}) \Leftrightarrow \\ & a \leq \mathtt{v}_k(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) + \mathtt{v}(\mathtt{x}) \leq b \Leftrightarrow \\ & a \leq \mathtt{v}_k(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) + \mathtt{v}_{i+1} - \mathtt{v}_i(\mathtt{x}) \leq b \Leftrightarrow \\ & a \leq \mathtt{v}_k(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) \leq b \end{aligned}$$

    For the case where $max(a - \mathtt{v}(\mathtt{x}), 0) = 0$ the proof is similar - we only need to realise that $a - \mathtt{v}(\mathtt{x}) < 0$

  - $\mathtt{Rew}(\varphi_2, \mathtt{P}_i, \mathtt{v}) \vee (\mathtt{Rew}(\varphi_1, \mathtt{P}_i, \mathtt{v}) \wedge \varphi_1 \mathsf{U}^{\mathtt{x}}_{[0,\, b-\mathtt{v}(x)]} \varphi_2)$. Two possibilities:
    1. $\pi^{i+1:} \vDash \mathtt{Rew}(\varphi_2, \mathtt{P}_i, \mathtt{v})$. Then by induction hypothesis $\pi^{i:} \vDash \varphi_2$ and since $a = 0$ then $\pi^{i:} \vDash \varphi_1 \mathsf{U}_{[a,b]} \varphi_2$.
    2. $\pi^{i+1:} \vDash \mathtt{Rew}(\varphi_1, \mathtt{P}_i, \mathtt{v}) \wedge \varphi_1 \mathsf{U}^{\mathtt{x}}_{[0, b-\mathtt{v}(\mathtt{x})]} \varphi_2$. Similar to the case where $\mathtt{Rew}(\varphi_1 \mathsf{U}^{\mathtt{x}}_{[a,\, b]} \varphi_2) = \mathtt{Rew}(\varphi_1, \mathtt{P}_i, \mathtt{v}) \wedge \varphi_1 \mathsf{U}^{\mathtt{x}}_{[max(a-\mathtt{v}(\mathtt{x}),\, 0),\, b-\mathtt{v}(\mathtt{x})]} \varphi_2$
  - Remaining cases are omitted. They are similar to above

- Remaining cases are omitted as they are quite similar to the U case.

□

Lemma 7. Let $\pi = (\mathtt{P}_0, \mathtt{v}_0), (\mathtt{P}_1, \mathtt{v}_1), \ldots$ be an infinite weighted word, and φ be a WMTL$_{[a,b]}$ formula. If $\pi^{i:} \vDash \varphi$ then $\pi^{i+1:} \vDash \mathtt{Rew}(\varphi, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i)$.

Proof. Induction in the structure of φ. Let $\mathtt{v} = \mathtt{v}_{i+1} - \mathtt{v}_i$ and let $\varphi =$

- p
  Then since $\pi^{i:} \vDash p$ we know that $p \in \mathtt{P}_i$ thus $\mathtt{Rew}(p, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i) = \mathtt{tt}$ and $\pi^{i+1:} \vDash \mathtt{tt}$.

- $\neg p$
  Similar to above.

- $\mathsf{X}\varphi_1$.
  Then $\pi^{i+1:} \vDash \varphi_1$ and $\mathtt{Rew}(\mathsf{X}\varphi_1, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i) = \varphi_1$ thus clearly $\pi^{i+1:} \vDash \mathtt{Rew}(\mathsf{X}\varphi_1, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i)$.

- $\varphi_1 \mathsf{U}^{\mathtt{x}}_{[a,\, b]} \varphi_2$. We will divide into different cases depending on the values of a and b in relation to $\mathtt{v}(\mathtt{x})$. Let

  - $a > 0 \wedge \mathtt{v}(\mathtt{x}) \leq b$
    Then there exists $k \geq i$ s.t. $\pi^{k:} \vDash \varphi_2$ and for all j, $i \leq j < k$, $\pi^{j:} \vDash \varphi_1$ and $a \leq \mathtt{v}_k(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) \leq b$ - notice that since $a > 0$ then $k > i$. Also $\mathtt{Rew}(\varphi, \mathtt{P}_i, \mathtt{v}) = \mathtt{Rew}(\varphi_1, \mathtt{P}_i, \mathtt{v}) \wedge \varphi_1 \mathsf{U}^{\mathtt{x}}_{[max(a-\mathtt{v}(\mathtt{x}),\, 0),\, b-\mathtt{v}(\mathtt{x})]} \varphi_2$.
    As $\pi^{i:} \vDash \varphi_1$ then by induction hypothesis $\pi^{i+1:} \vDash \mathtt{Rew}(\varphi_1, \mathtt{P}_i, \mathtt{v})$. To prove that $\pi^{i+1:} \vDash \varphi_1 \mathsf{U}^{\mathtt{x}}_{[a',\, b-\mathtt{v}(\mathtt{x})]} \varphi_2$, where $a' = max(a - \mathtt{v}(\mathtt{x}), 0)$ we need to show that $a' \leq \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) \leq b - \mathtt{v}(\mathtt{x})$. Let

    * $a' = a - \mathtt{v}(\mathtt{x})$. Then

$$\begin{aligned} a \leq \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) &\leq b \Leftrightarrow \\ a - \mathtt{v}(\mathtt{x}) \leq \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_i - \mathtt{v}(\mathtt{x}) &\leq b - \mathtt{v}(\mathtt{x}) \Leftrightarrow \\ a - \mathtt{v}(\mathtt{x}) \leq \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_i - (\mathtt{v}_{i+1}(\mathtt{x}) - \mathtt{v}_i(\mathtt{x})) &\leq b - \mathtt{v}(\mathtt{x}) \Leftrightarrow \\ a' \leq \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) &\leq b - \mathtt{v}(\mathtt{x}) \end{aligned}$$

∗ $a' = 0$. Then as before

$$a - \mathtt{v}(\mathtt{x}) \leq \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) \leq b - \mathtt{v}(\mathtt{x})$$

The crucial step is realising that $a - \mathtt{v}(\mathtt{x}) \leq 0$ and that $0 \leq \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x})$ due to $\mathtt{v}_j(\mathtt{x}) \geq \mathtt{v}_{i+1}(\mathtt{x})$ and hence

$$a - \mathtt{v}(\mathtt{x}) \leq 0 \leq \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_{i+1}(\mathtt{x}) \leq b - \mathtt{v}(\mathtt{x})$$

– $a = 0 \wedge \mathtt{v}(\mathtt{x}) \leq b$
Omitted. Similar to above.

– $a = 0 \wedge \mathtt{v}(\mathtt{x}) > b$
Then there exists $k \geq i$ s.t. $\pi^{k:} \vDash \varphi_2$ and for all j, $i \leq j < k$, $\pi^{j:} \vDash \varphi_1$ and $a \leq \mathtt{v}_k(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) \leq b$. Also $\mathtt{Rew}(\varphi, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i) = \mathtt{Rew}(\varphi_2, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i)$.
Since $\mathtt{v}(\mathtt{x}) > b \Leftrightarrow \mathtt{v}_{i+1}(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) > b$ then for any $k' > i$ we can easily see that $\mathtt{v}_{k'}(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) > b$ thus obviously $k = i$. This means that $\pi^{i:} \vDash \varphi_2$ and thus by induction hypothesis $\pi^{i+1:} \vDash \mathtt{Rew}(\varphi_2, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i) = \mathtt{Rew}(\varphi, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i)$.

– $a > 0 \wedge \mathtt{v}(\mathtt{x}) > b$. Then there cannot exist any j where $a \leq \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_i(\mathtt{x}) \leq b$ thus $\pi^{i:} \nvDash \varphi$.

• $\varphi_1 \mathsf{R}^{\mathtt{x}}_{[a,\,b]} \varphi_2$.
Omitted - quite similar to U case.

□

Lemma 8. Let $\pi = (\mathtt{P}_0, \mathtt{v}_0), (\mathtt{P}_1, \mathtt{v}_1), \ldots$ be an infinite weighted word and φ^0 be a WMTL$_{[a,b]}$ formula. For all $i \geq 0$ let $\varphi^{i+1} = \mathtt{Simp}(\mathtt{Rew}(\varphi^i, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i))$.
If there exists $k \geq 0$ such that $\varphi^k = \mathsf{tt}$ then $\pi \vDash \varphi^0$.

Proof. First realise that $\pi^{i:} \vDash \varphi$ if and only if $\pi^{i:} \vDash \mathtt{Simp}(\varphi)$ - this is given without proof but should be obvious from the definition of $\wedge$ and $\vee$.

Let $\varphi^k = \mathsf{tt}$. Then $\pi^{k:} \vDash \varphi_k$ and since $\mathtt{Simp}(\mathtt{Rew}(\varphi^{k-1}, \mathtt{P}_{k-1}, \mathtt{v}_k - \mathtt{v}_{k-1})) = \varphi_k$ then $\pi^{k-1:} \vDash \varphi^{k-1}$. Continuing this reasoning easily gives $\pi^{0:} \vDash \varphi^0$. □

Lemma 9. Let $\pi = (\mathtt{P}_0, \mathtt{v}_0), (\mathtt{P}_1, \mathtt{v}_1), \ldots$ be an infinite weighted word and φ^0 be a WMTL$_{[a,b]}$ formula. For all $i \geq 0$ let $\varphi^{i+1} = \mathtt{Simp}(\mathtt{Rew}(\varphi^i, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i))$.
If there exists $k \geq 0$ such that $\varphi^k = \mathsf{ff}$ then $\pi \nvDash \varphi^0$.

Proof. First realise that $\pi^{i:} \vDash \varphi$ if and only if $\pi^{i:} \vDash \mathtt{Simp}(\varphi)$ - this is given without proof but should be obvious from the definition of $\wedge$ and $\vee$.

Let $\varphi^k = \mathsf{ff}$ and assume towards a contradicition that $\pi^{0:} \vDash \varphi^0$. Since $\pi^{0:} \vDash \varphi^0$ then $\pi^{1:} \vDash \mathtt{Simp}(\mathtt{Rew}(\varphi^0), \mathtt{P}_0, \mathtt{v}_1 - \mathtt{v}_0)$ thus $\pi^{1:} \vDash \varphi^1$. Since $\pi^{1:} \vDash \varphi^1$ then $\pi^{2:} \vDash \varphi^2 \ldots$ In general we have that, since $\pi^{j:} \vDash \varphi^j$ then $\pi^{j+1:} \vDash \varphi^{j+1}$.

According to the above then $\pi^{k:} \vDash \varphi^k$ but as $\varphi^k = \mathsf{ff}$ then $\pi^{k:} \nvDash \varphi_k$. Contradiction. □

Lemma 10. Let $\pi = (\mathtt{P}_0, \mathtt{v}_0), (\mathtt{P}_1, \mathtt{v}_1), \ldots$ be an infinite weighted word that diverges for every clock used in a WMTL$_{[a,b]}$ formula $\varphi^0 = \psi_1^0 \wedge \psi_2^0$.

For all $i \geq 0$, let

$$\begin{aligned}
\varphi^{i+1} &= \mathtt{Simp}(\mathtt{Rew}(\varphi^i, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i)),\\
\psi_1^{i+1} &= \mathtt{Simp}(\mathtt{Rew}(\psi_1^i, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i)) \text{ and}\\
\psi_2^{i+1} &= \mathtt{Simp}(\mathtt{Rew}(\psi_2^i, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i)).
\end{aligned}$$

If there exists k_1 and k_2 such that $\psi_1^{k_1} = \mathtt{tt}$ and $\psi_2^{k_2} = \mathtt{tt}$ then there exists k such that $\varphi^k = \mathtt{tt}$.

Proof. Let $k_j = min\{k \mid \psi_j^k = \mathtt{tt}\}$ for $j \in \{1, 2\}$ and assume without loss of generality that $k_1 < k_2$. The proof is divided into two parts: We first prove for all j, $0 \leq j < k_1$, that $\varphi^j = \psi_1^j \wedge \psi_2^j$. Secondly we show that $\varphi^{k_1} = \psi_2^{k_1}$ after which the proof is done because we know that $\psi_2^{k_1}$ will be rewritten into $\mathtt{tt}$ - namely when we reach the k_2^{th} rewriting step.

1. For all j, $0 \leq j < k_1$ that $\varphi^j = \psi_1^j \wedge \psi_2^j$.
   This goes by induction in k_1

   Base case, $j = 0$
   Trivial

   Inductive step
   Let $\psi_1^{i+1\prime} = \mathtt{Rew}(\psi_1^i, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i)$ and $\psi_2^{i+1\prime} = \mathtt{Rew}(\psi_2^i, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i)$.
   Then by definition

   $$\varphi^{i+1} = \mathtt{Simp}(\mathtt{Rew}(\varphi^i, \mathtt{P}_i \mathtt{v}_{i+1} - \mathtt{v}_i))$$

   and by induction hypothesis

   $$\varphi^i = \psi_1^i \wedge \psi_2^i$$

   thus

   $$\begin{aligned}
   &\mathtt{Simp}(\mathtt{Rew}(\varphi^i, \mathtt{P}_i \mathtt{v}_{i+1} - \mathtt{v}_i)) =\\
   &\mathtt{Simp}(\mathtt{Rew}(\psi_1^i \wedge \psi_2^i, \mathtt{P}_i \mathtt{v}_{i+1} - \mathtt{v}_i)) =\\
   &\mathtt{Simp}(\mathtt{Rew}(\psi_1^i, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i) \wedge \mathtt{Rew}(\psi_2^i, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i)) =\\
   &\mathtt{Simp}(\psi_1^{i+1\prime} \wedge \psi_2^{i+1\prime}) =\\
   &\psi_1^{i+1} \wedge \psi_2^{i+1}
   \end{aligned}$$

   where the last step stems from the fact that $\mathtt{Simp}(\psi_1^{i+1\prime}) = \psi_1^{i+1} \neq \mathtt{tt}$ and $\mathtt{Simp}(\psi_2^{i+1\prime}) = \psi_2^{i+1} \neq \mathtt{tt}$.

2. $\varphi^{k_1} = \psi_2^{k_1}$

As $\varphi^{k_1-1} = \psi_1^{k_1-1} \wedge \psi_2^{k_1-1}$ then

$$\begin{aligned}\varphi^{k_1} =& \texttt{Simp}(\texttt{Rew}(\varphi^{k_1-1}, \texttt{P}_{k-1}\texttt{v}_{k_1} - \texttt{v}_{k_1-1})) = \\ & \texttt{Simp}(\texttt{Rew}(\psi_1^{k_1-1} \wedge \psi_1^{k_1-1}, \texttt{P}_{k-1}, \texttt{v}_{k_1} - \texttt{v}_{k_1-1})) = \\ & \texttt{Simp}(\psi_1^{k_1\prime} \wedge \psi_2^{k_1\prime})\end{aligned}$$

where $\psi_1^{k_1\prime} = \texttt{Rew}(\psi_1^{k_1-1}, \texttt{P}_{k_1-1}, \texttt{v}_{k_1} - \texttt{v}_{k_1-1})$ and $\psi_2^{k_1\prime} = \texttt{Rew}(\psi_2^{k_1-1}, \texttt{P}_{k_1-1}, \texttt{v}_{k_1} - \texttt{v}_{k_1-1})$. Now, since $\texttt{Simp}(\psi_1^{k_1\prime}) = \psi_1^{k_1} = \texttt{tt}$ it is evident that

$$\texttt{Simp}(\psi_1^{k_1\prime} \wedge \psi_2^{k_1\prime}) = \psi_2^{k_1}$$

□

Lemma 11. Let $\pi = (\texttt{P}_0, \texttt{v}_0), (\texttt{P}_1, \texttt{v}_1), \ldots$ be an infinite weighted word that diverges for every clock used in a WMTL$_{[a,b]}$ formula $\varphi^0 = \psi_1^0 \vee \psi_2^0$.

For all $i \geq 0$, let

$$\begin{aligned}\varphi^{i+1} &= \texttt{Simp}(\texttt{Rew}(\varphi^i, \texttt{P}_i, \texttt{v}_{i+1} - \texttt{v}_i)), \\ \psi_1^{i+1} &= \texttt{Simp}(\texttt{Rew}(\psi_1^i, \texttt{P}_i, \texttt{v}_{i+1} - \texttt{v}_i)) \text{ and} \\ \psi_2^{i+1} &= \texttt{Simp}(\texttt{Rew}(\psi_2^i, \texttt{P}_i, \texttt{v}_{i+1} - \texttt{v}_i)).\end{aligned}$$

If there exists k such that $\psi_1^k = \texttt{tt}$ or $\psi_2^k = \texttt{tt}$ then there exists k' such that $\varphi^{k'} = \texttt{tt}$.

Proof. Similar to proof for Lemma 10. □

Lemma 12. Let $\pi = (\texttt{P}_0, \texttt{v}_0), (\texttt{P}_1, \texttt{v}_1), \ldots$ be an infinite weighted word that diverges for every clock used in a WMTL formula φ^0. For all $i \geq 0$ let $\varphi^{i+1} = \texttt{Simp}(\texttt{Rew}(\varphi^i, \texttt{P}_i, \texttt{v}_{i+1} - \texttt{v}_i))$.

If $\pi^{0:} \vDash \varphi^0$ then there exists $k \geq 0$ such that $\varphi^k = \texttt{tt}$.

Proof. Induction in the structure of φ^0. Let $\varphi^0 =$

- p. Since $\pi^{0:} \vDash \varphi_0$ then $p \in \texttt{P}_0$ thus $\texttt{Simp}(\texttt{Rew}(\varphi_0, \texttt{P}_0, \texttt{v}_1 - \texttt{v}_0)) = \texttt{tt}$ thus $k = 1$.
- $\neg p$. Similar to above.
- $\psi_1^0 \wedge \psi_2^0$. Then $\pi^{0:} \vDash \psi_j^0$ for $j \in \{1, 2\}$ thus by induction hypothesis there exists formulas $\psi_j^1, \ldots$ and k_js such that, for all $i \geq 0$, $\psi_j^{i+1} = \texttt{Simp}(\texttt{Rew}(\psi_j^i, \texttt{P}_i, \texttt{v}_{i+1} - \texttt{v}_i)$ and $\psi_j^{k_j} = \texttt{tt}$.

  By Lemma 10 the conclusion holds.
- $\psi_1 \vee \psi_2$. Similar to above

- $\psi_1 \mathsf{U}^{\mathtt{x}}_{[a,b]} \psi_2$.
  In the following let $\nu_i = \mathtt{v}_{i+1} - \mathtt{v}_i$. We split into cases. Let
  - $a = 0 \wedge \nu_0(\mathtt{x}) < b$.
    Then $\mathtt{Rew}(\varphi^0, \mathtt{P}_0, \nu_0) = \mathtt{Rew}(\psi_2, \mathtt{P}_0, \nu_0) \vee (\mathtt{Rew}(\psi_1, \mathtt{P}_0, \nu_0) \wedge (\psi_1 \mathsf{U}^{\mathtt{x}}_{[0, b-\nu_i(\mathtt{x})]} \psi_2)$.
    Since $\pi \vDash \varphi^0$ and $a = 0$ then either
    - $\pi \vDash \psi_2$.
      Let $\psi_2^j = \mathtt{Simp}(\mathtt{Rew}(\psi_2^{j-1}, \mathtt{P}_{j-1}, \nu_{j-1}))$ for all $j > 0$ and let $\psi_2^0 = \psi_2$. Then by our induction hypothesis there exists a k_2 such that $\psi_2^{k_2} = \mathtt{tt}$.
      Notice now, that $\varphi^1 = \mathtt{Simp}(\mathtt{Rew}(\psi_2, \mathtt{P}_0, \nu_0) \vee (\mathtt{Rew}(\psi_1, \mathtt{P}_0, \nu_0) \wedge (\psi_1 \mathsf{U}^{\mathtt{x}}_{[0, b-\nu_i(\mathtt{x})]} \psi_2))$ and consider that $k_2 = 1$: then by definition of $\mathtt{Simp}$ and $\mathtt{Rew}$, $\varphi^1 = \mathtt{tt}$.

      If $k_2 \neq 1$ then $\varphi^1 = \psi_2^1 \vee \kappa$ where $\kappa = \mathtt{Simp}((\mathtt{Rew}(\psi_1, \mathtt{P}_0, \nu_0) \wedge (\psi_1 \mathsf{U}^{\mathtt{x}}_{[0, b-\nu_i(\mathtt{x})]} \psi_2))$.
      Notice that φ^1 is a disjunction and given our knowledge that $\varphi_2^{k_2} = \mathtt{tt}$ by Lemma 11 the conclusion follows.
    - $\pi \nvDash \psi_2^0$.
      Then there exists a m where $\pi^{m:} \vDash \psi_2$ and for all $i < m$, $\pi^{i:} \vDash \psi_1$.
      To make notation a bit easier in the following we will let $[\kappa]^{j,n} = \mathtt{Simp}(\mathtt{Rew}([\kappa]^{j,n-1}, \mathtt{P}_{j+n-1}, \nu_{j+n-1}))$ for $n \geq 1$ and $[\kappa]^{j,0}] = \kappa$ for any formula κ.
      Notice now that since $\pi \vDash \psi_1$ then by our induction hypothesis there exists a k_1 such that $[\psi_1]^{0,k_1} = \mathtt{tt}$. Thus since

      $$\mathtt{Rew}(\varphi, \mathtt{P}_0, \nu_0) = [\psi_2]^{0,1} \vee ([\psi_1^0]^{0,1} \wedge (\varphi_1 \mathsf{U}^{\mathtt{x}}_{[0, b-\nu_0(\mathtt{x})]})$$

      and due to Lemma 10 and Lemma 11 we only need to prove that there exists a k' where $[(\psi_1 \mathsf{U}^{\mathtt{x}}_{[0, b-\nu_0(\mathtt{x})]} \psi_2)]^{1,k'} = \mathtt{tt}$.
      Let $m = 1$ then $\pi^{1:} \vDash \psi_2$ thus there exists k'' where $[\psi_2]^{1,k''} = \mathtt{tt}$.
      As

      $$\begin{aligned}
      &[(\psi_1 \mathsf{U}^{\mathtt{x}}_{[0, b-\nu_0(\mathtt{x})]} \psi_2)]^{1,1} = \\
      &\mathtt{Simp}(\mathtt{Rew}(\psi_1 \mathsf{U}^{\mathtt{x}}_{[0, b-\nu_0(\mathtt{x})]} \psi_2, \mathtt{P}_1, \nu_1)) = \\
      &[\psi_2]^{1,1} \vee ([\psi_1]^{1,1} \wedge (\psi_1^0 \mathsf{U}^{\mathtt{x}}_{[0, \dot{-}\nu_0(\mathtt{x}) - \nu_1(\mathtt{x})]} \psi_2))
      \end{aligned}$$

      the proof is done.
      If $m \neq 1$ then $\pi^{1:} \vDash \psi_1$ thus there exists k''' where $[\psi_1]^{1,k'''} = \mathtt{tt}$.
      Because

      $$\begin{aligned}
      &[(\psi_1 \mathsf{U}^{\mathtt{x}}_{[0, b-\nu_0(\mathtt{x})]} \psi_2)]^{1,1} = \\
      &\mathtt{Simp}(\mathtt{Rew}(\psi_1 \mathsf{U}^{\mathtt{x}}_{[0, b-\nu_0(\mathtt{x})]} \psi_2, \mathtt{P}_1, \nu_1)) = \\
      &[\psi_2]^{1,1} \vee ([\psi_1]^{1,1} \wedge (\psi_1 \mathsf{U}^{\mathtt{x}}_{[0, \dot{-}\nu_0(\mathtt{x}) - \nu_1(\mathtt{x})]} \psi_2))
      \end{aligned}$$

we only need to show that there exists a k'''' such that

$$[(\psi_1 \mathsf{U}^{\mathtt{x}}_{[0,-\nu_0(\mathtt{x})-\nu_1(\mathtt{x})]}\psi_2)]^{2,k''''} = \mathsf{tt}.$$

This process can continue until we reach the m step where $\pi^{m:} \vDash \psi_2$ after which we are done.

- $a > 0 \wedge \nu_0(\mathtt{x}) < b$. Quite similar to above.
- $a = 0 \wedge \nu_0(\mathtt{x}) > b$. Then since $\pi \vDash \varphi^0$ it must be the case that $\pi \vDash \psi_2$ - this can easily be deduced from the semantics of until and the fact that $b < \nu_0(\mathtt{x}) = \mathtt{v}_1(\mathtt{x}) - \mathtt{v}_0(\mathtt{x}) < \mathtt{v}_j(\mathtt{x}) - \mathtt{v}_0(\mathtt{x})$ for any $j > 1$.
  Since $\pi \vDash \psi_2$ then by induction hypothesis the lemma holds for ψ_2^0 and as $\mathtt{Rew}(\varphi, \mathtt{P}_0, \nu_0) = \mathtt{Rew}(\psi_2, \mathtt{P}_0, \nu_0)$ the lemma holds for φ^0.
- $a > 0 \wedge \nu_i(\mathtt{x}) > b$. Then $\pi \nvDash \varphi$.

- $\psi_1 \mathsf{R}^{\mathtt{x}}_{[a,b]} \psi_2$.
  Quite similar to until case

□

Lemma 13. Let $\pi = (\mathtt{P}_0, \mathtt{v}_0), (\mathtt{P}_1, \mathtt{v}_1), \ldots$ be an infinite weighted word that diverges for every clock used in a WMTL formula φ. Let $\varphi_0 = \varphi, \varphi_1, \ldots$ be a sequence of $\mathrm{WMTL}_{[a,b]}$ formulas such that for all $i > 0$ $\varphi_{i+1} = \mathtt{Simp}(\mathtt{Rew}(\varphi_i, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i))$. If $\pi^{0:} \nvDash \varphi_0$ then there exists $k \geq 0$ such that $\varphi_k = \mathsf{ff}$.

Proof. Similar to the previous lemma. □

Together the previous lemmas easily gives the following main theorem of our technique.

Theorem 5. Let $\pi = (\mathtt{P}_0, \mathtt{v}_0), (\mathtt{P}_1, \mathtt{v}_1), \ldots$ be an infinite weighted word that diverges for every clock used in a $\mathrm{WMTL}_{[a,b]}$ formula φ^0. For all $i \geq 0$ let $\varphi^{i+1} = \mathtt{Simp}(\mathtt{Rew}(\varphi^i, \mathtt{P}_i, \mathtt{v}_{i+1} - \mathtt{v}_i))$. Then there exists $k \geq 0$ such that $\varphi^k = \mathsf{tt}$ if and only if $\pi \vDash \varphi^0$. Similarly, there exists $k \geq 0$ such that $\varphi^k = \mathsf{ff}$ if and only of $\pi \nvDash \varphi^0$.

Example 13. Consider the run

$$(\{a\},\{\tau \mapsto 0\})(\{a\},\{\tau \mapsto 2.5\})(\{b\},\{\tau \mapsto 3\})$$
$$(\{a\},\{\tau \mapsto 3.2\})(\{b,c\},\{\tau \mapsto 5\})(\{a\},\{\tau \mapsto 6\})\ldots$$

and the WMTL$_{[a,b]}$ formula $(a\mathsf{U}_{[0,4]}b)\mathsf{U}_{[0,10]}c$. Our algorithms will produce the following sequence of rewriting rules. The sequence results in tt thus the formula is satisfied by the run.

$$\begin{aligned}(a\mathsf{U}_{[0,4]}b)\mathsf{U}_{[0,10]}c &\xrightarrow{\{a\},\{\tau\mapsto 2.5\}} (a\mathsf{U}_{[0,1.5]}b) \wedge (a\mathsf{U}_{[0,4]}b)\mathsf{U}_{[0,7.5]}c\\ &\xrightarrow{\{a\},\{\tau\mapsto 0.5\}} (a\mathsf{U}_{[0,1.0]}b) \wedge ((a\mathsf{U}_{[0;3.5]}b) \wedge (a\mathsf{U}_{[0,4]}b)\mathsf{U}_{[0,7.0]}c)\\ &\xrightarrow{\{b\},\{\tau\mapsto 0.2\}} (a\mathsf{U}_{[0,4]}b)\mathsf{U}_{[0,6.8]}c\\ &\xrightarrow{\{a\},\{\tau\mapsto 1.8\}} (a\mathsf{U}_{[0,3.2]}b) \wedge ((a\mathsf{U}_{[0,4]}b)\mathsf{U}_{[0,5.0]}c)\\ &\xrightarrow{\{b,c\},\{\tau\mapsto 1\}} \mathsf{tt}.\end{aligned}$$

5 Experiments

Our approach has been implemented in Uppaal SMC. We now illustrate the technique and compare it with the one in [35] that relies on automata-based monitors. If there is no deterministic automaton for the corresponding formula, [35] builds a deterministic under/over approximation that may strongly impact the confidence interval computed by SMC.

5.1 Size of Intermediate Formulas and Precision

Our rewriting rules are recursive in the structure of the formula, which means that the performance of the technique is highly dependent on the size of the intermediate formulas. In the following example, we show how the size of the intermediate formulas vary. We also show that our technique is often much more accurate than the one of [35].

We first study the evolution of the size of the intermediate formula generated by our technique during the monitoring of several randomly generated formulas. We also study the precision of the confidence interval returned by the SMC algorithm in case [35] uses an over or under approximation of the monitor. We also exploit an encoding in Uppaal to show how the size of the formula varies over time for a validation of a single run. In both cases runs are randomly generated by automata. This is done by choosing a delay with respect to an exponential distribution with rate parameter r and after the delay with a discrete probabilistic choice set one of the propositions to true or false.

Formula	r	#U/R	Largest	E_{Max}	σ^2_{Max}	τ_R	τ_M	R_R	R_M
Random1	1	11	14	6.81	3.16	$0.19s$	$5.70s$	738	1748
Random1	4	11	18	7.03	4.92	$0.22s$	$5.83s$	738	1748
Random1	8	11	21	7.06	4.74	$0.23s$	$5.78s$	738	1748
Random2	1	8	17	8.52	5.33	$0.19s$	$6.13s$	738	1748
Random2	4	8	21	11.05	4.71	$0.34s$	$6.17s$	738	1748
Random2	8	8	27	12.79	7.16	$0.58s$	$6.26s$	738	1748
Random3	1	11	21	11.51	4.74	0.50	$10.99s$	738	1748
Random3	4	11	40	13.58	16.53	1.08	$11.06s$	738	1748
Random3	8	11	36	14.00	18.16	1.52	$11.38s$	738	1748

(a)

Formula	#U/R	r	$\%_R$	$\%_M$	R_R	R_M	τ_R	τ_M
random4	15	4	$[0.57; 0.67]$	$[0.57; 0.83]$	738	1748	0.34s	7.77s
random5	15	4	$[0.00; 0.05]$	$[0.00; 0.97]$	738	1748	0.94s	2.83s
random6	15	4	$[0.00; 0.05]$	$[0.00; 0.72]$	738	1748	0.81s	3.18s
random7	15	4	$[0.00, 0.07]$	$[0.00; 0.43]$	738	1748	2.36s	26.61s

(b)

Table B.1: Result of the random formula test. The r column is the rate at which the run was generated. The #U/R column contains the number of until or release modalities that was in the formula. The E_{Max} column is the average largest size of formula and σ^2_{Max} is the variance thereof. τ_M and τ_R is the verification time for the monitoring technique and the rewrite technique, respectively. The verification time for the monitors are the time to construct the monitors and use them - in all the cases the monitors were not exact and both the under and over approximation was used. The R_R and R_M columns contain the number of runs each method required to establish the verification result. The $\%_M$ and $\%_R$ columns refer to the confidence interval obtained by the monitoring and the rewrite process respectively.

Random Formulas

We compute the average size of the largest intermediary formula generated in the rewriting process of different formulas. We verified each formula with a confidence level of 0.05. The results of the test are shown in Table B.1a and Table B.1b. We also give the verification time and the time used in total for the monitor based approach, i.e. both the time to construct the monitor and to verify. The results show that the intermediate formula size depends on the transition rate of the model and as a result so does the validation. The monitor based approach, on the other hand, does not depend on this and the time used remain constant for all the models - due to the most significant part of the monitor based approach is constructing the monitor. However, the rewrite technique is significantly faster than the monitoring technique in all cases. For the results in Table B.1a the monitors are tight approximations thus we gain

time and not precision. However, results in Table B.1b show that we can obtain much more accurate confidence intervals with our new technique. This is due to the monitors might be a large over approximation/a small under approximation. The variance in Table B.1a is rather high due to the runs being random.

Modelling UPPAAL inside UPPAAL

In order to obtain a more in-depth view on how the size of formulas change over time, we have encoded the rules as UPPAAL timed automata. The objective being to use the visualisation features of the tool to see how the number of automata evolve over time. Our construction is recursive in the structure of the formula in the sense that a network of observing automata for φ is obtained as one automaton for φ and at least one automaton for each of the sub-formulas of φ.

The automaton for φ starts its sub-automata through a designated *init*-channel and the sub-automata informs the φ-automaton that their sub-formula has been rewritten to tt or ff through designated channels. The automata for until and release rely on having multiple automata for their sub-formulas that they can start one of after each observation. If there are insufficient sub-automata an error state is reached - because of this the encoding is an under-approximation of the WMTL formula in question.

Example 14. Consider the run

$$(p, \{\tau \mapsto t_0\})(p, \{\tau \mapsto t_1\})(p, \{\tau \mapsto t_2\})(\neg p, \{\tau \mapsto t_3\})(p, \{\tau \mapsto t_4\})$$
$$(p, \{\tau \mapsto t_5\})(p, \{\tau \mapsto t_6\})(p, \{\tau \mapsto t_7\})(?, \{\tau \mapsto t_8\}),$$

where we do not know if the proposition p is true at time t_8 and let $t_8 - t_0 > 10$. In Fig. B.3 we provide a snapshot of the set of active automata at time t_7. At the top we have an automaton that monitors the expression $\Diamond_{[0;10]}\Box_{[4;15]}p$ which has been active since t_0 thus it has $10 - (t_7 - t_0)$ time units left before its expression has been violated.
Below this automaton are automata observing the subexpression $\Box_{[4;15]}p$. These automata have been started at times t_4, t_5 and t_6 respectively and will report tt to the parent automaton at the moment they have oberved p for 15 time units or ff if they observe $\neg p$. Notice that all the automata started before t_4 are no longer active since $\neg p$ was true at time t_3. Also, there is one automaton (the gray one with dashed borders) that is being started by $\Diamond_{[0;10]}\Box_{[4;15]}p$ through its *init*-channel. Since $t_8 - t_0 > 0$ the top level automaton will not start any sub-automata at time t_8. Instead it will merely wait for the already started automata to return either tt or ff. If one of them return tt then the top-level automaton will return tt. In case all of the sub-automata return ff then the top-level automata will return ff.

We encoded the formula $\Diamond_{[0;1]}(p \wedge \Box_{[0;1]}(\neg r) \wedge \Diamond_{[0;1]}(q))$, and put the resulting automata in parallel with an automaton generating random runs and

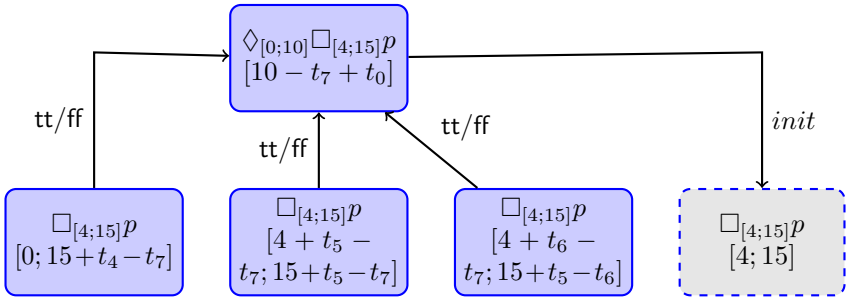
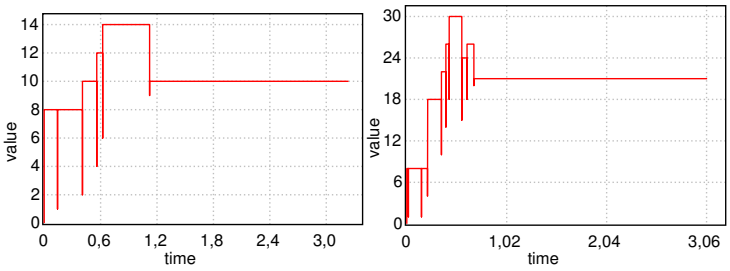


Figure B.3: Snapshot at time t_7 with 3 active automata and one being started.



(a) Simulation of model generating runs with rate 2

(b) Simulation of model generating runs with rate 20

Figure B.4: Plots of how the size of the formula varies over time. On the y-axis is plotted the number of active automata and the x-axis contain the time.

an automaton incrementing a counter whenever an automaton was started or decremented the counter, whenever an automaton stopped. We did this for transition rates 2 and 20 of the random run generating automaton and used the simulate query, `simulate 1 [<=3] {size}`.

In Fig. B.4 we show the plots we obtain for runs generated with varying transition rates. One can easily see that the number of automata does not increase exponentially. We have observed the phenomena on various case studies.

5.2 IEEE 802.15.4 CSMA/CA Protocol

IEEE 802.15.4 standard [113] specifies the physical and media access control layers for low-cost and low-rate wireless personal area networks. Devices operating in such networks share the same wireless medium and can corrupt the transmission of each other by sending data at the same time. We applied our technique to the analysis of Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) network contention protocol that is used in IEEE 802.15.4 to minimise the number of collisions.

Number of nodes	2	3	4	5	6
Monitor-based approach (time)	<1s	3s	57s	20m2s	-
Size of the monitor	230	2049	16306	123800	-
Rewrite-based approach (time)	55s	2m85s	4m11s	6m32s	9m21.47s
Average formula size	8.98	13.76	19.24	24	30.34

Figure B.5: Results for the CSMA/CA protocol

Our objective is to estimate the probability that if a collision occurs, then all nodes participating in it will recover from the collision within a given time bound. This can be specified with $\wedge_{i=1..N}\varphi_i$, where N is a number of network nodes and φ_i specifies the behaviour of a single node:

$$\varphi_i \equiv \Box_{\leq 10000}(\texttt{collision}_i \rightarrow \Diamond_{\leq 4000}\texttt{send}_i)$$

The monitor built by [35] is precise. We are thus not interested to reason on precision of the confidence interval, but rather on the evolution of computation time. In Fig. B.5 we observe that both the size of intermediary formulas used to rewrite φ and the computation time grow linearly as the number of components N increases. On the other hands, both the size of monitor and the computation time with the approach in [35] grow exponentially and cannot be applied to real-life deployments of CSMA.

Although the monitor-based approach is faster for smaller N, for larger N it quickly becomes intractable, while the rewrite-based approach scales well.

6 Conclusion

We presented a new monitoring procedure for WMTL formulas. The technique relies on a series of rewriting step for the formula and is guaranteed to terminate. Contrary to automata-based approaches, ours is precise in the sense that it does not depend on over and under approximation of the formula. We have implemented our approach in Uppaal SMC. Our results outperform those of the monitor-based approaches.

Statistical Model Checking of Dynamic Network of Stochastic Hybrid Automata

Abstract *In this paper we present a modelling formalism for dynamic networks of stochastic hybrid automata. In particular, our formalism is based on primitives for the dynamic creation and termination of hybrid automata components during the execution of a system. In this way we allow for natural modelling of concepts such as multiple threads found in various programming paradigms, as well as the dynamic evolution of biological systems.*

We provide a natural stochastic semantics of the modelling formalism based on repeated output races between the dynamic evolving components of a system. As specification language we present a quantified extension of the logic metric temporal logic (MTL). As a main contribution of this paper, the statistical model checking engine of Uppaal *has been extended to the setting of dynamic networks of hybrid systems and quantified MTL. We demonstrate the usefulness of the extended formalisms in an analysis of a dynamic version of the well-known Train Gate example, as well as in natural monitoring of a MTL formula, where observations may lead to dynamic creation of monitors for sub-formulas.*

1 Introduction

A computer program was originally seen as a single stream of instructions performed in a linear sequence. In contrast, multitasking systems of today have multiple computational threads and the threads executions are interleaved. To complicate matters, computational threads are even allowed to *spawn* other threads. The study of such systems was pioneered by the introduction of process algebras, e.g. CSP [77] and CCS [91]. Process algebras describe the behaviour of systems with a minimal set of primitives and allow us to reason about the equivalence of systems using bisimulation relations. Adding a recursion/replication operator permits expressing spawning of new threads.

Besides having multiple computational threads, modern software is getting more complex due to the distribution of labour: clients connect to servers, servers may delegate work to others and servers may need to contact some

other service. In general, systems may establish connections to other systems and share communication links. The π-calculus [93, 92], an extension of CCS, allows processes to pass communication links to each other. The minimalistic approach of process algebras, however, makes modelling actual systems tedious.

Statistical model checking is a software verification technique that relaxes the requirements to the modelling language. In particular, the state space does not need to be finite as an execution of the model is always terminated at a specific time point - provided the model is time diverging. We will construct a formalism that allows spawning new threads as in process algebras.

In this paper, we present a new modelling formalism founded on the basis of timed IO transition system (TIOTS). The formalism operates on a collection of TIOTSs (templates) that can be instantiated during transitions of active templates. The templates could be generated by any model with semantics given by a TIOTS but in our implementation we rely on Hybrid Automata. We present a stochastic semantics for our formalism based on races between the instantiated components. We develop a specification logic based on MTL [83]. The main difference from MTL is the addition of two operators, one to quantify on the (unknown) number of components of the network, and another to reason on arithmetic operations on this number. We have made an implementation of the modelling formalism and the monitoring technique inside Uppaal SMC [46].

Related Work. Dynamic creation of processes is already part of extensions of process algebras. An example is the fork calculus [67] that extends CCS with a fork primitive. The extensions do not consider quantities and runtime verification of complex requirements expressed in MTL. As said above, the study of dynamical architecture is an intensive research topic. At the software engineering level, several works propose extensions of UML/MODAF/DODAF to handle dynamicity. Those extensions do not rely on a formal semantic which makes run-time verification almost impossible. Additionally, timed and stochastic information are rarely considered in those works. From a more formal perspective, the work of Chen [37] deals with adaptive systems, but again assume that the state space is known in advance. Recently, Sharifloo proposed to avoid this assumption by combining verification and run-time of the deployed system within the Lover framework [111]. This work is in line with our objective, but ignores timed and stochastic aspects. Tools such as BIP have been extended to deal with dynamical architecture [31]. BIP focuses on interactions, while Uppaal proposes a quantitative framework. Other approaches such as PRS also consider dynamical networks. However, they remain at a highly theoretical level, mostly studying what is decidable and what is not [112]. Those approaches do not consider effective and efficient algorithms. Finally, Henzinger et al., have also considered dynamical extension of reactive module with an application to systems biology. The theory presented in [59] remains very complex, there is no run-time monitoring procedure and the verification process is limited to conformance. There are also a wide range of dynamical architectures dedicated to a specific problem [58]. Our approach is more generic and hence incomparable

to those approaches.

2 Dynamic Networks of Hybrid Automata

We introduce a general framework for dynamically evolving networks of real-time components. As the semantical basis of the individual components and the network itself, we use the notion of TIOTS.

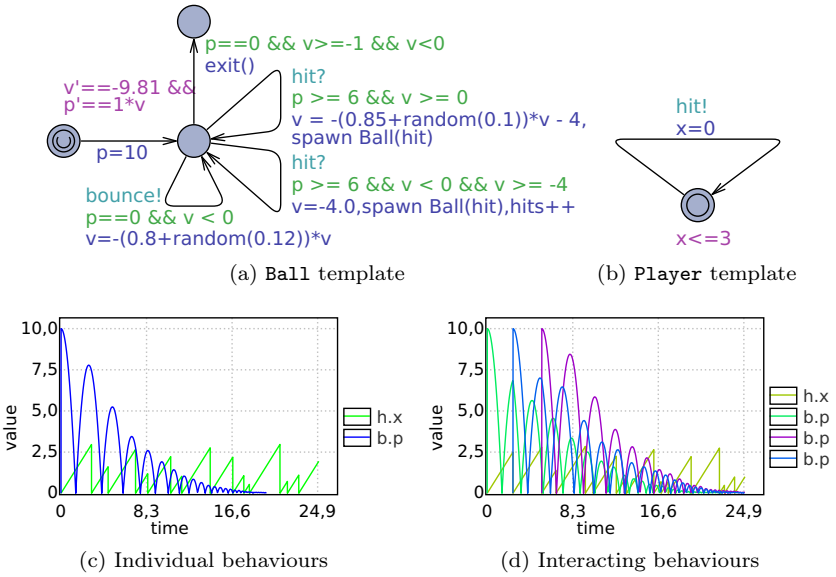
Definition 33 (Timed IO-transition System)**.** A timed IO transition system $\mathcal{K}$ is a tuple $(S, s_0, \Sigma, \rightarrow)$, where

- S is a set of states,
- $s_0 \in S$ is the initial state,
- $\Sigma = \Sigma_{\mathtt{o}} \uplus \Sigma_{\mathtt{i}}$ is a finite set of actions partitioned into inputs ($\Sigma_{\mathtt{i}}$) and outputs ($\Sigma_{\mathtt{o}}$) and
- $\rightarrow \subseteq S \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times S$ is the transition relation.

As usual we write $s \xrightarrow{a} s'$ whenever $(s, a, s') \in \rightarrow$, and $s \xrightarrow{a}$ whenever $s \xrightarrow{a} s'$ for some s'. We call $s \xrightarrow{a} s'$ a discrete (input respectively output) transition whenever $a \in \Sigma$ ($a \in \Sigma_{\mathtt{i}}$ respectively $a \in \Sigma_{\mathtt{o}}$), and a delay transition whenever $a \in \mathbb{R}_{\geq 0}$.

Following the compositional specification theory for timed systems in [44], we assume that a TIOTS, $\mathcal{K}$, is *deterministic*, i.e. whenever $s \xrightarrow{a} s'$ and $s \xrightarrow{a} s''$ then $s' = s''$. We denote by $[s]^a$ the unique state s' such that $s \xrightarrow{a} s'$ (whenever it exists). Also we assume that $\mathcal{K}$ is *input enabled*, i.e. $s \xrightarrow{a}$ for all $a \in \Sigma_{\mathtt{i}}$.

Well-known formalisms for expressing TIOTSs include timed automata [7], priced timed automata [20] and hybrid automata [73]. In these formalisms, states are of the type $(\mathtt{l}, \mathtt{v})$, with $\mathtt{l} \in \mathtt{L}$ being a location of the given automaton, and $\mathtt{v} \in \mathtt{V(X)}$ a valuation assigning values to the various continuous variables of the automaton (e.g. clocks, costs and hybrid variables). A *discrete transition*, $(\mathtt{l}, \mathtt{v}) \xrightarrow{a} (\mathtt{l}', \mathtt{v}')$, corresponds to an edge between $\mathtt{l}$ and $\mathtt{l}'$ in the given automaton, whose guard is enabled by the source valuation $\mathtt{v}$ and where the resulting valuation $\mathtt{v}'$ is obtained from $\mathtt{v}$ by performing the updates required by the edge. In *delay transitions*, $(\mathtt{l}, \mathtt{v}) \xrightarrow{d} (\mathtt{l}, \mathtt{v}')$, the values of the various continuous variables are changed according to a "flow" function $F_{\mathtt{l}} : \mathbb{R}_{\geq 0} \times \mathtt{V(X)} \rightarrow \mathtt{V(X)}$ specified by the location $\mathtt{l}$, i.e. $\mathtt{v}' = F_{\mathtt{l}}(d, \mathtt{v})$. For timed automata $F_{\mathtt{l}}(d, \_)$ simply corresponds to increasing the value of all clocks with d, whereas the "flow" function for hybrid automata are specified using differential equations.



(a) `Ball` template (b) `Player` template

(c) Individual behaviours (d) Interacting behaviours

Figure C.1: Bouncing `Balls` with a `Player`.

Example 15. Consider the variant of the bouncing ball in Fig. C.1. Here a ball is repeatedly bouncing on the floor expressed by the hybrid automaton (template) in Fig. C.1(a). In the model `p` is the height of the and `v` its velocity on the vertical axis. After being initialised to the state $(\mathtt{p} = 10, \mathtt{v} = 0)$ - by the first transition allowed in Fig. C.1a - the following transition sequence may occur:

$$(\mathtt{p} = 10, \mathtt{v} = 0) \xrightarrow{1.02} (\mathtt{p} = 0, \mathtt{v} = -10.00) \xrightarrow{\texttt{bounce!}} (\mathtt{p} = 0, \mathtt{v} = 9.01)$$

where in the `bounce!`-transition the dampening factor has non-deterministically been chosen from the interval $[0.80, 0.92]$ as 0.901. Fig. C.1(b) models an (inexperienced) player that attempts to repeatedly `hit` the ball after non-deterministic delays between 0 and 3. The individual behaviours of the ball and player are illustrated in Fig. C.1(c).

In our framework, a system consists of a dynamically changing number of interacting components, where each component is an instance of a template. The available templates is given by a template collection $\mathcal{J} = (\mathcal{K}_1, \ldots, \mathcal{K}_n)$, describing *closed networks*: all templates have the same action set Σ, and their output action sets provide a partitioning of Σ, i.e. $\Sigma = \cup_{j=1\ldots n} {\Sigma_o}^j$. For $a \in \Sigma$ we denote by $c(a)$ the unique j for which $a \in {\Sigma_o}^j$. For a set A, we shall denote all multisets over A by $\mathcal{M}(A)$. By $X \uplus Y$, we denote the multiset union of two multisets X and Y. Whenever $f : A \to B$, where A and B are sets, we shall extend f to the corresponding multisets in the obvious manner, i.e. for

$$\textsc{Delay}\quad (M_1,\ldots,M_n) \xrightarrow{d} (M_1',\ldots,M_n') \quad \text{if } d \in \mathbb{R}_{\geq 0} \text{ and for all } i \leq n, M_i \xrightarrow{d}_i M_i';$$

$$\textsc{Action}\quad (M_1,\ldots,M_j \uplus \{s\},\ldots,M_n) \xrightarrow{a} (M_1'',\ldots,M_j'',\ldots,M_n'') \oplus P \quad \text{if } a \in \Sigma_j \text{ and } s \xrightarrow{a}{}^{j}_{P} s', M_j'' = M_j \uplus \{s'\}, \text{ and } M_l \xrightarrow{a}_l M_l'' \text{ for } l \neq j.$$

Table C.1: Transition relation for $\boldsymbol{\mathcal{S}}_{\mathcal{J}}$, where $\mathcal{J} = (\mathcal{T}_1,\ldots,\mathcal{T}_n)$ is a template collection.

$X \in \mathcal{M}(A)$, $f(X) = \{\cdot\, f(a) : a \in X \,\cdot\} \in \mathcal{M}(B)$ where $\{\cdot \ldots \cdot\}$ defines a multiset construction.

Definition 34 (Template Collection)**.** Let Σ be a set of actions. A *template collection* over Σ is a tuple $\mathcal{J} = (\mathcal{T}_1,\ldots,\mathcal{T}_n)$, where for $j = 1 \ldots n$, $\mathcal{T}_j = (\mathcal{K}_j, \mathtt{spawn}^j)$ with $\mathcal{K}_j = (S^j, {s_0}^j, \Sigma, \rightarrow^j)$ and:

- $(S^j, {s_0}^j, \Sigma, \rightarrow^j)$ is a TIOTS over Σ with Σ_j as output action set;
- $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ is a disjoint partitioning of Σ;
- $\mathtt{spawn}^j : \Sigma_j \times S^j \rightarrow \mathcal{M}(\{\mathcal{T}_1,\ldots,\mathcal{T}_n\})$ gives for each output-action-state-pair of $\mathcal{K}_j$ a multiset of templates that should be spawned while performing the output action. Whenever $s \xrightarrow{a}{}^{j} s'$ with $a \in \Sigma_j$ and $P = \mathtt{spawn}^j(a,s)$, we write $s \xrightarrow{a}{}^{j}_{P} s'$.

Formally, a template collection $\mathcal{J} = (\mathcal{T}_1,\ldots,\mathcal{T}_n)$ describes a TIOTS $\mathcal{K}_{\mathcal{J}} = (\mathbf{S}_{\mathcal{J}}, \mathbf{s}_0, \Sigma, \rightarrow)$, where all actions are output actions. The set of states $\mathbf{S}_{\mathcal{J}}$ are tuples $(M_1,\ldots,M_n)$ with $M_j \in \mathcal{M}(S^j)$ describing the multiset of states comprising the currently active instances of template $\mathcal{T}_j$. The initial state $\mathbf{s}_0$ is $(\{\cdot {s_0}^0 \cdot\},\ldots\{\cdot {s_0}^n \cdot\})$ i.e. initially one instance of each template is instantiated. The transition relation $\rightarrow$ of $\mathcal{K}_{\mathcal{J}}$ is given by the rules of Table C.1. To delay from a state $\mathbf{s} = (M_1,\ldots,M_n)$, all active instances of all templates must participate in the delay. An a-action transition is driven by an instance of the template $\mathcal{T}_j$ for which a is an output. All other instances of $\mathcal{T}_j$ ignore this output, whereas instances of other templates respond with a corresponding input transition on a. Importantly, instances of the templates in the multiset P are spawned and added to the new configuration. Formally, $(M_1,\ldots,M_n) \oplus P$ is defined inductively in the size of P. As basis $(M_1,\ldots,M_n) \oplus \emptyset = (M_1,\ldots,M_n)$. If $P = P' \uplus \{\mathcal{K}_j\}$ and $(M_1,\ldots,M_n) \oplus P' = (M_1',\ldots,M_n')$, then $(M_1,\ldots,M_n) \oplus P = (M_1',\ldots,M_j' \uplus \{{s_0}^j\},\ldots,M_n')$.

An (infinite) *timed run* over $\mathcal{J}$ is a sequence $\pi = \mathbf{s}_0 d_0 \mathbf{s}_1 d_1 \ldots \mathbf{s}_n d_n \mathbf{s}_{n+1} \ldots$, where for all $i \geq 0$ $\mathbf{s}_i \in \mathbf{S}_{\mathcal{J}}$, $d_i \in \mathbb{R}_{\geq 0}$ and $\mathbf{s}_i \xrightarrow{d_i}\xrightarrow{a_i} \mathbf{s}_{i+1}$ for some $a_i \in \Sigma$. We denote by $\pi^{i:}$ the suffix $\mathbf{s}_i d_i \mathbf{s}_{i+1} d_{i+1} \ldots$.

Remark 7. For readability, we only consider spawning of template instances on output actions. Extending the semantics to also allow spawning on input actions is, however, straightforward. Alternatively, a desired transition $s \xrightarrow{a}{}^{j}_{P} s'$, where a is an input of template $\mathcal{T}_j$, may be encoded as a sequence $s \xrightarrow{a}{}^{j} s^a \xrightarrow{o}{}^{j}_{P}$, with o being a new output action for $\mathcal{T}_j$ and s^a being a new intermediate state that can only output o while spawning P.

Example 16. Reconsider the bouncing ball example from Fig. C.1. Jointly the ball and the player constitutes a template collection $\mathcal{J}$ with two templates (`Ball` and `Player`), with initially one ball and one player. Figure C.1(d) depicts the joint behaviour during the first 25 time-units, with the first transitions detailed as follows:

$$\begin{aligned}
&\big(\{\langle p = 10,\quad v = 0\quad\rangle\}, \{\langle x = 0\rangle\}\big) \xrightarrow{1.02} \big(\{\langle p = 0, v = -10.00\rangle\}, \{\langle x = 1.02\rangle\}\big)\\
&\qquad\xrightarrow{\texttt{bounce!}} \big(\{\langle p = 0, v = 9.01\rangle\}, \{\langle x = 1.02\rangle\}\big)\\
&\qquad\xrightarrow{0.8} \big(\{\langle p = 6.1, v = 1.16\rangle\}, \{\langle x = 1.82\rangle\}\big)\\
&\qquad\xrightarrow{\texttt{hit!}} \big(\{\langle p = 6.1, v = -5.04\rangle, \langle p = 10, v = 0\rangle\}, \{\langle x = 1.82\rangle\}\big)
\end{aligned}$$

In particular, we note that after the (initial) ball have bounced, the player successfully hits it resulting in a new (second) ball being spawned. We see that during the 25 time-units the player is also successful in hitting that (second) ball. In the figure we can see a ball being spawned by the extra curves compared to Fig. C.1(d).

Remark 8. For simplicity our theoretical construction does not allow for parameterising templates. It is, however, allowed in our implementation in Uppaal SMC.

3 Stochastic Semantics for Dynamic Networks

Reconsidering our dynamic version of the bouncing ball from Section 2, we may consider that there is a constant race between the ball(s) `bounce!`ing on the floor and the player `hit!`ing the ball(s). Whereas the time of bouncing is deterministic – given by the ODE obtained from the (stochastic) effect of the previous `bounce!` or `hit!` – the time of the hitting by the player is stochastic according to a uniform distribution in the interval $[0, 3]$. In the randomly generated trajectory of Fig. C.1d it seems that the player was successful in hitting twice, thus generating two additional balls. In fact, a measure on sets of runs of the system is induced, according to which quantitative properties such as *"the probability that there are two or more balls with a height greater than 5 within 4 and 6 time-units"* become well-defined.

Our stochastic semantics is based on the principle of independence between components. Repeatedly each component decides on its own – based on a given

delay density function and output probability function – how much to delay before outputting and what output to broadcast at that moment. Obviously, in such a race between components the outcome will be determined by the component that has chosen to output after the smallest delay: the output is broadcast and all other components may consequently change state.

Stochastic Template Collection Stochastic template collections refine the non-deterministic choices that may exist with respect to delay, output and next state in the specification of a template collection $\mathcal{J} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)$. Let $\mathcal{T}_j = (\mathcal{K}_j, \texttt{spawn}^j)$, be a template of the collection and let S^j denote the corresponding set of states of $\mathcal{K}$. For each state $s \in S^j$, we assume that there exist probability distributions for delays, outputs as well as next-state:

- the *delay density function*, μ_s over delays in $\mathbb{R}_{\geq 0}$, provides stochastic information for when the component will perform an output, thus $\int_{\mathbb{R}_{\geq 0}} \mu_s(t)\,\mathrm{d}t = 1$ [1];
- the *output probability function* γ_s assigns probabilities for resolving what output $o \in \Sigma_{\texttt{o}}{}^j$ to generate, i.e. $\sum_{o \in \Sigma_{\texttt{o}}{}^j} \gamma_s(o) = 1$.

Remark 9. In UPPAAL SMC uniform distributions are applied for states where delay is bounded, and exponential distributions (with location-specified rates) are applied for the cases, where a component can remain indefinitely in a location. Also, UPPAAL SMC provides syntax for assigning discrete probabilities to different outputs as well as specifying stochastic distributions on next-states (using the function `random[b]` denoting a uniform distribution on $[0, b]$).

Stochastic Dynamic Networks A *stochastic* template collection $\mathcal{J} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)$ in turns induces a stochastic semantics of the dynamic network timed IO transition system $\mathcal{K}_{\mathcal{J}} = (\mathbf{S}_{\mathcal{J}}, \mathbf{s}_0, \rightarrow, \Sigma)$. For $\mathbf{s} = (M_1, \ldots, M_n) \in \mathbf{S}_{\mathcal{J}}$, $\mathcal{C}_{\mathcal{J}}(\mathbf{s}, a_1 a_2 \ldots a_k)$ denotes the set of all maximal runs from $\mathbf{s}$ with a prefix $t_1 a_1 t_2 a_2 \ldots t_k a_k$ for some $t_1, \ldots, t_n \in \mathbb{R}_{\geq 0}$ (a cylinder), that is runs where the i'th action a_i has been outputted by some instance of $\mathcal{T}_{c(a_i)}$. Providing the basic elements of a σ-algebra, we now inductively define the measure for such sets of runs:

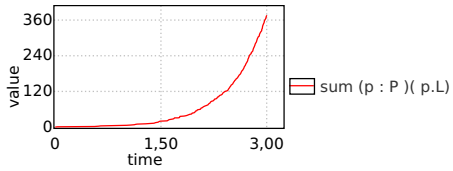
$$\mathbb{P}_{\mathcal{J}}\big(\mathcal{C}_{\mathcal{J}}(\mathbf{s}, a_1 \ldots a_n)\big) =$$
$$\sum_{s \in M_c} \int_{t \geq 0} \mu_s(t) \cdot \left(\prod_{j \neq c} \prod_{s' \in M_j} \left(\int_{\tau > t} \mu_{s'}(\tau)\,\mathrm{d}\tau \right) \right) \cdot \left(\prod_{s'' \in M_c \setminus \{s\}} \left(\int_{\tau > t} \mu_{s''}(\tau)\,\mathrm{d}\tau \right) \right) \cdot$$
$$\gamma_{[s]^t}(a_1) \cdot \mathbb{P}_{\mathcal{J}}\big(\mathcal{C}_{\mathcal{J}}(\big(\big[[\mathbf{s}]^t\big]^{a_1}, a_2 \ldots a_n)\big)\,\mathrm{d}t$$

[1] For outputs happening deterministically at an exact time point d, μ_s becomes a Dirac delta function δ_d.

where $c = c(a_1)$. This definition requires an explanation: at the outermost level we sum over all states from M_c, i.e. active instances of the template $\mathcal{T}_c$ for which a_1 is an output. For a given delay t, the outputting component, s, will choose to make the broadcast at time t with the stated density. Independently, the other components (other components of the template $\mathcal{T}_c$ as well as components of other templates) will choose a delay amount, which – in order for c to be the winner – must be larger than t; hence the (two) products of the probabilities that they each make such a choice. Having decided for making the broadcast at time t, the probability of actually outputting a_1 is included. Finally, the probability of runs according to the remaining actions $a_2 \dots a_n$ is taken into account.

Example 17. Given the dynamic spawning of new instances, the question arises whether the resulting network *explodes* in the sense that discrete actions may occur with shorter and shorter time between them as the number of instances grows, and hence the race between components become more and more intense over time. Stated differently, we worry that the dynamic network may exhibit Zeno behaviour with a non-zero probability. As a (potential) example consider the template P of Fig. C.2, where each instance will spawn new instances according to an exponential distribution with rate 2. The resulting evolution of the number of instances during a random run is illustrated in Fig C.2. Fortunately, it follows from Reuter's criteria for birth-and-death processes [103], that for template collections where all delay densities are either exponential distributions (spanning a finite range of rates) or uniform (spanning a finite range of intervals), the system does not explode. This is important as termination of our method of statistical model checking relies on the assumption that random runs will eventually exceed any given time-bound with probability one.

(a) Template P.



(b) Number of instances.

Figure C.2: An exploding template collection?

4 Dynamic Metric Interval Temporal Logic

In this section we present dynamic metric temporal logic (DMTL) for defining properties of runs of a dynamic network. The logic is based on MTL, where

atomic propositions have been extended with means for quantifying over the dynamic components of the systems.

Let $\mathcal{J} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)$ be a template collection. For each template $\mathcal{T} \in \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$, we assume the existence of a syntactic category of arithmetic expressions, $\textsc{Expr}_{\mathcal{T}}$, interpreted over the states $S^{\mathcal{T}}$ of $\mathcal{T}$. Thus, whenever $\varepsilon \in \textsc{Expr}_{\mathcal{T}}$ and $s \in S^{\mathcal{T}}$ then $[\![\varepsilon]\!](s) \in \mathbb{N}$. Similarly we assume a syntactic category of Boolean expressions, $\textsc{Bool}_{\mathcal{T}}$, interpreted over the states $S^{\mathcal{T}}$ of $\mathcal{T}$. Thus, whenever $\beta \in \textsc{Bool}_{\mathcal{T}}$ and $s \in S^{\mathcal{T}}$ then $[\![\beta]\!](s) \in \mathbb{B}$.

Considering now global states $(M_1, \ldots, M_n)$ of the template collection $\mathcal{J}$, we introduce the sets of arithmetic expressions, $\textsc{Expr}$, and Boolean expressions, $\textsc{Bool}$ given by the following grammars:

$$e ::= c \mid e_1 \,\mathsf{op}\, e_2 \mid \mathsf{sum}(t : \mathcal{T}).\varepsilon_{\mathcal{T}}$$

$$b ::= \mathsf{tt} \mid \mathsf{ff} \mid \neg b \mid b_1 \wedge b_2 \mid e_1 \bowtie e_2 \mid \mathsf{forall}(t : \mathcal{T}).\beta_{\mathcal{T}}$$

where $c \in \mathbb{Z}$, op is a binary arithmetic operator, $\bowtie$ is a binary comparison operator, $\mathcal{T}$ is a template from the collection $\mathcal{J}$, $\varepsilon_{\mathcal{T}} \in \textsc{Expr}_{\mathcal{T}}$ and $\beta_{\mathcal{T}} \in \textsc{Bool}_{\mathcal{T}}$. The semantics are:

- $[\![c]\!](M_1, \ldots, M_n) = \mathrm{c}$
- $[\![e_1 \mathsf{op}\, e_2]\!](M_1, \ldots, M_n) = [\![e_1]\!](M_1, \ldots, M_n) \mathsf{op}\, [\![e_2]\!](M_1, \ldots, M_n)$
- $[\![\mathsf{sum}(t : \mathcal{T}_j).\varepsilon_{\mathcal{T}_j}]\!](M_1, \ldots, M_n) = \sum_{S \in M_j} [\![\varepsilon_{\mathcal{T}_j}]\!](s)$
- $[\![\mathsf{tt}]\!](M_1, \ldots, M_n) = \mathsf{tt}$
- $[\![\mathsf{ff}]\!](M_1, \ldots, M_n) = \mathsf{ff}$
- $[\![\neg b]\!](M_1, \ldots, M_n) = \neg [\![b]\!](M_1, \ldots, M_n)$
- $[\![b_1 \wedge b_2]\!](M_1, \ldots, M_n) = [\![b_1]\!](M_1, \ldots, M_n) \wedge [\![b_2]\!](M_1, \ldots, M_n)$
- $[\![e_1 \bowtie e_2]\!](M_1, \ldots, M_n) = [\![e_1]\!](M_1, \ldots, M_n) \bowtie [\![e_2]\!](M_1, \ldots, M_n)$
- $[\![\mathsf{forall}(t : \mathcal{T}_j).\beta_{\mathcal{T}_j}]\!](M_1, \ldots, M_n) = \bigwedge_{S \in M_j} [\![\beta_{\mathcal{T}_j}]\!](s)$

Definition 35 (Dynamic metric temporal logic)**.** Let $\mathcal{J} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)$ be a collection of templates. A DMTL formula φ over $\mathcal{J}$ is defined by the grammar:

$$\varphi ::= b \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U}_{[x,y]} \varphi_2$$

where $b \in \textsc{Bool}$, $x, y \in \mathbb{Q}$ with $x \leq y$.

We use $\mathsf{exists}(t : \mathcal{T}).\beta_{\mathcal{T}}$ as an abbreviation for $\neg\mathsf{forall}(t : \mathcal{T}).\neg\beta_{\mathcal{T}}$. Commonly used operators of MTL are derived in the usual manners, e.g.: $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 = (\neg\varphi_1 \vee \varphi_2)$, $\Diamond_{[x,y]}\varphi = \mathsf{tt}\, \mathsf{U}_{[x,y]} \varphi$, $\Box_{[x,y]}\varphi = \neg\Diamond_{[x,y]}\neg\varphi$, and $\varphi_1 \mathsf{R}_{[x;y]} \varphi_2 = \neg(\neg\varphi_1 \mathsf{U}_{[x,y]} \neg\varphi_2)$, where R is the "release" operator. For a given timed run $\pi = \mathbf{s}_0 d_0 \mathbf{s}_1 d_1 \ldots \mathbf{s}_n d_n \mathbf{s}_{n+1} \ldots$ over $\mathcal{K}_{\mathcal{J}}$ and a DMTL formula φ, we define satisfaction $\pi^{i:} \models \varphi$ inductively as follows:

1. $\pi^{i:} \models b$ iff $[\![b]\!]\mathbf{s}_i$
2. $\pi^{i:} \models \neg\varphi$ iff $\pi^{i:} \not\models \varphi$
3. $\pi^{i:} \models \varphi_1 \wedge \varphi_2$ iff $\pi^{i:} \models \varphi_1$ and $\pi^{i:} \models \varphi_2$
4. $\pi^{i:} \models \mathsf{X}\varphi$ iff $\pi^{i+1:} \models \varphi$
5. $\pi^{i:} \models \varphi_1 \mathsf{U}_{[x,y]} \varphi_2$ iff there exists $j \geq i$ such that $\pi^{j:} \models \varphi_2$ and $\sum_{k=i}^{j-1} d_k \in [x, y]$ and $\pi^{k:} \models \varphi_1$ whenever $i \leq k < j$.

We say that a timed run π satisfies φ if $\pi^{0:} \models \varphi$. We say that a template collection $\mathcal{J}$ satisfies φ, $\mathcal{J} \models \varphi$, iff all timed runs of $\mathcal{K}_{\mathcal{J}}$ starting in $\mathbf{s}_0$ satisfies φ. Given the stochastic semantics of $\mathcal{J}$, we define $\mathbb{P}_{\mathcal{J}}(\varphi)$ to be the probability that a random run of $\mathcal{J}$ satisfies φ. As we shall see later, this probability is well-defined as it may be characterised as a countable union and intersection of cylinders over $\mathcal{J}$ extended with a monitor $\mathcal{M}_\varphi$ for φ, and is thus measurable.

Example 18. Reconsider the bouncing ball from Fig. C.1. The property *"there are two or more balls with a height greater than 5 within 2 and 3 time-units"* may be expressed as the DMTL formula $\Diamond_{[2,3]}\big(\mathsf{sum}(\mathtt{b} : \mathtt{Ball})(\mathtt{b.p} > 5)\big) \geq 2$. Similarly, the property *"for any time-point within 1 and 3 time units, all balls have height less than or equal to 4"* corresponds to the formula $\Box_{[1,3]}\mathsf{forall}(\mathtt{b} : \mathtt{Ball})(\mathtt{b.p} \leq 4)$. Using UPPAAL SMC, we find $[0.164092, 0.264092]$, resp. $[0.82, 0.92]$, to the interval of the probability that a random run will satisfy the first, resp. the second, property with 95% confidence.

Theorem 6. Let $\mathcal{J}$ be a template collection and φ be a DMTL formula . Then there exists a template collection $\mathcal{J}_\varphi = \{\mathcal{K}_\varphi, \mathcal{K}_{\varphi_1}, \ldots, \mathcal{K}_{\varphi_n}\}$ over Σ_ϕ with $\mathtt{tt}_\varphi!, \mathtt{ff}_\varphi! \in \Sigma_\phi$ associated with φ such that:

$$\mathbb{P}_{\mathcal{J}}(\varphi) = \mathbb{P}_{\mathcal{J}\cup\mathcal{J}_\varphi}\left(\bigcup_{\omega\in\Sigma^*} \pi\left((\mathbf{s}_0, {\mathbf{s}_0}^\varphi), \omega\mathtt{tt}_\varphi!\right)\right) \tag{C.1}$$

where ${\mathbf{s}_0}^\varphi = (M_\varphi, M_{\varphi_1}, \ldots M_{\varphi_n})$ with $M_\varphi = \{{s_0}^\varphi\}$ and for all $j \neq 0$, $M_{\varphi_j} = \emptyset$, and Σ is the combined alphabet of $\mathcal{J}$ and $\mathcal{J}_\varphi$ excluding $\mathtt{tt}_\varphi!$ and $\mathtt{ff}_\varphi!$.

5 Dynamic Networks of Hybrid Automata in UPPAAL

UPPAAL SMC has been extended to dynamic instantiation of templates. Templates are, as usual, defined as stochastic hybrid automata (SHAs). The extension includes extending the core language with two keywords (spawn and exit) to create and terminate processes, extending the notion of ranges to sets of processes using a similar syntax, and extending MTL to refer to the dynamic processes. We mention some interesting details that are important.

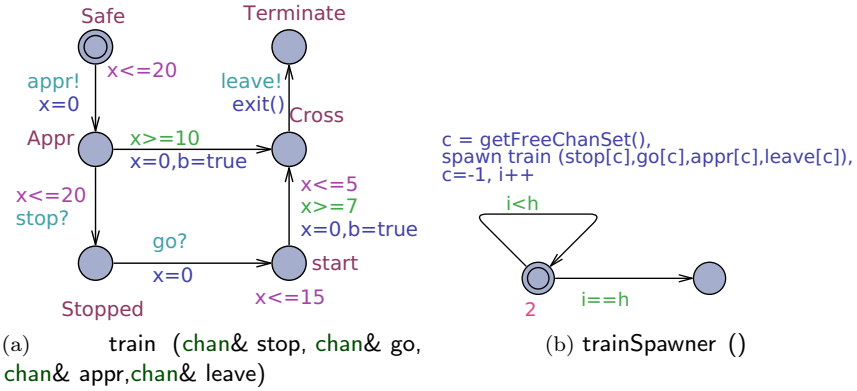
Core Language Extensions. New processes are created by calling spawn T(args...) where T is the name of a template. The argument list is similar to the ordinary process instances. However, the internal mechanism for dynamically creating processes is different. A dynamic process terminates by calling the special function exit().

Normally, an instance corresponds to a template with its arguments substituted that is then compiled to obtain some byte-code specific for that instance. This is too expensive for dynamic instances not to mention the serious issue concerning memory management when terminating an instance (and the byte-code in the engine too). The mechanism we have creates local variables that correspond to the instance parameters of these templates. The templates are instantiated when needed but are recycled and kept in the engine when they terminate, in particular between each run. Instantiating a dynamic instance (spawn) is done by taking a process in a pool, writing to its local variables (instead of substituting and generating new byte-code), and adding it to the state. Termination of a process (exit) corresponds to recycling.

Sets of Processes. The language extends the notion of ranges to sets of processes. The processes are here instances of stochastic hybrid automata templates. Normally, the user can use `for` loops, and the statements sum, **forall**, and **exists** over ranges, e.g., **for**(i:id_t) for iteration purposes. Now, this is extended to sets of processes for a given template type. For a template of type T, it is now possible to use the syntax t:T to iterate over this set with, e.g., sum(t:T) t.count.

Statistical model checking (SMC) We use SMC [87, 108, 121, 123] to estimate and test on the probability that a random run of a network of stochastic hybrid automata satisfies a given property. Given a model $\mathcal{H}$ and a trace property φ, SMC refers to a series of simulation-based techniques that can be used to answer two main questions: (1) *Qualitative:* is the probability that a random run of $\mathcal{H}$ satisfies φ greater or equal to a certain threshold θ? and (2) *Quantitative:* what is the probability that a random run of $\mathcal{H}$ satisfies φ? In both cases, the answer will be correct up to a user-specified level of confidence, providing a upper bound on the probability that the conclusion made by the algorithm is wrong. For the quantitative approach, the method computes a confidence interval that is an interval of probabilities that contains the true probability to satisfy the property. Here the confidence level provides the probability that the computed confidence interval indeed contains the unknown probability.

Implementation of DMTL Monitoring. When checking DMTL properties, the engine rewrites the DMTL formula φ on-the-fly in such a way that the new formula φ_1 is satisfied from the next state if and only if φ was satisfied from current state. The technique is similar to the one presented in [34], but the engine has been extended to check the construct $\mathsf{forall}(t : \mathcal{T}).\beta_{\mathcal{T}}$.



(a) train (chan& stop, chan& go, chan& appr,chan& leave)

(b) trainSpawner ()

Figure C.3: Automaton of a train (a) and an automaton for spawning trains (b)

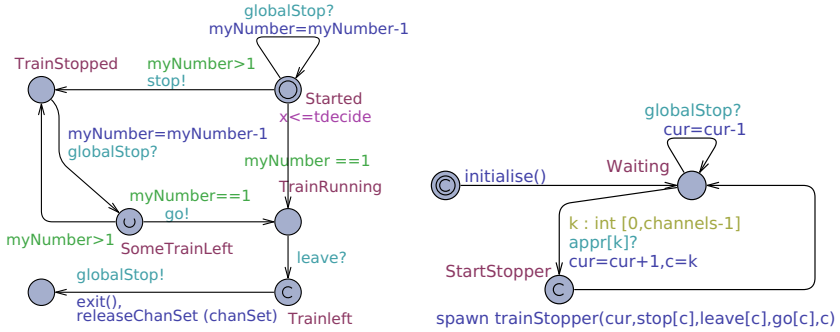
Visualisation of Simulated Runs. As part of the SMC analysis output, plots are very useful tools. When generating processes dynamically, we need a new way to ask for plots on such dynamic instances. To do this, the special statement `foreach(t:T) expr` is used to tell the model-checker to generate plots for each of the expressions `expr` that depend on `t`. An example of such a plot is shown in Figure C.1d.

6 Dynamic Train Gate

We consider a dynamic version of the train gate example distributed with UPPAAL: a gate controls the access to the one rail over a bridge. When a train approaches the bridge it signals the gate that it is approaching. Depending on the speed of the train, the train will proceed onto the crossing after 10 and before 20 time units. The actual crossing takes 5 time units. From the approach signal has been sent, the gate can stop the train to avoid a collision. When stopped, the train can be restarted which takes between 7 and 15 time units. Afterwards the train enters the bridge and cannot be stopped. To ensure only one train crosses at a time, the gate allows the train arriving first to proceed and queue the latter. When a train leaves the crossing, the gate will start the first train in the queue.

In our model each train is modelled as a single SHA, depicted in Figure C.3a. The train-SHA has four parameters, namely the channels leave,stop,go and appr. The trains are spawned according to an exponential distribution with rate 2 by the SHA in Figure C.3b. This uses a function to acquire channels, from a pre-allocated set of channels, to be used by the spawned train.

The gate, see Figure C.4b, listens on all channels that can be used as appr channels and when a train signals its approach a train stopper, depicted in Figure C.4a, is spawned . The train stoppers are given a number (myNumber) that they decrease when a train has left the crossing (signalled by a synchronisation



(a) trainStopper (int myNumber, chan &stop, chan &leave, chan &go)

(b) gate()

Figure C.4: Model of the queue (a) and the gate controller (b)

on globalStop?) . When myNumber is equal to one, the train stopper broadcasts go! to its associated train. Whenever a train broadcasts leave! the associated train stopper will broadcast globalStop!. This results in starting a stopped train. In this manner the train stoppers are encoding a queue. In the original model the queue was encoded explicitly in the C-like language of Uppaal but as this does not allow dynamic allocation of memory we resorted to using templates. Initially the only instances in the system are the gate and train spawner.

Experiments Figure C.5 shows the number of trains in each of their possible location. This plot was obtained by running the Uppaal SMC query:

```
simulate 1
[<=300] {sum (t:train)(t.Stopped),sum
        (t:train)(t.Safe)...}
```

We see in the run that only one train is on the bridge at a time (i.e. in the location Cross). We also see that the trains are spawned at the beginning of the run, resulting in a high number of trains being stopped. At some point,approximately about 40 time units into the run, all the trains have been spawned and the number of stopped trains decreases, as the trains are moved one by one to the crossing.

h	tdecide	Probability
20	10	[0.000; 0.097]
20	12	[0.091; 0.191]
20	15	[0.393; 0.494]
40	10	[0.000; 0.0974]
40	12	[0.156; 0.255]
40	15	[0.701; 0.801]

Table C.2: Probability of collision within 110 time units with h trains and a decision time of tdecide

In the model h trains are spawned by the the template in Figure C.3b with some delay in between. The gate is supposed to ensure only one train crosses the bridge at a time, and in order to do so it relies on the train stopper templates. The train stoppers need some time, tdecide, to decide if a train should be stopped or not. Using Uppaal SMC we can find the probability that two

trains are on the bridge simultaneously within 1100 time units through the query:

```
Pr[<=X] (<> exists (t :  train) (t.Cross && exists (p :
                train)(p.Cross && p!=t)))
```

The delayed reaction from the train stoppers may, obviously, result in the gate not being safe. The results in Table C.2 suggests, however, that if the time the train stoppers need is less than the minimum time for the train to enter the bridge then the probability is low.

7 Experiments with the Monitoring of DMTL

For experimenting with the MTL monitoring stated in Theorem 6 we have created an automaton that generate random runs over the propositions p, q and r. The automaton "flips" the truth assignment of p,q or r with a rate of 2 i.e. the delays between state changes is extracted from an exponential distribution with rate parameter 2.

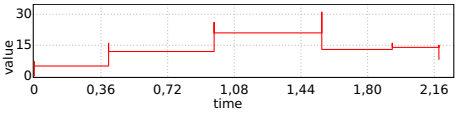


Figure C.6: The number active observers during a verification of the formula $\Diamond_{[0;1]}(p \wedge \Box[0;1](\neg r) \wedge \Diamond_{[0;1]}(q))$

The advantage of "implementing" the rewrite technique as observers is that we can use the plotting feature of Uppaal SMC to obtain plots of the number of active observers during a simulation. This can be done with a query in the style of

```
Pr[<=10]{sum (b :  T1) (1) +sum (b :  T2) (1) ...}.
```

We show such a plot in Figure C.6. Obtaining this plot with the optimised version in Uppaal SMC is more difficult because you would have to explicitly gather information during the rewriting. In addition to obtain such plots, we can experiment with variations of the rewrite technique. It is well-known, that for run time verification it is not feasible to observe a system every time it changes

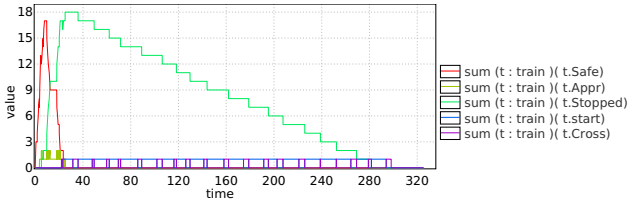


Figure C.5: The number of trains in each location. The number of trains in this run is 20 and the trains stoppers time to decide (tdecide) is 0

state. Instead the system is observed at distinct time points. This can result in different verification results thus we are interested in knowing how the verification result differs when using fewer observations. To exemplify this we have created three collections of observers for the formula $\Diamond_{[0;1]}(p \wedge \Box[0;1](\neg r) \wedge \Diamond_{[0;1]}(q))$ and parallel composed them with the random automata previously described: The first one, $\mathcal{J}_\phi^1$, observes every state change of the system. The second one $\mathcal{J}_\phi^2$ observes the system at time points randomly selected - with the restriction that there should be one observation each time unit. The third, $\mathcal{J}_\phi^3$, is similar to the second, but the maximal time between observations is 2.

Verifying the property with the different observer setups gives probability bounds $[0.02, 0.12]$ using $\mathcal{J}_\phi^1$ and $[0.03, 0.13]$ for $\mathcal{J}_\phi^2$. For $\mathcal{J}_\phi^3$ we obtain a probability bound $[0.00, 0.10]$. These results are not surprising, but they exemplify the possibility of using Uppaal SMC to make an analysis of a run validation technique.

8 Conclusion & Future Work

We have presented a formalism that allows for dynamical instantiation of templates of hybrid automata. The formalism is given a natural stochastic semantics based on repeated output races. Also, we have extended MTL with the possibility of quantifying over components at the propositional level. In particular, the engine of Uppaal SMC has been extended to allow for statistical model checking of DMTL properties for dynamic networks of stochastic hybrid systems. Also we added additional visualisation option to Uppaal SMC.

Future extension involve extension to more advanced specification formalism in which we allow for nesting the quantifications over components.

Quantified Dynamic Metric Temporal Logic for Dynamic Networks of Stochastic Hybrid Automata

D

Abstract *Multiprocessing systems are capable of running multiple processes concurrently. By now such systems have established themselves as the defacto standard for operating systems. At the core of an operating system is the ability to execute programs and as such there must be a primitive for instantiating new processes - also programs are allowed to die/terminate. Operating systems may allow the executing programs to split (spawn) into more computational threads in order to let programs take advantage of concurrent execution as well. One of the most used modelling languages, Timed Automata, is based on multiple automata interacting thus they easily model the concurrent execution of programs. However, this language assumes a fixed size system in the sense that automata cannot be created at will but must be instantiated when the overall system is created. This is in contrast with the fact that developers are able to create threads when needed. In this paper we present our continued work to incorporate spawning of threads into* Uppaal SMC*. Our modelling language, Dynamic Networks of Stochastic Hybrid Automata, is essentially Timed Automata extended with a spawning primitive and a tear-down primitive. The dynamic creation of threads has the side-effect that it is no longer possible to use ordinary logics to specify behaviours of individual threads - because the threads no longer have unique names. In this paper we propose an extension of* Metric Temporal Logic *with means for quantcifying over the dynamically created threads. This makes it possible to zoom in on individual threads and specify requirements to their future behaviour. Furthermore, we present a monitoring procedure for the logic based on rewriting formulas. The presented modelling language and the specification language have been implemented in* Uppaal SMC *version 4.1.18.*

1 Introduction

Computer systems of today are beyond the state where they were statically encoded entities that were disconnected from their surroundings. Instead many

software architectures are build with communication to other systems in mind thus each system may rely on other systems for accomplishing their tasks. The systems providing services to other systems must incorporate concurrency into their execution platforms in order to handle requests from multiple clients simultaneously - and to support an unknown number of clients they must be able to make new computational threads at will. Luckily, most mainstream programming languages and operating systems support concurrency out of the box and alleviate the programmer from the burden of programming the concurrency model.

Reasoning on dynamic systems poses a major challenge to the formal methods community, that is the one of being able to develop models and techniques for systems whose state-space is not known a priori. Additionally, it also requires to deal with communication between processes at run time. Process algebras, e.g. CSP [77] and CCS [91] have been designed to analyse such systems. In process algebras the behaviour of systems is described with a minimal set of primitives and they allow us to reason about the equivalence of systems using bisimulation relations. By adding a recursion/replication operator we can express spawning of processes. Whereas process algebras have been developed for reasoning about dynamic systems, we note that few formal tools support dynamic creation of processes. Instead, they require specifying all processes in advance which forces the modeller to encode an underlying resource manager with a preset finite capacity. This stands in deep contrast to the support given by operating systems and programming languages.

In a recent work [52], we developed a modelling framework that allowed spawning inside Uppaal SMC [45]. In our setting, processes are spawned from a finite set of templates, each being an arbitrarily complex input/output timed system. In this setting, processes communicate via an input/output mechanism of actions, and each of those actions may eventually lead to the creation of one or several new processes. The model checking problem is known to be undecidable for such systems. As a solution, we proposed to equip our system with a stochastic semantic, allowing us to unleash the power of simulation-based solutions such as statistical model checking (SMC) [123]. In this framework, verification reduces to monitor several executions of the system and then use an algorithm from statistics to deduce the overall correctness with a controllable confidence. One of the drawbacks of the work in [52] is that the logic for specifying properties of template systems was a limited extension of metric temporal logic (MTL) [83], where quantifications over processes were allowed at the atomic level. While such a logic is powerful enough to express properties like: “at any time, all the processes shall avoid state x”, it does not allow to deal with more complex requirements such as “if there exists a process that reaches state q, then it should reach state q’ in less than ten units of time after reaching q”.

In this paper, we propose an extension of MTL for dynamic timed systems, where quantifications over process templates can be nested. Our new quantified dynamic metric temporal logic (QDTML) is inspired by those proposed to

specify properties of infinite state systems, especially in the context of the so-called regular model checking approach [1, 25]. As a second contribution, we present a monitoring procedure for this new logic. The procedure uses rewriting techniques of subformulas of the logic. Our work has been implemented in Uppaal SMC, the SMC extension of Uppaal.

Related Work. Dynamic creation of processes is already part of extensions of process algebras. An example is the fork calculus [67] that extends CCS with a fork primitive. These extensions do not consider quantities and run time verification of complex requirements expressed in MTL. Recently, Sharifloo proposed to avoid this assumption by combining verification and run-time of the deployed system within the Lover framework [111]. This work is in line with our objective, but ignores timed and stochastic aspects. Tools such as BIP have been extended to deal with dynamical architecture [31]. BIP focuses on interactions, while Uppaal SMC proposes a quantitative framework. Other approaches such as PRS also consider dynamical networks. However, they remain at a highly theoretical level, mostly studying what is decidable and what is not [112]. Those approaches do not consider effective and efficient algorithms. Finally, Henzinger et al., have also considered dynamical extension of *reactive modules* with an application to systems biology. The theory presented in [59] is without a run-time monitoring procedure and the verification process is limited to conformance. There are also a wide range of dynamical architectures dedicated to a specific problem [58]. Our approach is more generic and hence incomparable to those approaches. Dynamic process creation is already part of the model checking tool SPIN[79]. Contrary to our work, SPIN does not consider *timed*, *hybrid* or *stochastic* aspects of systems. Boudjadar et al. [26] have developed a framework called Callable Timed Automata (CTA) that allows dynamic creation of processes. Our work distinguishes itself from theirs by having a stochastic semantics and their work did not consider a logic for expressing properties of the dynamic systems.

2 Client-Server Example

The purpose of this section is to provide the intuition behind our modelling formalism by means of an example. The example we consider is that of a client-server system shown in Figure D.1. Due to *Traffic* we have clients arriving in the network (1). These clients connect to a server (2) that generates threads (3) to handle the subsequent communication with the clients (4). When the exchange is over, the client and corresponding server threads terminate. This models the typical behaviour of servers that *listen* on a port, *accept* a connection, and delegate the connection to a forked process or a new thread while returning to listening on its port. We aim towards only giving the intuition behind our formalism in this section and leave the actual semantics for later. In short, a model consists of *templates* that we can instantiate to running processes. A

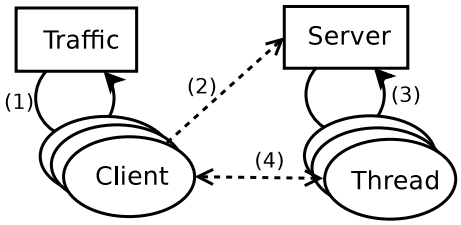


Figure D.1: A client-server example.

process is either instantiated as an initialisation step of the model, or by being spawned by a running process [1].

The model has four templates shown in Fig. D.2 through D.5.

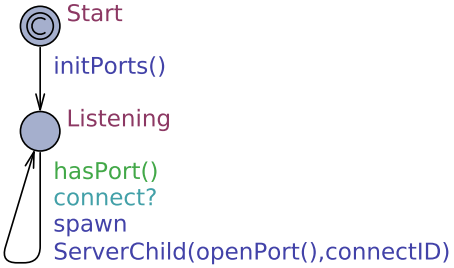


Figure D.2: Server()

Server. Figure D.2 shows the template modelling the server. To model that servers can accept only a limited number of connections, the server manages its available connection ports with an array. The first transition from Start initialises this array with initPorts(). The server awaits a connection request from a client with a channel synchronisation on connect?. The server reacts only if it has some available port (condition hasPort()) and spawns a server child. This is done with the spawn command that instantiates the ServerChild template (modelling a thread) with argument values computed on-the-fly. The server allocates a port with the function openPort() and forwards connectID that it receives from the client to the server child.

ServerChild. The template taking care of the connections is shown in Fig D.3. An instance of this template starts by taking some time to reply and accept the connection. The time is picked with an exponential distribution with rate 5. Then the instance synchronises back with the client that initiated the connection and "sends" the allocated port number with the synchronisation accept[port]! [2]. Here connectID is used to filter out the right client. The location

[1] In the same manner as you spawn processes in computer systems

[2] The trick uses an array of channels for message passing.

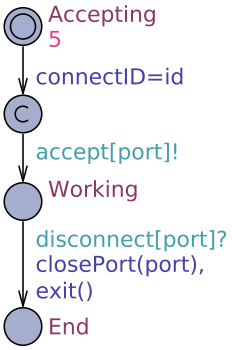


Figure D.3: ServerChild(**const** pid_t port, **const** int id)

Working abstracts from the actual communication until the client closes it, which is done by the synchronisation disconnect[port]?. The server child closes the port (makes it available again) and terminates, which is done with the special function exit().

Figure D.4: Traffic ()

Traffic. To model traffic, the template of Fig. D.4 generates clients, i.e., *spawns* client processes with the expression spawn Client(++clientID). The time between creation is picked with an exponential distribution with rate 10. Each new client receives a unique identifier.

Client. When spawned, the client of Fig. D.5 will take some time to connect (exponential distribution with rate 10). It will then wait for the synchronisation accept[p]? that passes the port. The client tests if the reply matches its ID with id==connectID. This is needed since we abstract from the actual communication protocol. The client times-out after 5 time units and will retry MAX_RETRIES times before aborting. If the connection is accepted then the client works for some time, then disconnects with disconnect[port]! and terminates. Here again the client process terminates by calling exit().

Support for Dynamic Processes. When the special command spawn is encountered, UPPAAL SMC creates a new instance of a given template with the current values of the expressions used for arguments. When the special function exit() is executed in a dynamic process, UPPAAL SMC discards this process. The templates that can be instantiated dynamically are declared to be *dynamic* in

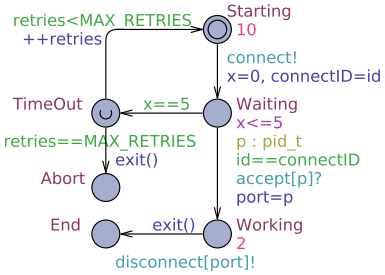


Figure D.5: Client(**const** int id)

the global declaration of the model as shown in Listing D.1. We also show the functions for opening and closing ports.

```
pid_t openPort() {
  assert(freePort>=0); return ports[--freePort];
}
void closePort(pid_t p) {
  assert(freePort<MAX_CONNECTIONS);
  ports[freePort++] = p;
}

dynamic ServerChild(const pid_t port, const int id);
dynamic Client(const int id);
```

Listing D.1: Global declarations of the client-server example

3 Dynamic Network of Hybrid Systems

In this section we provide the semantical part of our modelling framework. The framework is equivalent to the one we presented in [52] but the semantics are expressed differently: in our previous work all processes were anonymous whereas we in this work give them identities by giving them a name. We later use these names in defining the semantics of our logic.

We abstract from the actual modelling formalism and define our semantics on the basis of timed IO transition system (TIOTS).

Definition 36 (Timed IO-transition System)**.** A timed IO transition system over the input actions $\Sigma_{\mathtt{i}}$ and output actions $\Sigma_{\mathtt{o}}$ is a tuple $(S, s_0, \rightarrow)$ where

- S is a set of states,
- $s_0 \in S$ is the initial state and
- $\rightarrow \subseteq S \times (\Sigma_{\mathtt{i}} \cup \Sigma_{\mathtt{o}} \cup \mathbb{R}_{\geq 0}) \times S$ is the transition relation.

Let $(S, s_0, \rightarrow)$ be a TIOTS then we write $s \xrightarrow{a!} s'$ whenever $(s, a, s') \in \rightarrow$ and $a \in \Sigma_{\mathtt{o}}$. Similarly we write $s \xrightarrow{a?} s'$ if $a \in \Sigma_{\mathtt{i}}$ and $s \xrightarrow{d} s'$ if $d \in \mathbb{R}_{\geq 0}$.

In accordance with the compositional specification for timed systems [44] we assume any TIOTS is input-enabled i.e. for any state s and any input action a there exists s' s.t. $s \xrightarrow{a?} s'$. Also we assume determinism thus if $s \xrightarrow{x} s'$ and $s \xrightarrow{x} s''$ then $s' = s''$. Since we assume determinism we denote the $x-$successor, $x \in (\mathbb{R}_{\geq 0} \cup \Sigma_{\mathtt{i}} \cup \Sigma_{\mathtt{o}})$ of s by $[s]^x$ i.e. $s \xrightarrow{x} [s]^x$.

Timed IO transition systems can be generated by various formalisms. Well-known formalisms include Timed Automata [7] and Hybrid Automata [73], where states have the form $(\mathtt{l}, \mathtt{v})$ where $\mathtt{l}$ is a control location of the automaton and $\mathtt{v}$ is a valuation that assigns values to continuous variables e.g. clocks, costs and hybrid variables. A *discrete* transition from $(\mathtt{l}, \mathtt{v})$ to $(\mathtt{l}', \mathtt{v}')$ corresponds to an edge, in the automaton, between $\mathtt{l}$ and $\mathtt{l}'$ whose guard is enabled by $\mathtt{v}$. The resulting $\mathtt{v}'$ is obtained by performing the updates required by the edge. In *delay* transitions, the values of the continuous variables are changed according to a flow function that gives the rate of change for each clock. For a timed automaton this rate is always 1 hence a delay of d would increase all variables by d. For hybrid systems the rates are specified using differential equations.

Example 19. Consider the model of a client attempting to establish a connection to a server shown in Figure D.5. This timed automaton has one clock `x` with a starting value of 0. The client is initially in the location `Starting`. From this initial state a possible transition sequence is:

$$(\mathtt{Starting}, \mathtt{x} = 0) \xrightarrow{0.8} (\mathtt{Starting}, \mathtt{x} = 0.8) \xrightarrow{\mathtt{connect!}} (\mathtt{Waiting}, \mathtt{x} = 0)$$

In our framework a system consists of a dynamically evolving set of processes, where processes are running instances of templates and can spawn other processes. The available set of templates that a process can be spawned as is given in terms of a *Template Collection* $\mathcal{J} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)$ where each template defines a TIOTS. All the templates share a common set of actions, Σ. This set is partitioned into disjoint subsets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ and template $\mathcal{T}_i$ uses Σ_i as output actions and $\Sigma \setminus \Sigma_i$ as input actions.

Definition 37 (Template Collection)**.** A Template Collection over the set of actions Σ partitioned into n disjoint sets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ is a tuple $(\mathcal{T}_1, \ldots, \mathcal{T}_n)$ where for all i, $\mathcal{T}_i = (S^i, {s_0}^i, \rightarrow^i, \mathtt{spawn}^i)$ with:

- $(S^i, {s_0}^i, \rightarrow^i)$ is a TIOTS with output action Σ_i and input actions $\Sigma \setminus \Sigma_i$,
- $\mathtt{spawn}^i : S^i \times \Sigma_i \rightarrow 2^{\{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n\}}$ describes for each state-action-pair a set of templates that should be spawned while doing that action from that state.

As mentioned earlier the semantics is based on a dynamically evolving set of processes. When spawning a process, on the basis of a template $\mathcal{T}$, a name

p for the new process is extracted from an infinite but countable set of names PNames, and the global state is updated to point p to the initial state of $\mathcal{T}$. Furthermore, in the global state we record that p has type $\mathcal{T}$ and record that the name p has been used.

Formally, a state of a template collection $(\mathcal{T}_1, \ldots, \mathcal{T}_n)$ has the form $(\mathsf{Active}, \mathtt{T}, \mathtt{Sm})$, where

- $\mathsf{Active} \subseteq \mathsf{PNames}$ contains the names that has been bound to form a process,
- $\mathtt{T} : \mathsf{Active} \to \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ gives the template type of each process and
- $\mathtt{Sm} : \mathsf{Active} \to \bigcup_{i=1}^{n} S^i$ maps the bound names to their corresponding state.

Naturally, we require *consistency* in the sense that if $\mathtt{Sm}(p)$ points to a state of $\mathcal{T}_i$ then $\mathtt{T}(p) = \mathcal{T}_i$. For the actual spawning of processes we define an operator $\oplus$ defined between a state and a template.

$$(\mathsf{Active}, \mathtt{T}, \mathtt{Sm}) \oplus \mathcal{T} = (\mathsf{Active} \cup \{p\}, \mathtt{T}', \mathtt{Sm}'),$$

where $\mathcal{T} = (S, s_0, \to, \mathtt{spawn})$, $p \in \mathsf{PNames} \setminus \mathsf{Active}$, $\mathtt{Sm}'(p) = s_0$, $\mathtt{T}'(p) = \mathcal{T}$ and for all $p' \in \mathsf{Active}$, $\mathtt{Sm}'(p') = \mathtt{Sm}(p')$ and $\mathtt{T}'(p') = \mathtt{T}(p')$. This operator is straightforwardly generalised to sets of templates.

Remark 10. In the above the names for processes is selected non-deterministically. To make this selection deterministic we will assume an ordering on PNames and always extract the smallest element from Active. Also, we will assume an ordering among templates so that when spawning a collection of templates they are spawned in a deterministic order.

For simplicity we let each template initially be instantiated by one process thus the initial state is defined as $(\emptyset, \_, \_) \oplus \{\mathcal{T}_1 \ldots, \mathcal{T}_n\}$. This can easily be modified to only spawn a subset of the processes. The transition relation of a template collection is defined in Figure D.6. The semantics states that the entire system can delay if all processes can participate in the delay. Regarding actions, the system can perform an action $\mathtt{a}!$ if there exists some process, of the template $\mathcal{T}$ owning the action, that can perform it and all other components change state in accordance with the input. The processes of $\mathcal{T}$ not performing the action simply ignore it.

Example 20. In Figure D.7 we show a graphical representation of the location changes of the clients. In the plot we see that all the clients are spawned in location `Starting` and after some delay move into `Waiting`. Then after some time has passed they are accepted and enter the `Working` location. From there they disappear when entering the `End` location

$$\textsc{Delay}\quad (\mathsf{Active}, \mathtt{T}, \mathtt{Sm}) \xrightarrow{d} (\mathsf{Active}, \mathtt{T}, \mathtt{Sm}')$$
$$\text{if } d \in \mathbb{R}_{\geq 0} \text{ and for all } p \in \mathsf{Active}, \mathtt{Sm}(p) \xrightarrow{d} \mathtt{Sm}'(p);$$

$$\textsc{Action}\quad (\mathsf{Active}, \mathtt{T}, \mathtt{Sm}) \xrightarrow{a!} (\mathsf{Active}, \mathtt{T}, \mathtt{Sm}') \oplus P$$
$$\text{if } a \in \Sigma_j, \text{and there exists } p \in \mathsf{Active} \text{ such that:}$$
$$\mathtt{T}(p) = \mathcal{T}_j,\ \mathtt{Sm}(p) \xrightarrow{a!} \mathtt{Sm}'(p)\ ,\ P = \mathtt{spawn}^j(\mathtt{Sm}(p), a)),$$
$$\text{and for all } q \in (\mathsf{Active} \setminus \{p\}),\ \text{if } \mathtt{T}(q) = \mathcal{T}_j$$
$$\text{then } \mathtt{Sm}(q) = \mathtt{Sm}'(q), \text{otherwise } \mathtt{Sm}(q) \xrightarrow{a?} \mathtt{Sm}'(q).$$

Figure D.6: Transition relation of a Template collection

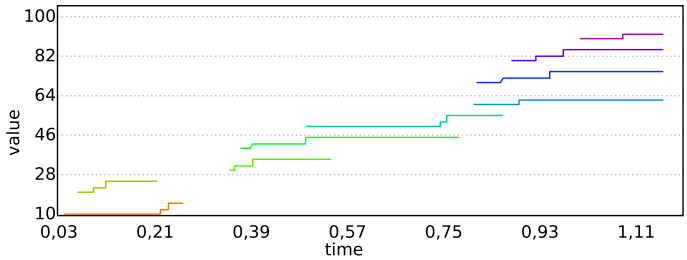


Figure D.7: Gantt chart of the clients in the client-server example. Each line represent a single client. What location a client is in can be read by first calculating $\bar{y} = y \mod 10$ where after the location is given as follows: $\bar{y} = 0 \implies \mathtt{Starting}$, $\bar{y} = 2 \implies \mathtt{Waiting}$, $\bar{y} = 5 \implies \mathtt{Working}$ and $\bar{y} = 8 \implies \mathtt{Timeout}$.

A timed run of a template collection $\mathcal{J}$ is an infinite sequence $\pi = \mathbf{s}_0 d_0 \mathbf{s}_1 d_1 \ldots$ such that $\mathbf{s}_0$ is the initial state of $\mathcal{J}$, and for all $i \in \mathbb{N}$: $d_i \in \mathbb{R}_{\geq 0}$ and $\mathbf{s}_i \xrightarrow{d_i} \xrightarrow{a_i} \mathbf{s}_{i+1}$ for some $a_i \in \Sigma$. An infinite run is called time-diverging, if for any constant, $c \in \mathbb{R}_{\geq 0}$, there exists a j such that $\sum_{i=0}^{j}(d_i) > c$. For a template collection $\mathcal{J}$, we let $\Omega(\mathcal{J})$ be the set of all diverging runs.

Consider each template $\mathcal{T}$ has a finite set of atomic propositions $\mathtt{AP}_{\mathcal{T}}$ that can be true in states of that template. Now, let s be a state of $\mathcal{T}$ then $\mathtt{Pm}^{\mathtt{AP}}(s)$ gives the finite subset of $\mathtt{AP}$ that is true in s.

For a template collection $\mathcal{J} = \mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_n$, we may consider having global propositions $\mathtt{AP}_{\mathcal{J}}$ as well as the propositions for each template type $\mathtt{AP}_{\mathcal{T}_1}, \ldots, \mathtt{AP}_{\mathcal{T}_n}$. With these we can then define the propositions that are true in a state $\mathbf{s}$ of $\mathcal{J}$ as

$$\mathtt{Pm}_{\mathcal{J}}(\mathbf{s}) = \{(a, \mathcal{T}_i, p) \mid a \in \mathsf{Active} \wedge \mathtt{T}(a) = \mathcal{T}_i \wedge p \in \mathtt{Pm}^{\mathtt{AP}_{\mathcal{T}_i}}(\mathtt{Sm}(a)\} \cup \mathtt{Pm}^{\mathtt{AP}_{\mathcal{J}}}(\mathbf{s}), \tag{D.1}$$

where $\mathtt{Pm}^{\mathtt{AP}_\mathcal{J}}$ gives the finite subset of $\mathtt{AP}_\mathcal{J}$ true in $\mathbf{s}$.

With the mapping of states to propositions the set of propositional runs of template collection $\mathcal{J}$ is:

$$\Omega^{\mathtt{AP}}(\mathcal{J}) = \{\mathtt{Pm}_\mathcal{J}(\mathbf{s}_0) d_0 \mathtt{Pm}_\mathcal{J}(\mathbf{s}_1) \cdots \mid \mathbf{s}_0 d_o \mathbf{s}_1 \cdots \in \Omega(\mathcal{J})\}$$

3.1 Stochastic Semantics

Following David et al. [45], our stochastic semantics is based on output races among components i.e. each component chooses a delay and the one with the smallest delay wins the race. Afterwards, the winning component chooses an action to perform and another race commences.

On the component-level we associate to each state of a TIOTS $(S, s_0, \to)$ a delay density function - for a state s we write μ_s to obtain this density. In addition we assign an output probability function γ_s to all states mapping output actions to probabilities. Naturally we require that $\gamma_s(a) = 0$ if and only if $s \overset{a}{\not\to}$.

Now, let $\mathcal{J} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)$ be a template collection. We want to define a measure on a set of propositional runs of $\mathcal{J}$. The set is defined through a cylinder construction: let $\omega = \mathtt{P}_0 I_0 \ldots \mathtt{P}_m$, where $\mathtt{P}_i \subset_{fin} \mathtt{AP}$,

$$\mathtt{AP} = \{(a, \mathcal{T}, p) \mid a \in \mathsf{PNames}, \mathcal{T} \in \{\mathcal{T}_1, \ldots, \mathcal{T}_n\} \text{ and } p \in \mathtt{AP}_\mathcal{T}\} \cup \mathtt{AP}_\mathcal{J}$$

and all I_i are non-empty intervals. Notice that $\mathtt{AP}$ is an infinite set, yet all $\mathtt{P}_i$ must be finite subsets.

Now, we can calculate the probability of observing a run in the cylinder $\mathcal{C}_\mathcal{J}(\omega) = \{\mathtt{P}_0 d_0 \mathtt{P}_1 d_1 \ldots \mathtt{P}_m \cdots \in \Omega^{\mathtt{AP}}(\mathcal{J}) \mid d_i \in I_i\}$ from state $\mathbf{s} = (\mathsf{Active}, \mathtt{T}, \mathtt{Sm})$ as

$$\begin{aligned}\mathbb{P}_\mathcal{J}(\mathbf{s}, \mathcal{C}_\mathcal{J}(\omega)) = \quad & (\mathtt{Pm}_\mathcal{J}(\mathbf{s}) \overset{?}{=} \mathtt{P}_0) \cdot \sum_{k \in \mathsf{Active}} \left(\int_{I_0} \mu_{\mathtt{Sm}(k)}(t) \cdot \right. \\ & \left(\prod_{k' \in \mathsf{Active} \setminus \{k\}} \int_{\tau > t} \mu_{\mathtt{Sm}(k')}(\tau)\, \mathrm{d}\tau \right) \cdot \\ & \left. \sum_{a \in \Sigma_o} \left(\gamma_{[\mathtt{Sm}(k)]^t}(a) \cdot \mathbb{P}_\mathcal{J} \left(\left[[\mathbf{s}]^t \right]^{a/k}, \mathcal{C}_\mathcal{J}(\mathtt{P}_1 I_1 \ldots \mathtt{P}_m) \right) \right) \mathrm{d}t \right)\end{aligned}$$

with base case $\mathbb{P}_\mathcal{J}(\mathbf{s}, \mathtt{P}) = P_\mathcal{J}(\mathbf{s}) \overset{?}{=} \mathtt{P}$, $(\mathtt{P} \overset{?}{=} \mathtt{P}')$ is 1 if $\mathtt{P} = \mathtt{P}'$ and 0 otherwise. In this expression we use $[\mathbf{s}]^{a/k}$ to obtain the uniquely defined state that is reached if the process with name k performs action a.

The probability defined above requires some explanation: first it is checked if the propositions true in the first state matches those of the cylinder, then on the outermost level we sum over all active processes. After some delay t in

I_0 , the winning process chooses to perform some action. Independently, the other processes choose a delay, τ, greater than t - captured by the inner integral. Having delayed t, all the possible actions that the winning component can perform and their probabilities are taken into account. Finally, the probability of seeing the remaining part of the cylinder is multiplied.

For the remainder of this paper we let $\mathcal{C}_{\mathcal{J}}(\mathcal{J})$ be all cylinders.

Remark 11. Allowing spawning of templates one might worry if the system will *explode* in the sense that discrete actions may occur with shorter and shorter time between them due to growing number of components. Essentially, one might worry if the system would exhibit a zeno behaviour. Luckily it follows from Reuters criteria for birth-and-death processes[103] that if we only have exponential distributions (spanning a finite range of rates) or uniform distributions (spanning a finite range of intervals), the system will not explode. We rely on this fact as our statistical model checking algorithm requires that runs are time-diverging.

4 Quantified Dynamic Metric Temporal Logic

In this section we present the syntax and semantics of quantified dynamic metric temporal logic (QDTML) that is highly based on MTL. The logic is defined over a template collection $\mathcal{J} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)$.

For any TIOTS $\mathcal{K} = (S, s_0, \rightarrow)$ we assume there exists a set of numeric expressions $\textsc{Expr}_{\mathcal{K}}$ that we can evaluate in any of its states. Similarly, we assume there exists a set of boolean expressions $\textsc{Bool}_{\mathcal{K}}$. In both cases we evaluate the expression e in state $s \in S$ by $[\![e]\!](\mathcal{K}, s)$. If $e \in \textsc{Bool}_{\mathcal{K}}$ the result is contained in the set $\{\mathtt{tt}, \mathtt{ff}\}$ otherwise it returns a real-valued number. Now let $\mathbf{s} = (\mathsf{Active}, \mathtt{T}, \mathtt{Sm})$ be a state of $\mathcal{J}$, $p \in \mathsf{Active}$, $\mathtt{T}(p) = \mathcal{T}$ and let $\mathcal{T} = (\mathcal{K}, \mathtt{spawn})$. Then we denote the evaluation of $e \in \textsc{Expr}_{\mathcal{K}}$ in the context of p in $\mathbf{s}$ by $[\![e]\!](p, \mathbf{s}) = [\![e]\!](\mathcal{K}, f(p))$. Similar notation is used for the boolean expressions.

Assume we have a finite set of names PVar each assigned a template type, that will act as placeholders for processes in formulas. For $P \in \mathsf{PVar}$ we denote its type $\mathcal{T}$ by $(P : \mathcal{T})$. Given this set of variables, the set of numeric expressions over a template collection $\mathcal{J}$ is generated by the syntax

$$\mathcal{E} ::= c \mid \mathcal{E}_1 \,\mathsf{op}\, \mathcal{E}_2 \mid \mathsf{sum}(\mathcal{T})(e) \mid P.e_2$$

where $c \in \mathbb{R}$, $\mathsf{op} \in \{+, -, \cdot, /\}$, $e \in \textsc{Expr}_{\mathcal{T}}$ and if $(P : \mathcal{T}_1)$ then $e_2 \in \textsc{Expr}_{\mathcal{T}_1}$. To evaluate these expressions we need to bind the process names in PVar to actual process names in PNames. We do this in terms of a mapping $M : \mathsf{PVar} \rightarrow \mathsf{PNames} \cup \{\star\}$, where $\star \notin \mathsf{PNames}$. The symbol $\star$ is here used to denote that a name has not been bound to a process. We then give the semantical meaning of an expression $\mathcal{E}$ in a given state $\mathbf{s} = (\mathsf{Active}, \mathtt{T}, \mathtt{Sm})$ and with mapping M, denoted $[\![\cdot\mathcal{E}\cdot]\!](\mathbf{s}, M)$, recursively as:

- $[\![\cdot c\cdot]\!](\mathbf{s}, M) = c$,
- $[\![\cdot \mathcal{E}_1 \,\mathsf{op}\, \mathcal{E}_2 \cdot]\!](\mathbf{s}, M) = [\![\cdot \mathcal{E}_1 \cdot]\!](\mathbf{s}, M) \,\mathsf{op}\, [\![\cdot \mathcal{E}_2 \cdot]\!](\mathbf{s}, M)$,
- $[\![\cdot \mathsf{sum}(\mathcal{T})(e) \cdot]\!](\mathbf{s}, M) = \sum_{p \in \{\mathbf{s}^{\mathcal{T}}\}} ([\![e]\!](p, \mathbf{s}))$ and
- $[\![\cdot P.e \cdot]\!](\mathbf{s}, M) = [\![e]\!](M(P), \mathbf{s})$

where if $\mathbf{s} = (\mathsf{Active}, \mathtt{T}, \mathtt{Sm})$ then $\{\mathbf{s}^{\mathcal{T}}\} = \{p \in \mathsf{Active} \mid \mathtt{T}(p) = \mathcal{T}\}$. Naturally the latter is only defined if $M(P) \neq \star$.

The set of QDTML formulas for a template collection $\mathcal{J} = (\mathcal{T}_1, \ldots, \mathcal{T}_n)$ is generated by the syntax

$$\varphi := \mathtt{tt} \mid P.\tilde{b} \mid \mathcal{E}_1 \bowtie \mathcal{E}_2 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \,\mathsf{U}_{[a,b]}\, \varphi_2 \mid \mathsf{forall}(P : \mathcal{T})\varphi$$

where $\bowtie \in \{<, \leq, \geq, >\}$, $P \in \mathsf{PVar}$, $(P : \mathcal{T})$, $a, b \in \mathbb{R}_{\geq 0}$, where $a \leq b$ and $\tilde{b} \in \textsc{Bool}_{\mathcal{T}}$.

As it is custom in the family of MTL logics we use $\Diamond_{[a;b]}\varphi$ as a syntactical short hand for $\mathtt{tt}U_{[a;b]}\varphi$, $\Box_{[a;b]}\varphi$ for $\neg\Diamond_{[a;b]}\neg\varphi$. We derive the classic boolean operators $\vee$ and $\Longrightarrow$ in the usual way and let $\mathsf{exists}(\mathsf{P} : \mathcal{T})\varphi = \neg\mathsf{forall}(\mathsf{P} : \mathcal{T})\neg\varphi$. We call an occurrence of a subformula where every $P.b$ or $P.e$ is surrounded by a binding occurrence of the form $\mathsf{forall}(P : \mathcal{T})$ a sentence. In the semantics, the actual binding is accomplished by updating a mapping whenever encountering an occurrence of $\mathsf{forall}(P : \mathcal{T})$: if M is a mapping then

$$M[P \mapsto p](x) = \begin{cases} p & \text{if } x = P \\ M(x) & \text{otherwise} \end{cases}$$

Let $\pi = \mathbf{s}_0 d_0 \mathbf{s}_1 d_1 \ldots$ be a run of $\mathcal{J}$, where $\mathbf{s}_i = (\mathsf{Active}_i, \mathtt{T}_i, \mathtt{Sm}_i)$ for all $i > 0$, and let φ be a QDTML formula. Then we define satisfaction of φ with respect to a mapping M recursively as,

- $\pi \models^M \mathtt{tt}$
- $\pi \models^M P.\tilde{b}$ if $M(P) \neq \star$ and $[\![\cdot \tilde{b} \cdot]\!](M(P), \mathbf{s}_0) = \mathtt{tt}$
- $\pi \models^M \mathcal{E}_1 \bowtie \mathcal{E}_2$ if $[\![\cdot \mathcal{E}_1 \cdot]\!](\mathbf{s}_0, M) \bowtie [\![\cdot \mathcal{E}_2 \cdot]\!](\mathbf{s}_0, M)$
- $\pi \models^M \neg\varphi$ if $\pi \not\models^M \varphi$
- $\pi \models^M \varphi_1 \wedge \varphi_2$ if $\pi \models^M \varphi_1$ and $\pi \models^M \varphi_2$
- $\pi \models^M \mathsf{X}\varphi$ if $\pi^{1:} \models^M \varphi$
- $\pi \models^M \varphi_1 \,\mathsf{U}_{[a,b]}\, \varphi_2$ if there exists j such that $\pi^{j:} \models^M \varphi_2$, $\sum_{i=0}^{j-1} d_i \in [a, b]$, and for all $k < j$, $\pi^{k:} \models^M \varphi_1$.

- $\pi \models^M \mathsf{forall}(P : \mathcal{T})\varphi$ if for each $p \in \mathsf{Active}_0$, if $\mathtt{T}_0(p) = \mathcal{T}$ then $\pi \models^{M[P \mapsto p]} \varphi$

Example 21. Consider our running example of a client-server model. Possible QDTML formulas over this system are:

$$\Box_{0;5}(\mathsf{forall}(c : \mathtt{Client})(c.\mathtt{Waiting} \implies \Diamond_{[0;10]}(c.\mathtt{Working})))$$
$$\Box_{0;5}(\mathsf{forall}(c : \mathtt{Client})(c.\mathtt{Waiting} \implies \Diamond_{[0;10]}($$
$$c.\mathtt{Working} \wedge \mathsf{exists}(\mathtt{s} : \mathtt{ServerChild})$$
$$(s.\mathtt{Working} \wedge s.id == c.id))))$$

The first formula asserts that if a client within the first five time units is awaiting a connection, then it will come to the working location. The second formula asserts, in addition to this, that a `ServerChild` should also be in the `Working` location and have the same id i.e it asserts that a `ServerChild` is communicating with the client.

Definition 38. Let φ be a QDTML sentence and let M_0 be a function where for all $P \in \mathsf{PVar}$ $M_0(P) = \star$. Then we define that $\pi \vDash \varphi$ iff $\pi \vDash^{M_0} \varphi$.

Theorem 7. For all QDTML formulas φ, all template collections $\mathcal{J}$ and mappings $M : \mathsf{PVar} \to \mathsf{PNames}$, the set $\{\pi \in \Omega(\mathcal{J}) \mid \pi \vDash^M \varphi\}$ is measurable.

Proof. (sketch)
For this sketch we focus on the subset of QDTML where the construction $\mathcal{E}_1 \sim \mathcal{E}_2$ is omitted. First, we define the propositions per template $\mathcal{T}$ that we need to know the value of at each state. Let φ be a QDTML formula and $\mathcal{T} = (\mathcal{K}, \mathtt{spawn})$ a template then $\mathtt{AP}_{\mathcal{T}} \subseteq \textsc{Bool}_{\mathcal{K}}$ is a finite set of properties that are relevant for φ i.e. $\mathtt{AP}_{\mathcal{T}} = \{\tilde{b} | P.\tilde{b}$ is a sub-expression of φ and $(P : \mathcal{T})\}$. Let $\mathbf{s}$ be a state of the template collection $\mathcal{J}$ then the global propositions $\mathtt{AP}_{\mathcal{J}}$ we are interested in are the active processes and their type thus the global propositions $\mathtt{AP}_{\mathcal{J}} = \{(a, \mathcal{T}) | a \in \mathsf{Active} \wedge \ \mathcal{T}$ is a template$\}$. We let

$$\mathtt{Pm}_{\mathcal{J}}(\mathbf{s}) = \left\{ \ (a, \mathcal{T}, \tilde{b}) \ \middle| \ \begin{array}{l} a \in \mathsf{Active} \wedge \exists \tilde{b} \in \mathtt{AP}_{\mathcal{T}} \wedge \mathtt{T}(a) = \mathcal{T} \wedge \\ [\![\cdot \tilde{b} \cdot]\!](a, \mathbf{s}) = \mathsf{tt} \end{array} \right\} \bigcup$$
$$\{(a, \mathcal{T}) | a \in \mathsf{Active} \wedge \mathtt{T}(a) = \mathcal{T}\}$$

This is merely a concrete instance of the abstract proposition mapping given in Eq. (D.1). Notice that the propositional run induced by the above proposition mapping contains enough information to conclude if a QDTML formula is satisfied on that run [3] thus we can easily define a satisfaction relation between a propositional run and a QDTML formula and easily show that

[3] recalling that we omitted the $\mathcal{E}_1 \sim \mathcal{E}_2$

$$\mathbf{s}_0 d_0 \mathbf{s}_1 \ldots \vDash^M \varphi \Leftrightarrow \mathtt{Pm}_{\mathcal{J}}(\mathbf{s}_0) d_0 \mathtt{Pm}_{\mathcal{J}}(\mathbf{s}_1) \ldots \vDash^M \varphi$$

What remains is to show that the set

$$\{\pi | \pi \in \Omega^{\mathtt{AP}}(\mathcal{J}) \wedge \pi \vDash^M \varphi\}$$

is indeed measurable i.e. is representable by a set of cylinders.

We do so by structural induction in φ. For this sketch we only show the construction for one simple formula. Let $\varphi = P_1.\tilde{b}\,\mathsf{U}_{[a,b]}\,P_2.\tilde{b}$ and Consider the set of cylinders:

$$\begin{aligned}\{&\mathtt{P}_0 I_0 \mathtt{P}_1 I_1 \ldots \mathtt{P}_n \in \mathcal{C}_{\mathcal{J}}(\mathcal{J}) \mid \exists j \text{ s.t. } \exists (M(P_2), \mathcal{T}_2, \tilde{b}) \in \mathtt{P}_j \\ &\quad \text{with } (P_2 : \mathcal{T}_2) \text{ and } \forall i < j, \exists (M(P_1), \mathcal{T}_1, \tilde{b}) \in \mathtt{P}_i \text{ with } (P_1 : \mathcal{T}_1) \text{ and} \\ &\quad \left(\sum_{k=1..j-1} I_k \right) \in [a, b]\}\end{aligned}$$

Quite clearly this is representable by a union of cylinders and equally clearly any run contained in any of the cylinders satisfy $\varphi = P.\tilde{b}\mathsf{U}_{[a,b]}P_2.\tilde{b}$.

□

5 Statistical Model Checking of QDMTL

Statistical model checking [123] is a simulation based software verification technique. Underlying the technique is that the model has a *Stochastic* semantics and that we efficiently can obtain runs from its associated probability distribution. In addition we need a logic for which we can settle if a formula is satisfied by a run. Generating a run of a model and validating if a formula is satisfied gives rise to a *Bernoulli* variable X that obtains the value 1 if the formula was satisfied and 0 if it was not satisfied. The probability that $X = 1$ is the probability that a random run satisfies the formula. Let this probability be θ. Now let $X_1, X_2 \ldots X_n$ be n such Bernoulli variables and let Y be a random variable obtaining the value

$$Y = \sum_{i=1}^{n} X_i,$$

i.e. Y counts the number of runs that satisfied the formula. Y is distributed according to a binomial distribution with succes-parameter θ. If we want to answer the *qualitative* question "is θ greater than a threshold" we may employ a *hypothesis testing* approach with a controllable level of significance[118]. In case we want to answer the *quantitative* question "what is the probability θ" we can employ an estimation approach and obtain a confidence interval. One such method is using the Chernoff-Bound as described in [45]

Statistical Model Checker

A naive statistical model checker thus consists of (1) a component that generate runs of a model, (2) a component that can settle if a formula is satisfied for a given run and (3) an algorithm from statistics that either estimates the probability θ or tests if it exceeds a threshold value.

The decoupling of the generation of the run and the validation of a run has the positive effect, that implementing a statistical model checker is easy and the individual components may easily be exchanged for others. The decoupling is, however, inefficient as time may be wasted generating a long run violatinh the property after one step thus we wish to perform the validation of a run in parallel with the run generation. Previously [34], we developed such a monitoring scheme for MTL that was based on rewriting formulas: Given a run

$$\pi = (\mathbf{s}_0, d_0)(\mathbf{s}_1, d_1)\dots$$

and a MTL formula φ_0 the monitor rewrites φ_0 into φ_1 using $\mathbf{s}_1$ and d_0 as input (denoted $\varphi_0 \xrightarrow{\mathbf{s}_1, d_0} \varphi_1$) in such a way that $\pi^{1:} \vDash \varphi_1$ if and only if $\pi \vDash \varphi_0$. Continuing to rewrite the formulas eventually transforms a formula into $\mathtt{tt}$ signalling satisfaction, or ff signalling violation of the property. We now provide some of the rewrite rules needed for QDMTL. The remaining rules are similar to those presented in [34].

Since we have variables in the formulas, we need to take a mapping into account i.e. we rewrite tuples of the form (φ, M) where φ is a QMDTL formula and M is a mapping from PVar to PNames.

$$\frac{}{(\mathtt{tt}, M) \xrightarrow{\mathbf{s},d} (\mathtt{tt}, M)} \ (Atom)$$

$$\frac{[\![\cdot\mathcal{E}_1\cdot]\!](\mathbf{s}, M) = r_1 \qquad [\![\cdot\mathcal{E}_2\cdot]\!](\mathbf{s}, M) = r_2 \qquad r_1 \bowtie r_2}{(\mathcal{E}_1 \bowtie \mathcal{E}_2, M) \xrightarrow{\mathbf{s},d} (\mathtt{tt}, \mathtt{M})} \ (Eval_1)$$

$$\frac{[\![\cdot\mathcal{E}_1\cdot]\!](\mathbf{s}, M) = r_1 \qquad [\![\cdot\mathcal{E}_2\cdot]\!](\mathbf{s}, M) = r_2 \qquad r_1 \not\bowtie r_2}{(\mathcal{E}_1 \bowtie \mathcal{E}_2, M) \xrightarrow{\mathbf{s},d} (\mathtt{ff}, \mathtt{M})} \ (Eval_2)$$

Above we show the most basic rewrite rules of our monitoring technique. The first rule states that if the formula is $\mathtt{tt}$, then this will not change due to a rewrite[4]. The two latter rules expresses that to rewrite a comparison between expressions, the two expressions should first be evaluated and afterwards compared. If the comparison is true then the formula is rewritten into $\mathtt{tt}$, otherwise it becomes $\mathtt{ff}$.

Monitoring a formula $\psi = \mathsf{forall}(P : \mathcal{T})\varphi$ requires monitoring φ for each process of $\mathcal{T}$ i.e. we start a rewriting sequence per process - each of these rewrite

[4] An equivalent rule applies for ff

sequences should have their own mapping. To denote that multiple formulas should be rewritten in parallel with each other we use the syntactical construct $\bigwedge^{+}[(\varphi_1, M_1), (\varphi_2, M_2) \ldots, (\varphi_n, M_n)]$.

The formula ψ is satisfied along a run if and only if φ was satisfied for all processes when we encountered ψ. Consequently, we rewrite $\bigwedge^{+}[(\varphi_1, M_1), (\varphi_2, M_2) \ldots, (\varphi_n, M_n)]$ into $\mathtt{tt}$ if all of the formulas are rewritten into $\mathtt{tt}$ at some point (see $ConjList_2$ below). Similarly, ψ is not satisfied if one of the processes did not satisfy φ thus if a formula is rewritten into $\mathtt{ff}$ - captured by the rule $ConjList_1$. Finally, if none of the above rules apply we simply rewrite the individual formulas.

$$\frac{\exists i \in \{1, \ldots, n\} \qquad (\varphi_i, M_i) \xrightarrow{\mathbf{s},d} (\mathtt{ff}, M)}{\bigwedge^{+}[(\varphi_1, M_1), \ldots, (\varphi_n, M_n) \xrightarrow{\mathbf{s},d} (\mathtt{ff}, M_i)]} \; (ConjList_1)$$

$$\frac{\forall i \in \{1, \ldots, n\} \qquad (\varphi_i, M_i) \xrightarrow{\mathbf{s},d} \mathtt{tt}}{\bigwedge^{+}[(\varphi_1, M_n), \ldots, (\varphi_n, M_n) \xrightarrow{\mathbf{s},d} (\mathtt{tt}, M_1)]} \; (ConjList_2)$$

$$\frac{\left[(\varphi_i, M_i) \xrightarrow{\mathbf{s},d} (\varphi_i', M_i')\right]_{i=1,n}}{\bigwedge^{+}[(\varphi_1, M_n), \ldots, (\varphi_n, M_n)] \xrightarrow{\mathbf{s},d} \bigwedge^{+}[(\varphi_1', M_n'), \ldots, (\varphi_n', M_n')]} \; (ConjList_3)$$

The initiating step of transforming ψ into a $\bigwedge^{+}$ construct consist of three rules(below): the first rule corresponds to immediately discovering that one of the processes did not satisfy the formula φ thus ψ is not satisfied. The second one correspond to all processes immediately satisfy φ and consequently that ψ is satisfied.
The final rule is instantiating the parallel rewrite process by first rewriting φ for all processes and then construct the conjunction between these.

$$\frac{\exists p \in \{\mathbf{s}^{\mathcal{T}}\} \qquad (\varphi, M[P \mapsto p]) \xrightarrow{\mathbf{s},d} \mathsf{ff}}{(\mathsf{forall}(P : \mathcal{T})\varphi, M) \xrightarrow{\mathbf{s},d} \mathtt{ff}} \; (Binding_1)$$

$$\frac{\forall p \in \{\mathbf{s}^{\mathcal{T}}\} \qquad (\varphi, M[P \mapsto p]) \xrightarrow{\mathbf{s},d} \mathtt{tt}}{(\mathsf{forall}(P : \mathcal{T})\varphi, M) \xrightarrow{\mathbf{s},d} \mathtt{tt}} \; (Binding_1)$$

$$\frac{\left[(\varphi, M[P \mapsto p]) \xrightarrow{\mathbf{s},d} ((\varphi_p, M_p)\right]_{p \in \mathbf{s}^{\mathcal{T}}}}{(\mathsf{forall}(P : \mathcal{T})\varphi, M) \xrightarrow{\mathbf{s},d} \bigwedge^{+}_{p \in \mathbf{s}^{\mathcal{K}}} [(\varphi_p, M_p)]} \; (Binding_3)$$

Theorem 8. Let $\pi = \mathbf{s}_0 d_0 \mathbf{s}_1 d_1 \mathbf{s}_2 d_2 \ldots$ be an infinite time-diverging run and let φ be a QMDTL sentence. In addition let M be a function where for all $P \in \mathsf{PVar}\ M(P) = \star$. Then $\pi \vDash \varphi$ if and only if there exists a rewrite sequence

$$(\varphi, M) \xrightarrow{\mathbf{s}_0, d_0} (\varphi', M') \xrightarrow{\mathbf{s}_1, d_1} \ldots (\mathtt{tt}, M'')$$

and $\pi \nvDash \varphi$ if and only if there exists a rewrite sequence

$$(\varphi, M) \xrightarrow{\mathbf{s}_0, d_0} (\varphi', M') \xrightarrow{\mathbf{s}_1, d_1} \ldots (\mathtt{ff}, M'')$$

6 Experiments

6.1 Robot

We consider the imaginary example of robots moving randomly on a 2-dimensional grid in search of a specific location. In the following we describe a parameterised model so that we can make a series of experiments. Let the grid have size $x \times y$ and the goal be location $(x-1, y-1)$. Initially no robots are present on the grid but are spawned by an extra component. The robots are spawned in location $(i, 0)$ where i is randomly selected in the range $[0, x-1]$ for each robot. The robots each have a variable x and y that tells where they are on the grid and move with a rate of 2. The robots can only move up,down,left or right and only inside the grid thus they cannot move from one boundary to the other in one step.

In this random setup we may wonder how likely it is for two robots to be in the same location at once within some time limit τ. In Uppaal SMC we can find the probability of no robots being in the same field at once by checking the query

```
Pr ([][0; τ] forall (t:Robot)
             (forall (t2:Robot)
             (t.x == t2.x && t.y ==t2.y
              imply  t==t2))).
```

We call this formula ϕ_1. Naturally the probability we are interested in is $1 - \theta$ where θ is the probability of satisfying ϕ_1.

In Table D.1 we show the probability estimated for ϕ_1 by Uppaal SMC for various parameters to the model.

Setting aside that robots may actually be on the same field, we might be interested in estimating the probability that one robot reaches the goal, within τ time units, and it does so with a margin of δ to the second robot reaching

x	y	#Robots	Probability	Time (s)
2	2	1	$[0.95, 1.00]$	0.15
4	4	2	$[0.00, 0.07]$	1.01
8	8	4	$[0.00, 0.08]$	1.06
64	64	4	$[0.41, 0.51]$	2.55
64	64	8	$[0.06, 0.16]$	2.93

Table D.1: Verification results for ϕ_1 for various grid sizes and number of robots with $\tau = 100$. In the probability column is a 95% confidence interval satisfying ϕ_1.

the goal. This is straightforwardly expressed in UPPAAL SMC by the query

```
Pr (<>[0;τ]exists{t:Robot} (t.x==xgoal
          && t.y==ygoal ∧
          ([][0;δ] forall{t2:Robot}
          ((t2.x==xgoal && t2.y==ygoal)
           imply t==t2))))
```

where (`xgoal`, `ygoal`) is the goal location. We call this formula for ϕ_2 Again we show the verification result for various parameters to the model in Table D.2.

x	y	#Robots	Probability	Time (s)
2	2	1	$[0.95, 1.00]$	0.13
4	4	2	$[0.58, 0.68]$	1.55
8	8	4	$[0.47, 0.57]$	6.01
8	8	8	$[0.42, 0.52]$	22.27
16	16	8	$[0.00, 0.05]$	10.27

Table D.2: Verification results for ϕ_2 for various grid sizes and number of robots with $\tau = 100$ and $\delta = 20$.

6.2 Client-Server

We now turn our focus towards applying QDTML to the client-server example. For a start, we consider the property that all clients get a connection within 10 time units after their initial connection request. We verify this within the first 10 time units of a run. In UPPAAL SMC the property is expressed as

```
Pr ([][0;10] forall { c :  Client}
          (!(c.Waiting && c.retries==0) ∨
          (<>[0;10]c.Working )))
```

`Ports`	Probability	Time	Largest	σ
10	$[0.17, 0.27]$	17m56.750s	2054	2079.53
13	$[0.84, 0.94]$	4m55.545s	491	854.49
15	$[0.94, 1.00]$	2m11.51s	198	440.1

Table D.3: Probability that client do not get a connection in their first try. The `Ports` column is the number of simultaneous connections the server can handle. The Probability column is a 95% confidence interval, time is the time used for the verification in seconds. Largest is the average largest intermediate formula encountered during a verification and σ is its standard deviation

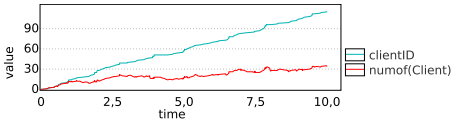


Figure D.8: Development of the number of Clients. The `clientID` curve is the total number of spawned clients - the `numOf(Client)` is number of active clients.

This probability depends on the number of simultaneous connections that the server can handle - in our model this corresponds to the number of ports. In Table D.3 we show the 95% confidence intervals obtained by verifying the property for three different number of ports. We also present the time used for the verification. In all of the three cases, the verification used 738 runs. During the generation of the runs we measured the size of the intermediate formulas, in terms of the number of terminals and recorded the largest formula encountered. In the *Largest* column we give the average of these and in the σ column we show the standard deviation. We observe that the probability is higher when we provide the server with more ports. At first the verification time seems high, but recall that each discrete step in the model results in an expansion of the forall ,and as a large large number of clients are spawned, as seen in Figure D.8, this results in large formulas. The monitoring technique is recursive in the formulas parse tree thus large formulas result in large verification time. We conjecture that the verification is smaller when the probability is high simply because the intermediate formulas encountered during verification is smaller. This is supported by the results in Table D.3.

If we take a closer look at the model, we notice that it is possible for the `ServerChild` to get stuck. If it chooses a long delay > 5 before signalling a connection to the client, the `Client` would have timed out and might not be ready for receiving the synchronisation from `ServerChild`. In the model there is no time out on the server side thus the `ServerChild` would wait for a disconnect synchronisation forever. This is possible in the model but verifying the property

```
Pr (<>[1;10] exists { s :  ServerChild}
                ([][0;20] s.Working))
```

in UPPAAL SMC, with 10 ports available, shows that the probability of having a `ServerChild` in the `s.Working` location for 20 time units is in the interval $[0.00, 0.01]$ with a confidence of 95% thus it seems like an improbable event.

7 Conclusion

In this paper we have presented the logic QDMTL. The logic has been developed with the specific aim of reasoning on the behaviour of the dynamic systems i.e. systems consisting of a dynamically evolving set of processes. The logic is complemented by an on-the-fly monitoring technique that given a time diverging run of the system is guaranteed to terminate and provide a correct result. We have applied our new logic to two examples, one being several robots moving autonomously on a grid and the other being our running example of a client-server architecture.

In the future we will continue our work towards providing a tool set for reasoning on dynamic systems. Especially we will extend the modelling formalism towards dynamic generation of communication channels and for allowing processes to communicate these to each other.

Statistical Model Checking for Biological Systems

Abstract *Statistical Model Checking (SMC) is a highly scalable simulation-based verification approach for testing and estimating the probability that a stochastic system satisfy a given linear temporal property. The technique has been applied to (discrete and continuous time) Markov chains, stochastic timed automata and most recently hybrid systems using the tool* Uppaal SMC. *In this paper we enable the application of SMC to complex biological systems, by combining* Uppaal SMC *with ANIMO, a plugin of the tool Cytoscape used by biologists, as well as with SimBiology®, a plugin of* Matlab *to simulate reactions. ANIMO and SimBiology® are two domain specific tools that have their own user interfaces and formalisms specifically tailored towards the biological domain. However – though providing means for simulation – both tools lack the powerful analytic capabilities offered by SMC, which in previous work have proved very useful for identifying interesting properties of biological systems. Our aim is to offer the best of the two worlds: optimal domain specific interfaces and formalisms suited to biology combined with powerful SMC analysis techniques for stochastic and hybrid systems. This goal is obtained by developing translators from the XGMML and SBML formats used by Cytoscape and SimBiology® to stochastic and hybrid automata, allowing* Uppaal SMC *to be used as an efficient backend analysis tool, that we demonstrate can handle real-world biological systems by pitting it against the BioModels database. We present detailed analysis on two particular case-studies involving the ANIMO and SimBiology® tools.*

1 Introduction

It is conceivable to design systems to make their analysis easier, but usually they are optimised for other constraints (efficiency, size, cost, etc.) and they evolve over time, developing highly complex and unforeseen interactions and redundancies. These phenomena are epitomised by biological systems, which have no inherent need to be understandable or analysable. The discovery that the genetic recipe of life is written with just four characters (nucleotides Adenine, Cytosine, Guanine and Thymine) that are algorithmically transcribed

and translated into the machinery of the cell (RNA and proteins) has led scientists to believe that biology also works in a computational way. The further realisation that biological molecules and interactions are discrete and stochastic then suggests that biological systems can be analysed using the same tools used to verify for instance a complex aircraft control system.

Using formal methods to investigate natural systems can thus be seen as a way to challenge and refine the process of investigating man-made systems. It is very difficult to reason about systems of this type at the level of their descriptions, however. It is much more convenient to directly analyse their observed behaviour. In the context of computational systems we refer to this approach as runtime verification, while in the case of biological systems this generally takes the form of monitoring the simulation traces of executable computational models.

There already exists several formal tools dedicated to this purpose. As an example, the ANIMO toolset [106] can be used to model biological pathways. This tool handles phenomena described via the Cytoscape library [110]. The tool chain goes via a translation from the library to timed automata, which allows exploiting UPPAAL for verifying properties of the system. Unfortunately, ANIMO is restricted so that it cannot capture the stochastic behaviours inherent to many biological phenomena. Moreover, its expressive power is restricted to timed automata while many behaviours have to be described via general Ordinary Differential Equations (ODE). Another interesting toolset is MATLAB with SimBiology®[1] frontend from Mathworks. While MATLAB is clearly powerful enough to handle complex phenomena, including non linear ones, it does not provide efficient verification techniques and powerful logics to describe eventually complex properties one may want to measure and check on the model.

In this paper, we propose a new tool-chain for the analysis of biological systems. Our approach heavily relies on statistical model checking (SMC) [87, 108, 121], a powerful formal approach that has recently been proposed as a new validation technique for large-scale, complex systems. The core idea of SMC is to conduct some simulations of the system, monitor them, and then use statistical methods (including sequential hypothesis testing or Monte Carlo simulation) to decide with some degree of confidence whether the system satisfies the property or not. By nature, SMC is a compromise between testing and classical formal method techniques. Simulation-based methods are known to be far less memory and time intensive than exhaustive ones, and are some times the only option. SMC has been implemented in a series of tools [80, 86, 120] that have defeated well-known numerical-based analysis tools on several non-academic case studies. Such tools have been applied to large-size industrial applications such as the verification of a complex aircraft control system [16], schedulability analysis of mixed criticality systems [50] or more recently for complex systems of systems within the integrated project DANSE[2] and performance evaluation of

[1] http://www.mathworks.se/products/simbiology/
[2] http://www.danse-ip.eu/home/

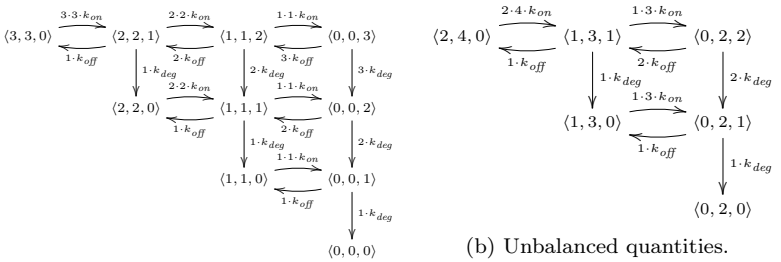
energy-aware buildings in the Sino-Danish Basic Research Center IDEA4CPS[3].

In fact, one shall see that biological phenomena can be represented by networks of stochastic hybrid automata (SHAs). Stochastic hybrid automata are timed automata whose clocks can evolve with different rates, which may depend not only on values of discrete variables but also on the value of other clocks, effectively amounting to ordinary differential equations (ODEs). In [45, 47], we showed that SHAs can be equipped with a stochastic semantic based on stochastic delays and repeated races between the components of composite model. Importantly, the stochastic semantics provide the foundation for well-defined probability measures for a range of linear temporal properties. In the present paper, we shall see that this model is general and can be used to capture a wide range of biological phenomena. More precisely, our approach can handle biochemical reactions that rely on the interaction of molecules of different species. In one approach, the models based on *elemental reactions* with *mass action kinetics* may be *simulated exactly* as a continuous time Markov chain (CTMC) having discrete states. Alternatively, in case of huge amount of molecules, approximate analysis may be preferable (or necessary) using sets of coupled ODEs that assume continuous states. The method of conversion between these two paradigms, along with explanations of the emphasised terms, is given in [63].

We then see how to connect ANIMO and MATLAB SimBiology® to UPPAAL SMC. UPPAAL SMC is a stochastic and statistical model checking extension of UPPAAL that relies on the SHA model described above. UPPAAL SMC comes together with a friendly user interface that allows a user to specify complex problems in an efficient manner as well as to get feedback in the form of probability distributions and compare probabilities to analyse performance aspects of systems. The UPPAAL SMC model checking engine has been applied to a wide range of examples ranging from networking and Nash equilibrium [33] through systems biology [47, 51], real-time scheduling [50] to energy aware systems [48]. Our connection is based on intermediary translations to XGMML (used by ANIMO) or SBML (used by SimBiology®) to the CTMCs and ODEs format of UPPAAL SMC. By connecting ANIMO and SimBiology® to UPPAAL SMC, we not only unify their expressive power, but also make the powerful simulation engine of SMC available for efficient verification of complex behaviours. It is worth observing that our transformation is general and can be used to connect UPPAAL SMC to other libraries to describe biological phenomena, including the BioNetGen framework of Faeder et al. [56].

Finally, we compare the performances of ANIMO, SimBiology®, and UPPAAL SMC on several biology examples. The structure of the paper is as follows. Section 2 presents the formalisms used in ANIMO and Simbiology, Section 3 reviews the SHA modelling formalism of UPPAAL SMC as well as application of its SMC engine, Section 4 presents our translators from ANIMO and SimBiology® (SBML format in fact) to UPPAAL SMC, and Section 5

[3] `http://www.idea4cps.dk/`



(a) Balanced quantities.

(b) Unbalanced quantities.

Figure E.1: Continuous Time Markov Chain models of reactions.

focuses on two case-studies and compares the different tools.

2 Modeling Formalisms for Biology

Here we introduce a simple example and use it to demonstrate features of ANIMO and MATLAB SimBiology®. Suppose we have two reactants A and B which produce C with a rate $k_{on} = 0.2m^{-1}s^{-1}$, the reaction can be reversed with a rate $k_{off} = 0.1s^{-1}$ and the product C can decay into some other materials with rate $k_{deg} = 1.0s^{-1}$. The system can be described by the following chemical reactions:

$$\mathsf{A} + \mathsf{B} \underset{k_{off}}{\overset{k_{on}}{\rightleftharpoons}} \mathsf{C} \xrightarrow{k_{deg}}$$

Due to results of [63], such a system of chemical reactions can be modelled as a CTMC under the assumptions that at most two molecules participate in a reaction instance and that molecules are well mixed (uniform reaction rates). Figure E.1a shows a CTMC model of our reactions where the state $\langle[\mathsf{A}], [\mathsf{B}], [\mathsf{C}]\rangle$ corresponds to the number of molecules of each reactant and transition probability is proportional to the reaction rate and available reactant molecules. For example, if we start with a mix containing [A]=3, [B]=3 and [C]=0 molecules (state $\langle 3, 3, 0\rangle$ in Figure E.1a), then the molecules of A and B may react with a probability rate of $3 \cdot 3 \cdot k_{on}$ by consuming one molecule of each A and B and producing one C, thus the resulting target state contains [A]=2, [B]=2 and [C]=1 molecules (state $\langle 2, 2, 1\rangle$). The other reactions can be modelled likewise. Eventually all of our molecules decay into other materials which do not participate in our system resulting in a dead-end state $\langle 0, 0, 0\rangle$. If the initial quantities are not so well balanced, then the final state may contain some of the initial reactants like in Figure E.1b. It is easy to see that the modelling approach can be generalised by encoding the quantities of species as variables and disregarding the reactions where the reaction rate is zero due to some reactant having zero molecules.

Alternatively, when there is an abundance of reactants (the number of molecules is large enough that changes can be viewed as continuous) the system of reactions can be modelled as a dynamical system using ordinary differential equations (ODE):

$$\frac{d[\mathsf{C}]}{dt} = +k_{on}[\mathsf{A}]\cdot[\mathsf{B}] - (k_{off}+k_{deg})\cdot[\mathsf{C}] \quad \text{(E.1)}$$

$$\frac{d[\mathsf{A}]}{dt} = -k_{on}[\mathsf{A}]\cdot[\mathsf{B}] + k_{off}\cdot[\mathsf{C}] \quad \text{(E.2)}$$

$$\frac{d[\mathsf{B}]}{dt} = -k_{on}[\mathsf{A}]\cdot[\mathsf{B}] + k_{off}\cdot[\mathsf{C}] \quad \text{(E.3)}$$

In our examples we will assume the initial conditions as follows: [A]=50m, [B]=80m, [C]=0m.

2.1 ANIMO

ANIMO [105, 106] is a tool developed for modelling biological pathways. In Figure E.2 we provide a screenshot of the main window of the tool. At the center of the screen is a model of our running example. In this model the nodes represent species and the edges represent reactions. To the right of this is presented a single simulation of the model. Below this simulation is a slider that the user can use to zoom into a specific time point of the simulation: during this movement the representation of the model is updated such that the colouring of the nodes represent the quantity(colouring scheme is shown in the left of the screen) of each species and the thickness of the edges represent how likely that reaction is to occur next. In this manner ANIMO gives a visual representation of the current state of the pathway. In ANIMO a molecule can be in an inactive or active state and the reactions of the pathway may alter the state of a single molecule[4]. If a reaction activates a molecule we call it an activation reaction and if it deactivates a molecule, we call it an inhibitory reaction. In the graph an edge $A \to C$ means A activates C and $A \dashv C$ means A inhibits C. In ANIMO the amount of each species is fixed and given as a user defined value known as the number of activation levels. An activation reaction increases the current activation level of a species and inhibitory reactions decrease the current activation level. Initially the species starts at an activity level defined by the user.

The example in Figure E.2 is the model of our running example. Because reactions in ANIMO can only influence one species (and condition its behaviour on others) the model does not correspond exactly to the description. For instance, the reaction binding A and B molecules into a C molecule is split into three separate reactions: one increasing the activity level of C and two decreasing the activity level of A and B respectively. Similarly, the splitting of a C molecule into A and B is given as three reactions. A final difference from

[4]In fact the colouring of the nodes represent how large a fraction of each species is active

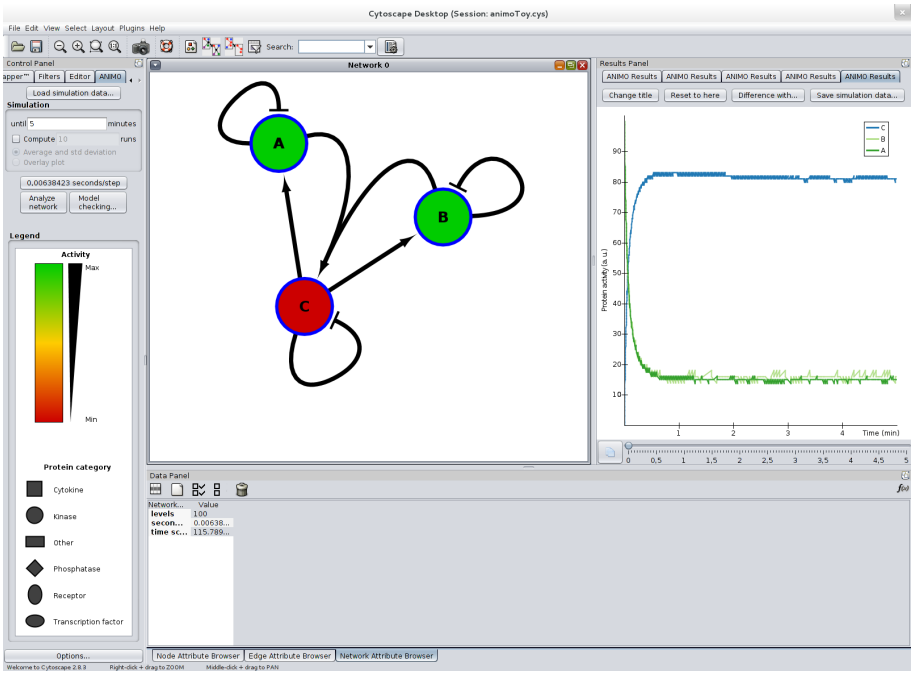


Figure E.2: Screenshot from ANIMO.

the description is that the model does not incorporate the decay of C molecules into "something else".

The graph view of a pathway shows how species influence each other but provides no visual means to indicate how the species influence the time before a reaction occurs. The user controls this by choosing an appropriate reaction type and by setting a reaction rate k. Also the user should provide a scaling factor `timeScale` that translates the model time units to real-world time units.

ANIMO supports three kinds of reactions:

- A reaction $C \to A$($C \dashv A$) is a type-1 activation (inhibition) reaction, if the time before the reaction occurs only depend on the activity level of A,
- a reaction $E \to F$ ($E \dashv F$) is a type-2 activation (inhibition) reaction, if the time before the reaction occurs depends on the activity level of E and the inactivity (activity) level of F and
- the last reaction type supported by ANIMO is a type-3 reaction. In this reaction scheme two species, called reactants, inhibits/activates a third species - an example of this reaction is the double arrow from A and B to C in Figure E.2. The time before the reaction occurs depend on the activity level of the reactant, the inactivity levels of the reactants or the

activity of one and inactivity level of the other reactant.

The time before a reaction occurs is selected in the interval $[0.95 \times f, 1.05 \times f]$ where f is calculated differently depending on the reaction type as described by Schivo et al. [105].

In ANIMO the pathway is translated into a network of timed automata. We notice that it is more natural to use exponential distributions for reaction times compared to the uniform distributions in ANIMO.

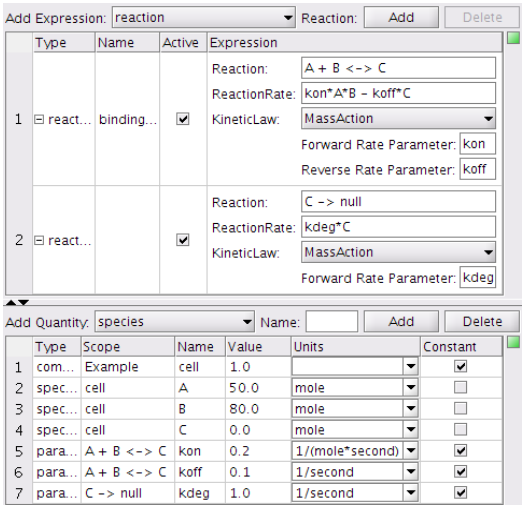
2.2 MATLAB SimBiology®

MATLAB SimBiology® is a tool for modelling, simulation and analysis of dynamic systems with a focus on pharmacokinetics/pharmacodynamics and systems biology. The tool features an editor with textual and graphical notation to model chemical reactions. For example, the screenshot in Figure E.3a shows that reactants, reaction rates and kinetic laws of our basic example can be specified at the upper half, and the quantities with units can be declared at the lower half. Besides simple reaction specification, the tool provides means of grouping reactions into compartments allowing to model physical isolation of materials, also the coefficient declarations can be separated by a reaction scope allowing reuse of variable names. The reactions can be specified using graphical notation interchangeably with textual notation. For example our reactions were automatically rendered in graphical diagram shown in Figure E.3b: species are drawn as blue ellipses, reactions appear as yellow circles, lines denote participating reactants and arrows point to reaction products. Lines are dashed if a reactant is only participating as a catalyst but is not consumed (a few instances are shown in Figure E.12).

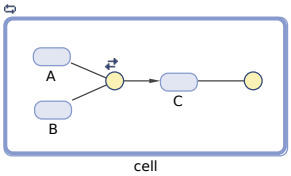
Once the model is complete, SimBiology® can simulate the model as either dynamical system using stiff numerical differentiation formula solver (`ode15s` stiff/NDF, the plot is shown in Figure E.3c) or stochastic simulation using `ssa` solver (plot shown in Figure E.3d). The simulation can be accelerated by compiling the model into native executable code. Note that the solver has to be selected for entire simulation and thus stochastic and dynamical phenomena cannot be combined in one simulation.

3 UPPAAL SMC

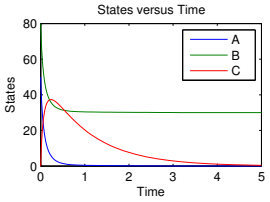
The verification tool UPPAAL [21, 86] provides support for modelling and efficient analysis of real-time systems modelled as networks of timed automata [7]. To ease modelling, the tool comes equipped with a user-friendly GUI for defining and simulating models. Also, the modelling formalism extends basic timed automata with discrete variables over basic, structured and user-defined types that may be modified by user-defined functions written in a UPPAAL specific C-like imperative language. The specification language of UPPAAL is a fragment of



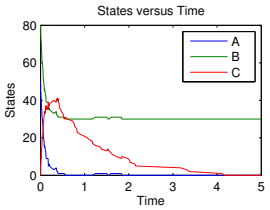
(a) Modelled reactions.



(b) Graphical editor.



(c) Dynamical output.

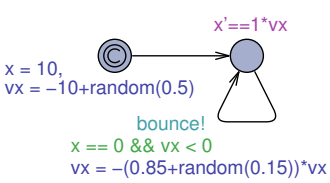


(d) Stochastic output.

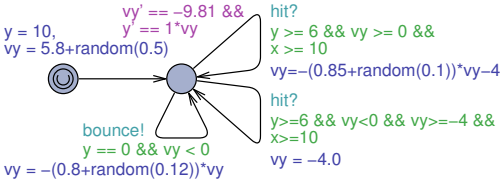
Figure E.3: MATLAB SimBiology® modelling and simulation using different solvers.

timed computation tree logic supporting a range of safety, liveness and bounded liveness properties.

UPPAAL SMC is a recent branch of UPPAAL which supports statistical model checking of *stochastic hybrid systems*, based on a natural stochastic semantics [47]. Firstly, UPPAAL SMC extends the basic timed automata formalism of UPPAAL by allowing rates of clocks to be defined by general expressions possibly including clocks, thus effectively defining ODEs. Secondly, UPPAAL SMC comes equipped with a stochastic semantics [45] that refine the non-deterministic choices that may exist with respect to delay, output as well as next state. For delay of individual components, uniform distributions are applied for states where delay is bounded, and exponential distributions (with location-specified rates) are applied for the cases where a component can remain indefinitely in a location. Also, UPPAAL SMC provides syntax for assigning discrete probabilities to different outputs as well as specifying stochastic distributions on next-states (using the function random(b) denoting a uniform distribution on $[0, b]$). The stochastic semantics of a network is given in terms of repeated races between the constituent components, where in each

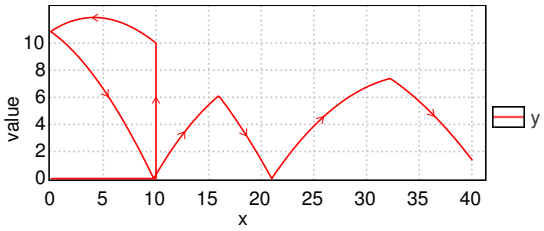


(a) Model of x coordinate.



(b) Model of y coordinate.

(c) Player trying to hit with exponentially distributed delay.



(d) Trajectory as (x, y) plot.

Figure E.4: Models and a trajectory of a thrown/bouncing ball hit by a player.

round the component choosing to output at the earliest time-point is the winner.

The specification formalism of UPPAAL SMC is that of *weighted metric temporal logic*(WMTL) [34, 35] with respect to which four different (simulation-based) statistical model checking queries are supported: hypothesis testing, probability estimation, probability comparison and simulation. Here the user may control the accuracy of the analysis by a number of statistical parameters (size of confidence interval, significance level, etc.). UPPAAL SMC also provides distributed implementations of the hypothesis testing and probability estimation demonstrating linear speed-up [35].

Throwing & Bouncing Ball To give an illustration of the expressive power of the modelling formalism of UPPAAL SMC, we consider a variant of the well-known bouncing ball. In our version the ball is initially thrown against a wall, bounces against it, and then continues its trajectory by falling and bouncing against the floor. In addition, an inexperienced player tries to hit the ball randomly according to an exponential distribution. The model is depicted in Figure E.4(a)–(c). The player is modelled as a simple automaton that broadcasts `hit!` with an exponential distribution of rate $\frac{5}{2}$. The `x` coordinate (Figure E.4a) is initialised to 10 with an uncertain derivative `vx` uniformly distributed between $[-10, -9.5]$ (the ball is thrown against the wall), after which the ball moves toward the wall (placed at 0). Here, the automaton outputs bounce! on an *urgent* channel, which forces the transition to take place deterministically at `x=0`. After

a bounce with a random dampening factor of the velocity vx uniformly between $[0.85, 1]$, the ball continues to move in the opposite direction. The y coordinate (Figure E.4b) is initialised to 10 with an uncertain derivative vy. The model shows the effect of gravitation with vy'=−9.81. The ball bounces with a random dampening factor on the floor (at 0) and when the ball is away from the wall (x>= 10) then it can be hit by the player provided it is high enough (y>= 6). Depending on the current direction of the ball, the ball may bounce or it is pushed. One possible trajectory of the ball is shown in Figure E.4d. The plot is obtained by checking the query "simulate 1 [x<=40] {y}". The vertical line shows the ball moving to its initial position and should be ignored. The ball bounces as expected against the wall, the floor, and the hitting of the player. Uppaal SMC is able to simulate this hybrid system that has a second order ODE, a stochastic controller (the player), and a stochastic environment (random dampening factor).

In addition, we may perform SMC in order to estimate the probability that the ball is still bouncing above a height of 4 after 12 time units with the query:

Pr[<=20](<> time>=12 and y>=4)

which returns the confidence interval $[0.44, 0.55]$ with 95% confidence after having generated 738 runs. We can also test for the hypothesis

Pr[<=20](<> time>=12 and y>=4) >= 0.45,

which gives a more precise lower bound. The hypothesis holds with a region of indifference ± 0.01 and a level of significance of 5% after generating 970 runs. In the queries the [<=20] tells the engine that the property should be true within the first 20 model time units hence the sample runs should only have a length of 20 model time units..

Uppaal SMC, and in principle any statistical model checker, consist of three components: A generator, a validator and a core algorithm as depicted in Figure E.5. The generator takes as input a model M and produces a single run of a given length, the validator takes as input a property ϕ and a run and returns `true` if the run satisfies the property and `false` otherwise. This control flow is shown by the solid lines in Figure E.5. The core algorithm essentially "keeps score" of the number of `true` and `false` results, in order to perform hypothesis testing or estimation with respect to the unknown probability $\mathbb{P}_M(\phi)$ using classical statistical algorithms. For hypothesis testing, additional information with respect to required significance level (α_1) and threshold probability (θ) is required. For estimation, information concerning the size of the confidence interval (δ) and the required confidence level (α_2) is required.

In order to increase performance, Uppaal SMC is validating a run while it is generated. Here the generator passes a single state forward to the validator and the validator can ask for the next state in the run being generated or may pass `true` or `false` to the core algorithm. This one-step validation is shown by the dashed lines i Figure E.5 and is essentially made to avoid generating and

storing long runs, e.g. 1000 steps, if the property may already be settled after a few, e.g. two, steps.

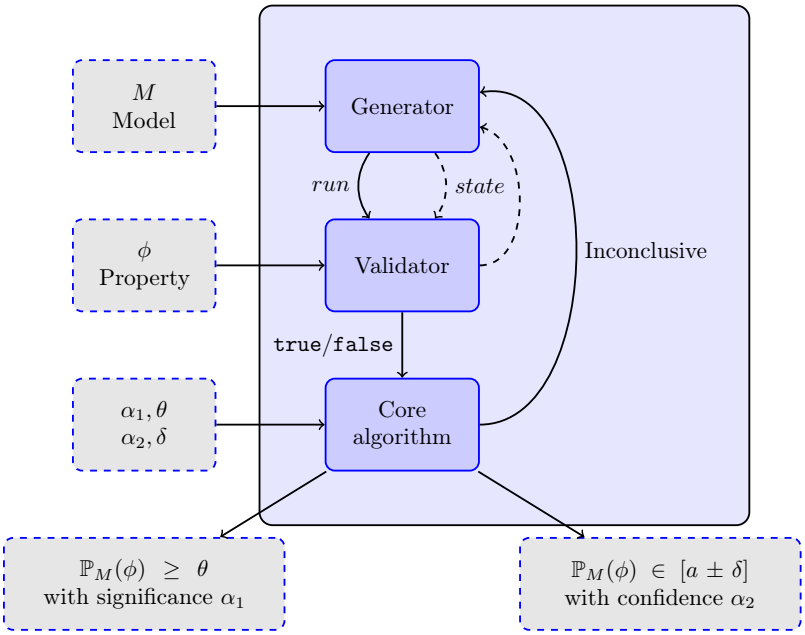


Figure E.5: High level view of a statistical model checker for probability estimation and hypothesis testing. The dashed lines correspond to UPPAAL SMC optimised generation/validation loop and the solid lines to a generic statistical model checker.

Weighted Metric Temporal Logic Besides pure reachability properties UPPAAL SMC also supports Weighted MTL (WMTL) properties with a point-wise semantics. An example of a WMTL property is that the ball within 4 but not earlier than 2 time units should reach a height greater than 6 and afterwards within 1 time unit should drop below 4. In WMTL we can describe this as

$$(\mathsf{tt}\,\mathsf{U}_{[2;4]}\,(y > 6 \wedge (\mathsf{tt}\,\mathsf{U}_{[0;1]}\,y < 4)),$$

in which an expression as $\phi_1 \mathsf{U}_{[a;b]} \phi_2$ should be interpreted as ϕ_1 must be true until ϕ_2 is true and ϕ_2 must be true before b time units have passed but not before a time units.

The syntax of WMTL extends propositional logic with time- and weight-constrained Until and Release modalities:

$$\varphi ::= \mathsf{tt} \mid \mathsf{ff} \mid p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U}^{\mathsf{x}}_{[a;b]} \varphi_2 \mid \varphi_1 \mathsf{R}^{\mathsf{x}}_{[a;b]} \varphi_2$$

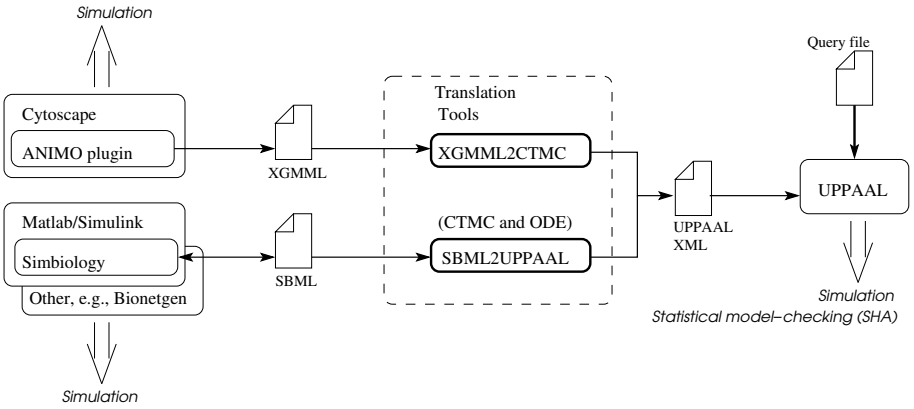


Figure E.6: Overview of our tool chain.

where x is a clock that for any infinite run will exceed any given bound of the system (we omit this clock when we bound over the global time), p is an atomic expression, $a, b \in \mathbb{Q}$ and $a < b$. An expression $\phi_1 \mathsf{R}_{[a;b]} \phi_2$ means that ϕ_2 must be true until both ϕ_1 and ϕ_2 are true and they should be true after a time units and before b time units. Alternatively ϕ_2 is true from now and until b time units have passed. The expression $\mathsf{X}\varphi$ means φ should be true in the next observation.

In Uppaal SMC there are two different ways of using WMTL. For the MTL fragment of WMTL, a MTL property can be passed directly to the engine [34]. For the full WMTL language, the formula can be converted into an observer automaton and that automaton be parallel composed into the system [35]. The observer is guaranteed to reach a specific location if the property is satisfied and another if its not satisfied thus we can use the optimised reachability engine of Uppaal SMC to verify WMTL. There is a catch though - sometimes an exact observer cannot be constructed and only an over or underapproximation can be made.

In adition to verify properties in WMTL, we can also use the observers to, at run time, detect certain phenomena of the system (peaks for instance). By coordinating multiple observers we can detect the period of the periodic behaviours of the observed system. In Section 5.2 we will see an example of this usage.

4 Translators

4.1 Overview

We have implemented two tools to translate (i) network description files (XGMML) with the semantics of the ANIMO plugin, and (ii) a subset of the

standard format SBML[5] to UPPAAL SMC stochastic hybrid automata (XML). Figure E.6 gives an overview of our translations. We can export ANIMO models to XGMML within the Cytoscape tool and translate this format into a CTMC UPPAAL SMC model. From the SimBiology® plugin we export SBML that we can translate to both a CTMC or an ODE UPPAAL SMC model. We note that we could connect to other tools that use this standard format, such as *BioNetGen*. The translators complement the functionalities of other tools by offering statistical model-checking. In addition, we can envision that UPPAAL SMC could be used as a backend to other tools that translate models from a domain specific formalism to stochastic hybrid automata.

Implementation and Availability The translators are implemented in C++. The XGMML translator uses the *rapidxml* library and the SBML translator the *libsbml* library. Our translators are available with the distribution of UPPAAL version 4.1.14. The translator for XGMML interprets the network according to the semantics of ANIMO only. The translator for SBML is general and has been tested against the BioModels database of biological models[6].

4.2 General Principle of The Translations

To explain the general principles of our translations, we consider the following example that is representative of the types of reactions we support:

$$A + B \underset{k_{off}}{\overset{k_{on}}{\rightleftharpoons}} C \xrightarrow{k_{deg}}$$

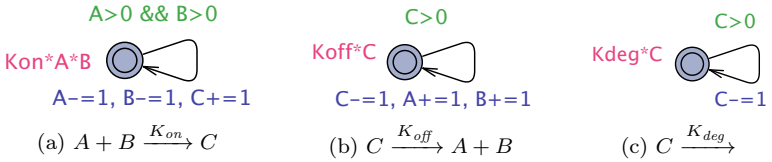
Here, A and B are reactants, C is the product of the first reaction, and C is itself a reactant for a second reaction where it degrades. In general, we can have more reactants or products. To model these reactions we need to pick a kinetic law V for each of them (in function of the concentrations of the reactants). If we separate the reactions we have:

$$\begin{aligned} A + B &\xrightarrow{k_{on}} C & V_{on} &= k_{on}[A][B] \\ C &\xrightarrow{k_{off}} A + B & V_{off} &= k_{off}[C] \\ C &\xrightarrow{k_{deg}} & V_{deg} &= k_{deg}[C] \end{aligned}$$

We note that the kinetic laws may contain some extra terms, e.g., concentrations of catalysts that are not explicitly present in the reactions. These reactions can be modelled as CTMCs (the stochastic model) or with ordinary differential equations (the ODE model).

Basic Stochastic Model The stochastic model for these reactions is a network of simple stochastic hybrid automata. The template for each reaction has one edge with:

[5] `http://sbml.org`
[6] `http://www.ebi.ac.uk/biomodels-main/publmodels`



(a) $A + B \xrightarrow{K_{on}} C$ (b) $C \xrightarrow{K_{off}} A + B$ (c) $C \xrightarrow{K_{deg}}$

Figure E.7: The three stochastic hybrid automata to model the three reactions of our example.

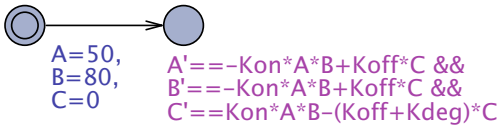


Figure E.8: The ODE hybrid automaton to model the three reactions of our example.

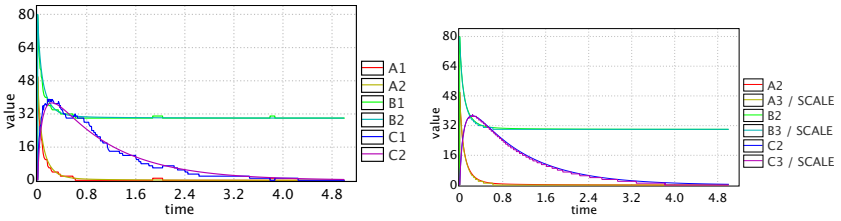
- A guard that ensures there is enough reactant, in our case A>0 && B>0 for the first reaction of our example and
- updates that encode the actual reaction, e.g., A−=1, B−=1, C+=1.

Furthermore, the kinetic law is encoded in the exponential rate that define the distribution of the delay to take this transition. For this example we have Kon∗A∗B. Figure E.7 shows the three stochastic automata used to model the three separate reactions of our example. The automatic generation of these automata is straight-forward from the reactions. The variables used are integers since we only need counters. This may require rescaling the values of the initial reactions.

Basic ODE Model To model with ODEs, we derive the equations from the reactions and their kinetic laws. The derivative for each reactant is the weighted sum (positive if produced or negative if consumed) of the kinetic laws of the reactions that involve this reactant. For our example, we obtain the derivatives of equations E.1, E.2 and E.3 in Section 2.

To model these equations in Uppaal SMC, we use one automaton with one transition from an initial state to a main state to initialise the reactants, and then the equations are declared in the invariant of the main state as shown in Figure E.8. In practice, the values of the constants are inlined in the model.

The generation of the model is done by first collecting the reactants and the different sums for their derivatives and second by writing the resulting list of rates. The variables are clocks here. We do not need to rescale values as for the stochastic model.



(a) Unscaled CTMC and ODE simulations. (b) Scaled CTMC and ODE simulations.

Figure E.9: Simulation results of the CTMC and ODE models for our running example. The $\{A, B, C\}1$ variables are from the non-scaled CTMC model, the $\{A, B, C\}2$ variables are from the the ODE model and the $\{A, B, C\}3$ variables are from the scaled CTMC model.

Example We show the results of the simulation of our example in Figure E.9. Figure E.9(a) compares the simulations of the CTMC and the ODE models (the ODE corresponding to the smooth plot). To increase the precision of the CTMC simulation (done with small numerical values), we can scale the amounts by 100 (and correct the kinetic laws). Figure E.9(b) plots the scaled-down values from the scaled model. The plot shows how the CTMC is now much closer to the ODE solution and is in fact its discretization. This illustrates that by taking large amount of reactants, the behaviour converges to the solution of the ODEs.

4.3 ANIMO to UPPAAL SMC

ANIMO models are exported to the XGMML format, which is a general XML based format for representing networks. Our tool recognises the types of nodes and their semantics used by ANIMO. The translation from ANIMO follows our general principle for translating the reactions into CTMCs. Let A, B and C be species with nA, nB and nC activation levels and let the time scale in ANIMO be ts, then the translation is performed as shown in Table E.1.

Recall that the reaction time in ANIMO was chosen uniformly in the interval $[0.95 \times f; 1.05 \times f]$, f being calculated differently depending on the type of reaction. The exponential rate we use in our translation to CTMCs correspond to $\frac{1}{f}$ thus we ensure the average reaction time is the same for our translation and the semantics for ANIMO.

4.4 SimBiology® to UPPAAL SMC

SimBiology® models are exported to SBML, a standard format to describe biological systems. The SBML language has been designed for biologists and can be used by several tools, in particular BioNetGen or SimBiology®. SBML is rich and is unfortunately not used in a standard way nor do different simulators agree on simulation results [23], which shows that handling general models is difficult: As an example, for the model #24, only six out of 12 simulation

	Type	Guard Exponential	Update
$A \dashv B$	1	A > 0 && B−1>0 nB/(nA∗ts)∗k∗A	B=B−1
	2	A>0 && B−1 >=0 && B>0 1/(nA∗ts)∗k∗A∗B	B=B−1
$A \to B$	1	A > 0 && B+1<=nB nB/(nA∗ts)∗k∗A	B=B+1
	2	A>0 && B+1 <=nB && (nB−B)>0 1/(nA∗ts)∗k∗A∗(nB−B)	B=B+1
$A, B \dashv C$	3	f(A,actA) > 0 && f(B,actB)>0 && C−1>0 nB/(nA∗nB∗ts)∗k∗f(A,actA)∗f(B,actB)	C=C−1
$A, B \to C$	3	f(A,actA) > 0 && f(B,actB)>0 && C−1>0 nB/(nA∗nB∗ts)∗k∗f(A,actA)∗f(B,actB)	C=C+1

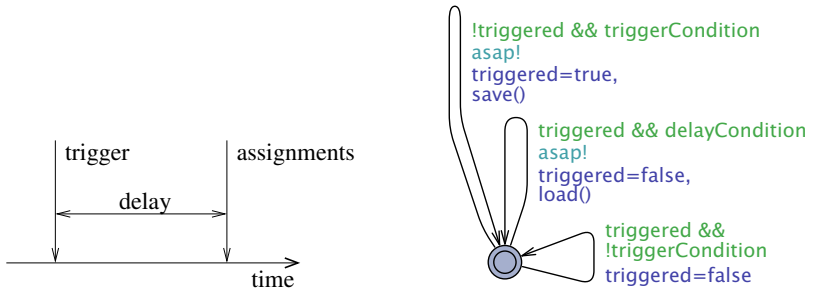
Table E.1: Overview of the translation where $actA, actB \in \{active, inactive\}$, $f(X, active) = X$ and $f(X, inactive) = nX - X$.

packages returned a result *and* only two of these seemed to agree on a specific behaviour. We choose to support a subset of SBML and to judge if it is relevant and to assess the validity of our translations, we apply our tool to all the 436 models of the BioModels database. According to [23],

> " *The curated models in the BioModels Database cover a wide range of features of the SBML language and are therefore an optimal choice as a base set of models for simulator comparison.* "

Translation To translate SBML models we use both the basic stochastic model and the ODE models that we need to extend to accommodate the extra features of SBML. We mention the following features that need extra handling:

- Functions: User functions in the form of *lambda expressions* can be defined. These functions return floating point values, which is not yet supported by Uppaal SMC, so we inline them in the model.
- Compartments: Species exist in compartments that have volumes, e.g., a cell. We use compartments as namespaces.
- Species: Species (names for the reactants and products) are declared with either initial amounts or initial concentrations (this is inconsistent across models). We choose on-the-fly the right initial state. In addition, quantities are not integers in general so our tool suggests a scaling factor (option `-scale value`) to scale the values and update the kinetic laws automatically.
- Parameters: Models can be parameterized by global or local parameters (local to kinetic laws). We inline the definition of these parameters in



(a) Time frame of events.

(b) Simplified template for encoding events.

Figure E.10: Events in SBML and their encodings.

formulas where they are used if they are constants. If parameters are not constants, they are typically used in event assignments, in which case they are added to the model as extra variables.

- Assignment and rate rules: The models may contain reactions and explicit equations that define how species evolve. The explicit definition of $[A] = expr$ is an assignment rule and $d[A]/dt = expr$ is a rate rule. Sometimes assignment or rate rules are present together with reactions involving the same species and they may conflict. When rules are given, we consider that they override any equation that we may infer from the reactions. Assignment rules are inlined in formulas and rate rules override the derivatives. It is forbidden to have both assignment and rate rules for the same species.

- Names: Species are not used directly in the model, but rather species references (to some identifier). Sometimes the identifier is misused as the name and sometimes the name of the species is the right name. By default our translator uses IDs but there is an option (`-name`) to use the names of the species instead. This helps to understand the output.

- Events: External events to the reactions can occur, e.g., an operator pours a reactant or a cell divides. They are akin to discrete transitions in hybrid systems. An event is triggered when its trigger condition is true. Event assignments are performed after an optional delay with either the state or the state when the trigger occured (that means it has to be saved). Events can be *persistent*. A non-persistent event means that the trigger condition has to hold until the event assignment takes place otherwise it is cancelled. Figure E.10(a) shows the time frame of such events. Figure E.10(b) shows the extra transitions for each event that are added to the basic ODE model[7]. The flag `triggered` records

[7]They can be added to the CTMC model as well but this is not yet implemented.

Inconsistent models	2.7%
Unsupported features	25%
CTMC and ODE	10.1%
CTMC or ODE	30.3 %
Failed simulations	31.9%

Table E.2: Synthetic results on the database in percentage of the 436 models.

the occurrence of the event, `triggerCondidion` is the trigger condition, `delayCondition` is the optional delay condition, `save()` is replaced by saving the clocks needed for the assignment if the values at the trigger are needed, and `load()` is replaced by the actual event assignments that read either the current values or the values saved. The bottom transition can disable the event if it is not persistent.

We do not support the other features. In particular, we currently assume that the units in the model are consistent. Units can be handled via rescaling and this is not a fundamental limitation. On the other hand, algebraic rules that define equations of the form $f(x_i) = A$ where $f(x_i)$ is some arbitrary function depending on species and A a constant, are more problematic. In practice they are rarely used (not at all in the BioModels database) so this is not a limitation either.

Validating the Tool To validate and assess the tool we use the following methodology. We download all curated models from the BioModels database. A curated model is a manually sanitised model. We run our translator on all of them and then we (manually) simulate all of those that were successfully translated. We record the following results:

- We found models with negative initial amounts or that use reaction identifiers as variables. They cannot be translated and are considered to be *inconsistent*.
- We failed to translate some models that use unsupported features.
- For the models we successfully translate (72.3%), we can simulate CTMC or ODE models (or both)[8]. The simulations may also fail mainly due to numerical problems.

The detailed results of our validation experiments can be found in Table E.3 and are summarised in Table E.2.

These experiments show that we can obtain meaningful simulations *automatically out of 40.4% of all the models from this database*, which is a positive result considering that these models are not easy to handle even by specialised tools in biology. This shows that Uppaal SMC can realistically be used as

[8]The individual scaling or simulation steps are not reported here for brevity.

Nr.	Res.	Nr.	Res.	Nr.	Res.	Nr.	Res.	Nr.	Res.	Nr.	Res.	Nr.	Res.
001	-	002	x	003	ode	004	ode	005	ctmc	006	*ode*	007	x
008	ode	009	x	010	both	011	both	012	both	013	x	014	both
015	x	016	x	017	x	018	x	019	-	020	x	021	ode
022	ode	023	ode	024	-	025	-	026	ode	027	both	028	both
029	both	030	both	031	ode	032	x	033	x	034	-	035	both
036	ode	037	both	038	x	039	x	040	x	041	ode	042	ode
043	ctmc	044	ctmc	045	ctmc	046	x	047	-	048	x	049	both
050	both	051	-	052	both	053	x	054	ode	055	-	056	x
057	ode	058	ode	059	x	060	ode	061	ctmc	062	ode	063	!
064	ctmc	065	ode	066	ode	067	ode	068	x	069	ode	070	ctmc
071	x	072	x	073	ode	074	ode	075	-	076	both	077	-
078	-	079	ode	080	x	081	-	082	x	083	-	084	ode
085	x	086	x	087	x	088	-	089	x	090	ctmc	091	ctmc
092	both	093	both	094	both	095	-	096	-	097	-	098	ode
099	ode	100	x	101	-	102	ode	103	ode	104	-	105	both
106	ode	107	ode	108	ctmc	109	-	110	ode	111	x	112	-
113	ode	114	ode	115	ode	116	ctmc	117	-	118	x	119	x
120	-	121	-	122	-	123	x	124	-	125	-	126	-
127	-	128	-	129	-	130	-	131	-	132	-	133	-
134	-	135	-	136	-	137	-	138	x	139	-	140	-
141	-	142	-	143	x	144	ode	145	ode	146	ode	147	ode
148	ctmc	149	-	150	x	151	x	152	-	153	-	154	-
155	-	156	ode	157	ode	158	x	159	ode	160	ode	161	-
162	-	163	x	164	-	165	-	166	ode	167	-	168	x
169	ode	170	both	171	-	172	x	173	-	174	-	175	x
176	x	177	x	178	x	179	-	180	-	181	ode	182	x
183	x	184	ode	185	ode	186	-	187	-	188	-	189	-
190	*ode*	191	both	192	x	193	-	194	-	195	x	196	-
197	both	198	ode	199	x	200	x	201	x	202	*ode*	203	x
204	x	205	x	206	ode	207	*ode*	208	-	209	x	210	x
211	ode	212	x	213	x	214	-	215	-	216	ode	217	ode
218	x	219	x	220	x	221	x	222	x	223	x	224	ode
225	ode	226	x	227	-	228	ode	229	ode	230	ode	231	ctmc
232	x	233	ode	234	x	235	-	236	ode	237	-	238	x
239	ode	240	ode	241	-	242	ode	243	both	244	-	245	!
246	x	247	x	248	!	249	ode	250	x	251	x	252	x
253	x	254	*ode*	255	-	256	-	257	ode	258	ode	259	both
260	both	261	both	262	-	263	-	264	-	265	x	266	x
267	both	268	-	269	ode	270	x	271	both	272	x	273	-
274	*ode*	275	ode	276	*ode*	277	*ode*	278	x	279	x	280	x
281	-	282	x	283	both	284	ode	285	x	286	-	287	-
288	x	289	x	290	x	291	x	292	ode	293	both	294	ode
295	-	296	ode	297	x	298	*ode*	299	*ode*	300	x	301	-
302	x	303	x	304	x	305	!	306	x	307	x	308	x
309	both	310	x	311	x	312	-	313	ode	314	ode	315	both
316	-	317	-	318	-	319	ode	320	ode	321	ode	322	ode
323	ode	324	x	325	!	326	-	327	-	328	both	329	*ode*
330	x	331	x	332	ode	333	ode	334	ode	335	x	336	x
337	-	338	-	339	-	340	-	341	*ode*	342	-	343	x
344	x	345	x	346	x	347	x	348	x	349	x	350	-
351	-	352	-	353	*ode*	354	ctmc	355	x	356	-	357	both
358	both	359	ode	360	ode	361	both	362	x	363	ode	364	ode
365	ode	366	ode	367	*ode*	368	x	369	x	370	x	371	!
372	x	373	!	374	!	375	!	376	!	377	!	378	!
379	*ode*	380	x	381	*ode*	382	x	383	x	384	x	385	x
386	x	387	x	388	ode	389	ctmc	390	x	391	x	392	x
393	x	394	ode	395	ode	396	ode	397	both	398	both	399	-
400	x	401	*ode*	402	*ode*	403	*ode*	404	-	405	x	406	x
407	x	408	-	409	ode	410	ode	411	-	412	-	413	both
414	ode	415	x	416	x	417	both	418	ode	419	ode	420	ode
421	ode	422	-	423	x	424	x	425	ode	426	x	427	ode
428	x	429	-	430	both	431	both	432	both	433	both	434	x
435	ode	436	-										

Table E.3: Detailed results on all 436 (curated) SBML models from the database available at `http://www.ebi.ac.uk/biomodels-main/publmodels`. The results are reported as follows: "!" means the model was inconsistent, "x" means the simulation failed, "-" means the translation failed, "ctmc" means only the CTMC simulation worked, "ode" means only the ODE simulation worked, and "both" means both simulations worked. Italic "ode" means that the CTMC model could not be generated because of the presence of rate rules in the SBML file (it is not possible to get a CTMC model)

a backend tool to handle real-world models. In the next section, we study in details one of these models (number 35).

Remark 12. Some of the models used for the experiments are stiff and the step size has to be adjusted for the integration to work. In particular, the biological oscillator (model 35) needs a step size of 10^{-4}.

5 Case Studies

In this section, we demonstrate the intended usage and benefit of the developed tool chain on two case studies.

5.1 PC-12 Neural Pathways

We consider the case that the authors of [106] used to exemplify the use of ANIMO. The case is a pathway that coordinates the neural differentiation of PC-12 cells. The original ANIMO model was obtained from the authors and afterwards translated into a UPPAAL SMC model using `xgmml2ctmc`. We use this model to compare the model of ANIMO with our translated model. For the experiments we have used version 2.55 of ANIMO.

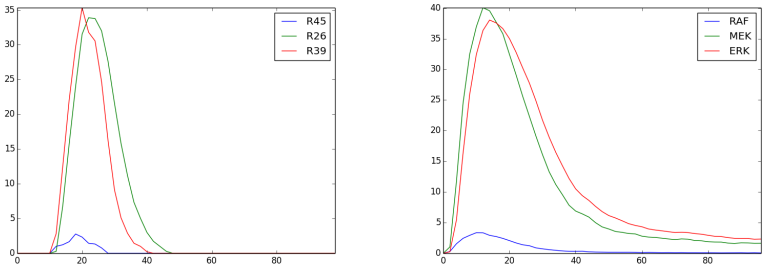
For the ANIMO model we generated 100 runs and took the average on each sampling point to obtain an "expected" run. We did similarly for our translated model but for 10000 runs - we used a greater number of runs for our model to accommodate for the higher variability that the exponential distributions give.

In Figure E.11a we have plotted the expected run from ANIMO and in Figure E.11b we show the expected run of UPPAAL SMC. By visual inspection we notice that the graphs of ANIMO and UPPAAL SMC are not lining up perfectly. However, they do share a common structure where ERK, and MEK rises rapidly in the beginning and then drops and approaches zero as the time progresses. Similarly RAF rises in the beginning and drops afterwards. The primary difference in the runs is that the UPPAAL SMC run takes longer to reach the maxima for the species and equivalently takes longer in approaching zero. This difference may easily be explained by our use of exponential distribution versus the uniform distributions of ANIMO.

5.2 Genetic Oscillator

We show how the genetic oscillator [15, 117] can be modelled and simulated in MATLAB SimBiology® exactly as a stochastic process with discrete states, approximated using ODEs that assume continuous states, and then demonstrate statistical model-checking approach using UPPAAL SMC.

The synthetic genetic oscillator distills the essence of several real circadian oscillators to demonstrate how nature constructs a reliable system in the face of inherent stochasticity: the oscillator has been shown to exhibit a kind of regularity referred to as stochastic coherence [76]. The model is based on elemental reactions with mass action kinetics shown in Table E.4 .



(a) A simulation from ANIMO. (b) Average run from UPPAAL SMC.

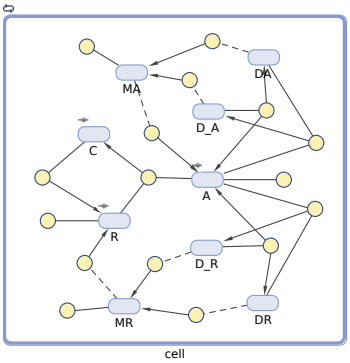
Figure E.11: Comparison between UPPAAL SMC and ANIMO. The time is on the x-axis and on the y-axis we have the activity levels of the variables. In (a) $\mathsf{R45} = \mathsf{RAF}$, $\mathsf{R26} = \mathsf{MEK}$ and $\mathsf{R39} = \mathsf{ERK}$

.

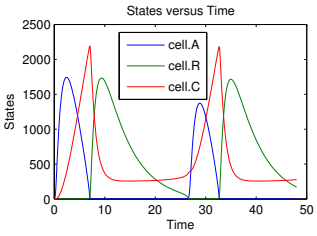
$$\mathsf{A} + \mathsf{D}_A \xrightarrow{\gamma_A} \mathsf{D}'_A \qquad \mathsf{M}_A \xrightarrow{\beta_A} \mathsf{M}_A + \mathsf{A}$$
$$\mathsf{D}'_A \xrightarrow{\theta_A} \mathsf{D}_A + \mathsf{A} \qquad \mathsf{M}_R \xrightarrow{\beta_R} \mathsf{M}_R + \mathsf{R}$$
$$\mathsf{A} + \mathsf{D}_R \xrightarrow{\gamma_R} \mathsf{D}'_R \qquad \mathsf{A} + \mathsf{R} \xrightarrow{\gamma_C} \mathsf{C}$$
$$\mathsf{D}'_R \xrightarrow{\theta_R} \mathsf{D}_R + \mathsf{A} \qquad \mathsf{C} \xrightarrow{\delta_A} \mathsf{R}$$
$$\mathsf{D}'_A \xrightarrow{\alpha'_A} \mathsf{M}_A + \mathsf{D}'_A \qquad \mathsf{A} \xrightarrow{\delta_A} \emptyset$$
$$\mathsf{D}_A \xrightarrow{\alpha_A} \mathsf{M}_A + \mathsf{D}_A \qquad \mathsf{R} \xrightarrow{\delta_R} \emptyset$$
$$\mathsf{D}'_R \xrightarrow{\alpha'_R} \mathsf{M}_R + \mathsf{D}'_R \qquad \mathsf{M}_A \xrightarrow{\delta_{M_A}} \emptyset$$
$$\mathsf{D}_R \xrightarrow{\alpha_R} \mathsf{M}_R + \mathsf{D}_R \qquad \mathsf{M}_R \xrightarrow{\delta_{M_R}} \emptyset$$

$\alpha_A = 50\text{h}^{-1}$	$\delta_{M_R} = 0.5\text{h}^{-1}$
$\alpha'_A = 500\text{h}^{-1}$	$\theta_A = 50\text{h}^{-1}$
$\alpha_R = 0.01\text{h}^{-1}$	$\theta_R = 100\text{h}^{-1}$
$\alpha'_R = 50\text{h}^{-1}$	$\mathsf{D}_A = 1\text{m}$
$\beta_A = 50\text{h}^{-1}$	$\mathsf{D}'_A = 0$
$\beta_R = 5\text{h}^{-1}$	$\mathsf{D}_R = 1\text{m}$
$\gamma_A = 1\text{m}^{-1}\text{h}^{-1}$	$\mathsf{D}'_R = 0$
$\gamma_R = 1\text{m}^{-1}\text{h}^{-1}$	$\mathsf{M}_A = 0$
$\gamma_C = 2\text{m}^{-1}\text{h}^{-1}$	$\mathsf{M}_R = 0$
$\delta_A = 1\text{h}^{-1}$	$\mathsf{A} = 0$
$\delta_R = 0.2\text{h}^{-1}$	$\mathsf{R} = 0$
$\delta_{M_A} = 10\text{h}^{-1}$	$\mathsf{C} = 0$

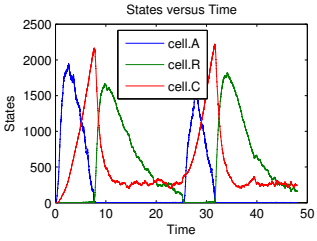
Table E.4: Reactions and initial values for constants and species from the genetic oscillator [117].



(a) Graphical overview of reactions.



(b) Output of ODE solver.



(c) Output of stochastic solver.

Figure E.12: Modelling and simulating the genetic oscillator in SimBiology®.

Oscillations arise in the model as a result of a phase shift between competing processes of production and sequestration of protein A. Genes D_A and D_R are transcribed to messenger RNA M_A and M_R, that are translated to proteins A and R, respectively. A and R dimerise to produce complex C. Protein A acts as a promoter for its own gene, creating a positive feedback loop that causes the production of A to increase rapidly. Protein A also promotes the production of protein R. Hence, as A increases, so too does R, but after a delay due to the two step transcription-translation process. As R increases it sequesters A via the second order dimerisation reaction, thus limiting the maximum amount of protein A. The delay in this negative feedback causes the system to oscillate.

MATLAB SimBiology® is designed specifically for biochemical processes, thus modelling is straightforward and simulation benefits from a similar set of solvers with an important addition of the stochastic one. The remarkable feature of SimBiology® is that simulations can be performed in ensembles and the simulation can be accelerated further by compiling into native executable through translation into C, but eventually the behavioural data is recorded as large MATLAB arrays for later post-processing.

We have modelled the genetic oscillator [117] in our previous study [47, 51] and in this paper we compare the results and performance of UPPAAL SMC with MATLAB SimBiology® and Simulink®. Figure E.12 shows the reaction model and simulation plots using a deterministic ODE and a stochastic solver. The simulated behaviour is identical with our previous results and the translated models to UPPAAL SMC through SBML are equivalent to the ones we studied

Tool	Simulation	Performance, s
Simulink®	fixed 10^{-4} step ode1 (Euler)	3.7900 ±0.11
	variable step ode45 (Runge-Kutta)	0.7700 ±0.02
	variable step ode15s (stiff/NDF)	0.0783 ±0.0065
SimBiology®	ode15s (stiff/NDF)	0.1805 ±0.0017
	ssa (stochastic)	0.2575 ±0.0090
	accelerated ode15s (stiff/NDF)	0.0203 ±0.0015
	accelerated ssa (stochastic)	0.2476 ±0.0054
Uppaal SMC	fixed 10^{-4} step ODE (Euler)	1.7520 ±0.0056
	CTMC (stochastic)	1.0400 ±0.24

Table E.5: Genetic oscillator simulation performance using various tools: the intervals computed using 20 measurements with 95% confidence.

Tool	Simulation	Tuples	Min memory, KB
SimBiology®	ode15s	747	58.4
	ssa	286080 ±7674	22350.0
Uppaal SMC	ODE	5354	418.3
	stochastic	4437 ±34	346.6

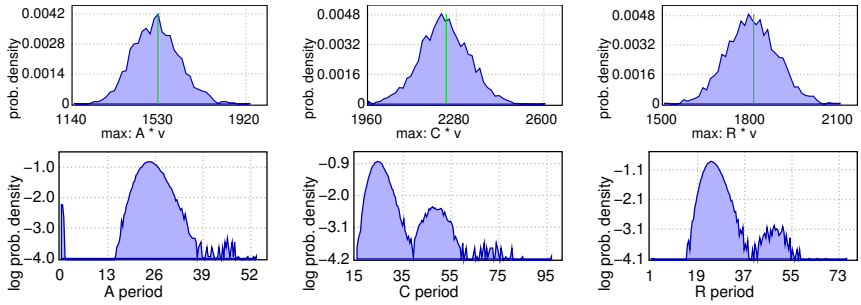
Table E.6: Simulation data comparison.

before. We also modelled the differential equation model in Simulink® and got identical results.

Table E.5 summarises the timing measurements of 100h of model time simulations (containing about four periods of oscillations) with graphical plotting turned off. Matlab family of tools provides a wide range of solvers with varying quality and behaviour. We tried a fixed time step ode1 Euler solver because it is the same method used by the current release of Uppaal SMC. The results show that Uppaal SMC implementation is about two times faster. A default choice in Simulink® is a variable step ode45 solver which is more accurate and faster than Uppaal SMC because it can leap in larger time steps when dynamics change little. Solver ode15s seems to be even a better choice here because it is designed for stiff functions and still fast. The problem with Matlab Simulink® is that resulting models are large[9] and thus the modelling process is tedious and difficult to debug.

Table E.6 shows amounts of data generated for simulation purposes. A tuple of data consists of ten double precision numbers (one for time and nine for species quantities). SimBiology® works very well for deterministic ODE simulations and acceleration can be dramatic, but the stochastic simulations hardly get any benefit from acceleration and can be problematic due to vast amount of generated data while performing small time steps and even simple

[9]e.g. StateSpace approach is not applicable due to multiple species coupling.



(a) A. (b) C. (c) R.

Figure E.13: Estimated probability density distributions for amplitude and period.

ensemble simulation becomes a memory bottleneck in the process. For example, generating ensemble simulation of just 100 runs (without plots), MATLAB's virtual memory grows from 1329MB to 5633MB (504MB to 4600MB of resident memory). The small time steps are inevitable in stochastic simulations of biological systems because the reaction rates are proportional to molecular quantities and some of them can get very high. While it is possible to use the memory more efficiently by analysing one run at a time, it still requires programming and is still constrained to storing at least one entire run.

UPPAAL SMC contains twice as fast Euler ODE integrator, but it is still much slower than variable step solvers like ode45 and ode15s. On the other hand, stochastic simulation is more than 64 times efficient in space due to data filtering and thus is more suitable for interactive exploration. In addition, UPPAAL SMC provides a query language and evaluates the statistical properties on-the-fly by storing only one state at a time, and thus does not have a storage bottleneck. UPPAAL SMC may also terminate simulation earlier due to feedback from its *Validator* and *Core algorithm* to *Generator* that it has acquired enough information about a single run or observed enough runs.

Next, we demonstrate statistical query language application. For example, the amplitude of each protein quantity can be measured by the following query:

$$\mathsf{E[<=100;\ 2000](max:\ Q)}$$

where 100 is the time bound for simulation, 2000 is the number of simulations and Q is a model expression of interest: in our case it is simply one of variables A, C or R. As a result, UPPAAL SMC renders a probability density plot of possible amplitudes and a vertical line for an average value shown in upper plots of Figure E.13. The memory consumption remains about the same: around 40MB of virtual and 3.7MB of resident.

UPPAAL SMC can also estimate a distance between peaks by the construction of monitors for WMTL. The idea of the approach is to translate a WMTL formula that describes the shape of a peak into a monitoring automaton. By

resetting a clock x and start a secondary monitor when a peak is detected we can estimate the distance as the value of x when the second monitor detects a peak[51].

To detect peaks of A when its amount rises above 1100 and drops below 1000 within 5 time units, we use the formula (in the tool syntax):

```
true U[<=100] (A>1100 && true U[<=5] A<=1000)
```

Then the distance between peaks can be estimated by measuring maximal value of clock x with similar queries we used to estimate amplitude. The result is rendered by the tool as a logarithm of probability density shown in the second row of Figure E.13. The plots show that in most cases the measured distance between peaks is about 24.2 hours (slightly more than one day-night cycle). Then there are some smaller bumps with several magnitudes lower probability which can be explained by either a) false positive peak as the WMTL monitor is confused by a sudden stochastic saw-tooth in signal A, or b) missing a peak or two, or even three (in C) if the peak is not high enough to be registered at all, hence the next one is registered instead.

6 Conclusions

The present paper proves three main points:

1. Uppaal SMC can handle a wide range of biological systems.
2. The simulation engine of Uppaal SMC is of comparable performance with Simulink® and SimBiology®.
3. The essential benefit of Uppaal SMC comes with the power of statistical model checking with respect to a range of temporal logic properties.

In order to show the range of applicability we have connected the Uppaal SMC toolset to ANIMO and SimBiology®– two tools that can be used to specify and analyse biological phenomena. In particular we support SBML, used in many other research projects. Our implementation works via a translation from the input languages of those tools to the one of Uppaal SMC. This approach allows us not only to exploit the efficient ODEs solver of Uppaal SMC, but also to apply powerful techniques such as Statistical Model Checking to the systems.

As future work, we would like to implement new functionalities to capture a wider range of biological phenomena. We also plan to improve the ODE solving capabilities of Uppaal SMC by implementing more advanced ODE solvers such as `CVODE` and `ode15s`.

Checking & Distributing Statistical Model Checking

Abstract *In this paper we propose a general framework for distributed statistical model checking of networks of priced timed automata. The first contribution is a new algorithm to distribute sequential hypothesis testing without introducing bias in the results. The second contribution is an implementation of this algorithm in* Uppaal. *The major contribution is an experimental and analytical evaluation of the approach through case studies, including an analysis of the SMC algorithm itself.*

1 Introduction

Statistical model checking (SMC) techniques [75, 108, 123], can be seen as a trade-off between testing and formal verification. The core idea of the approach is to conduct some simulations of the system and verify if they satisfy a given property. The results are then used together with statistical algorithms to decide whether the system satisfies the property with some probability. Statistical model checking techniques can also be used to estimate the probability that a system satisfies a given property [75, 65]. Of course, in contrast to an exhaustive approach, a simulation-based solution does not guarantee a correct result with 100% confidence. However, it is possible to bound the probability of making an error. Statistical model checking gets widely accepted in various research areas and applied to problems that are beyond the scope of classical formal techniques [24, 41, 87, 102, 107, 122, 124].

Unfortunately, extremely huge sized problems and a demand of extremely high confidence may require generation of a large number of simulation runs, each of which may itself be extremely time consuming. To address this *confidence-explosion* problem, we suggest in this paper to take advantage of PC-clusters and GRID computers. In fact, it is well-known that statistical solutions methods that use samples of independent observations are often trivially parallelisable. As observed in [120], SMC algorithms can be distributed through the help of a master/slave architecture where multiple computers are used to generate

the simulations. The idea is as follows: one or more slave processes register their ability to generate simulation with a single master process that is used to collect those simulations and perform the statistical test. However, this process may become complex when considering sequential hypothesis testing (when the number of simulations is not known in advance). The problem is that there might exist a correlation between a time needed to generate a random simulation and the fact that a property is satisfied by this simulation. Thus it is important to guarantee that the technique will not introduce a bias towards the results that are generated by shorter simulations.

In a series of recent works [45], we have extended UPPAAL with SMC algorithms applied to Networks of Priced Timed Automata – hence leading to the first implementation of SMC for real-timed stochastic systems. The objective of this paper is to go one step further and propose the first complete study of distributed SMC, in general, and in the framework of UPPAAL in particular. Our contributions are:

1. A *distributed implementation* of the estimation algorithm proposed in [75]. Building on classical Monte Carlo techniques [66], an estimation algorithm precomputes the number of simulations needed to estimate the probability to satisfy a property with a given confidence. Such an algorithm which is trivially parallelisable amounts to equally distribute the number of simulations to perform between the slave computers.

2. A *thorough evaluation* of our implementation through new applications of SMC algorithms. In particular, we apply the distributed SMC engine to an analysis of an instance of the LMAC protocol of unprecedented size. Additionally, a thorough evaluation of the distributed SMC framework itself is made aiming at identifying optimal settings of the parameters for the framework. The evaluation is carried out both experimentally (using the implementation) as well as analytically (using SMC) based on a model of the distributed SMC algorithm itself, and with high consistency in identifications made by the two approaches.

2 Statistical Model-Checking in UPPAAL

This section introduces the formalisms used in UPPAAL for modelling systems and specifying properties. Then, we briefly survey existing statistical model checking (SMC) algorithms. Finally, a novel application of SMC is presented.

2.1 Networks of Priced Timed Automata

The new SMC engine of UPPAAL [46] supports the analysis of priced timed automata (PTAs) that are timed automata whose clocks can evolve with different rates, and with no restrictions in guards and invariants. Additionally, we support other features of the UPPAAL model checker's input language such as integer variables, data structures and user-defined functions.

We also assume PTAs are input-enabled, deterministic (with a probability measure defined on the sets of successors), and non-zeno. PTAs communicate via broadcast channels and shared variables to generate NPTAs.

Figure F.1 provides an NPTA with three components A, B, and T as specified using the UPPAAL GUI. One can easily see that the composite system $(A|B|T)$ has the transition sequence:

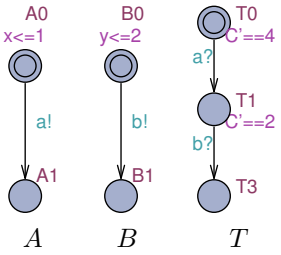


Figure F.1: An Network of priced timed automata (NPTA), $(A|B|T)$.

$$\big((A_0, B_o, T_0), [x = 0, y = 0, C = 0]\big) \xrightarrow{1} \xrightarrow{a!} \\ \big((A_1, B_0, T_1), [x = 1, y = 1, C = 4]\big) \xrightarrow{1} \xrightarrow{b!} \\ \big((A_1, B_1, T_2), [x = 2, y = 2, C = 6]\big),$$

demonstrating that the final location T_3 of T is reachable. In fact, location T3 is reachable within cost 0 to 6 and within total time 0 and 2 in (A|B|T) depending on when (and in which order) A and B choose to perform the output actions a! and b!. Assuming that the choice of these time-delays is governed by probability distributions, a measure on sets of runs of NPTAs is induced, according to which quantitative properties such as *"the probability of* T3 *being reached within a total cost-bound of 4.3"* become well-defined.

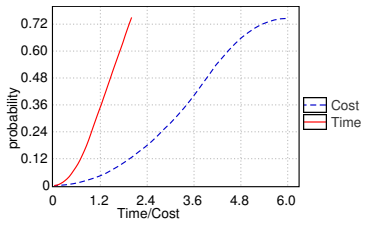


Figure F.2: Cumulative probabilities for `time` and `Cost`-bounded reachability of T_3.

In our early works [45], the stochastic semantic of PTA components associates probability distributions on both the delays one can spend in a given state as well as on the transition between states. In UPPAAL uniform distributions are applied for bounded delays and exponential distributions for the case where a component can remain indefinitely in a state. In a NPTA the components repeatedly race against each other, i.e. they independently and stochastically decide on their own how much to delay before outputting, with the "winner" being the component that chooses the minimum delay. For instance, in the NPTA of Figure F.1, A wins the initial race over B with probability 0.75.

Properties For specifying properties of NPTAs, we use cost-constrained temporal properties over runs of the form $\psi = \Diamond^{\mathtt{x}}_{\leq c}\phi$. Here $\mathtt{x}$ is an observer clock (that is never reset and should grow to infinity on any infinite run), $c \in \mathbb{R}_{\geq 0}$ and ϕ is a state-predicate. We say that a run π satisfies $\psi = \Diamond^{\mathtt{x}}_{\leq c}\phi$ if there exists a state $(\mathtt{l}, \mathtt{v})$ in π satisfying ϕ and with $\mathtt{v}(\mathtt{x}) \leq c$. For an NPTA $\mathcal{N}$ we define $\mathbb{P}_{\mathcal{N}}(\psi)$ to be the probability that a random run of $\mathcal{N}$ satisfies ψ.

Reconsider the example of Figure F.1, we can evaluate the probabilities `Pr[time<=2](◇ T.T3)` and `Pr[C<=6](◇ T.T3)` in UPPAAL, obtaining as expected 0.75 for the composition (A|B|T) for both of these probabilities. In fact Figure F.2 gives the time- and cost-bounded reachability probabilities for (A|B|T) for a range of bounds.

2.2 Statistical Model Checking Algorithms

We briefly recall statistical algorithms allowing to answer the following two types of questions : (1) *qualitative* Is the probability that a random run of a given NPTA $\mathcal{N}$ satisfies a property $\Diamond^{\mathtt{x}}_{\leq c}\phi$ greater than a certain threshold θ? and (2) *quantitative:* What is the probability that a random run of $\mathcal{N}$ satisfies $\Diamond^{\mathtt{x}}_{\leq c}\phi$? For both question a run of the system is encoded as a Bernoulli random variable that is true if the run satisfies the property and false otherwise.

Qualitative Question This reduces to test the hypothesis H: $p = \mathbb{P}_{\mathcal{N}}(\Diamond^{\mathtt{x}}_{\leq c}\phi) \geq \theta$ against $K : p < \theta$. To bound the probability of making errors, we use strength parameters α and β and test the hypothesis $\mathrm{H}_0 : p \geq p_0$ and $\mathrm{H}_1 : p \leq p_1$ with $p_0 = \theta + \delta_0$ and $p_1 = \theta - \delta_1$ (δ_0 and δ_1 are parameters of the algorithm). The interval $[p_0, p_1]$ defines an indifference region, and p_0 and p_1 are used as thresholds in the algorithm. The parameter α is the probability of accepting H_0 when H_1 holds and the parameter β is the probability of accepting H_1 when H_0 holds. The above test can be solved by using Wald's sequential hypothesis testing [119]. This test, which is presented in Algorithm F.1, computes a proportion r among those runs that satisfy the property. With probability 1, the value of the proportion will eventually cross $\log(\beta/(1-\alpha)$ or $\log((1-\beta)/\alpha)$ and one of the two hypotheses will be selected.

Algorithm F.1: Hypothesis testing

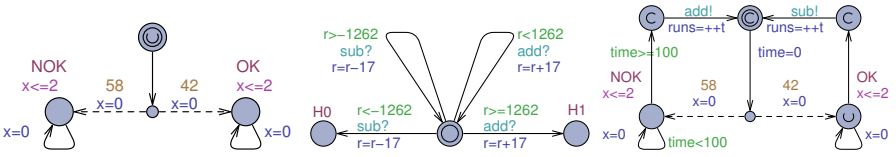
```
1  function hypothesis(S:model , ψ: property)
3  r:=0
5  while true do
7  |  Observe the random variable x corresponding to ◇_{x≤c}φ for a run.
9  |  r := r + x * log(p1/p0) + (1 − x) * log((1 − p1)/(1 − p0))
11 |  if r ≤ log(β/(1 − α)) then accept H0
13 |  if r ≥ log((1 − β)/α) then accept H1
14 end
```

Quantitative Question This reduces to a Monte Carlo approach that computes the number N of runs needed in order to produce an approximation interval $[p - \epsilon, p + \epsilon]$ for $p = Pr(\psi)$ with a confidence $1 - \alpha$. The values of ϵ and α are chosen by the user and N relies on the Chernoff-Hoeffding bound.

2.3 Analysing SMC in UPPAAL

In this section we will use the SMC engine of UPPAAL to our first non-trivial task, namely to analyse itself! More precisely, by suitably modelling the sequential testing algorithm as well as a sample model M, we will be able to use the SMC engine of UPPAAL to analyse the performance of SMC on M. Later, in Section 4, this will allow us to evaluate various naive (and even faulty) proposals for distributing SMC.

The sample model M given in Figure F.3a[1] makes an initial probabilistic choice between the two branches, each having a looping transition taken repeatedly with a delay chosen uniformly from $]0, 2]$. Performing sequential testing of the hypothesis H_0: `Pr[<=100](◇ OK)≥ 0.5` some 10 times with $\alpha = 0.05$ as level of significance and with an indifference region of ± 0.01, we consistently (and correctly) dismiss the hypothesis with an average of 408.6 runs and with standard deviation 127.5.



(a) M. (b) Master. (c) Generator.

Figure F.3: Sample model M (a) satisfying `Pr[<=100](◇ OK)` = 0.42 and modelling SMC of M (b, c) with respect to H_0: `Pr[<=100](◇ OK)≥ 0.5` with 0.05 as level of significance and $[0.49, 0.51]$ as indifference region.

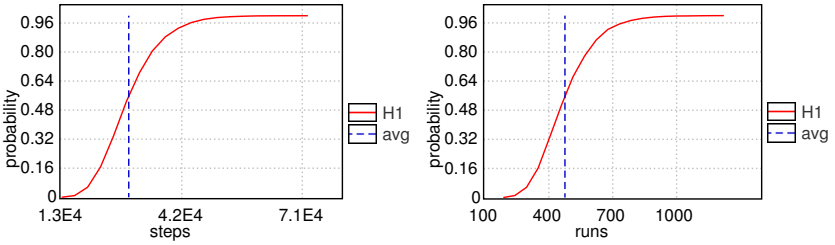
Now, aiming at obtaining a better understanding of sequential testing[2] we may simply model the sequential testing algorithm of M directly and analyse its (expected) performance using UPPAAL SMC. The resulting model is given in Figure F.3 and consists of an extension of the sample model M into the component Generator that will repeatedly generate random runs of M (of time-duration 100) and report the outcome to a Master using the channels add (when 100 time-units has elapsed without OK having been observed) and sub (used as soon as it is observed that the OK branch has been taken, note the absence of the `time>=100` guard on the right side of the Generator model). The Master has the obligation of adjusting appropriately the ratio-variable r according to Algorithm F.1, and conclude on H_0 or H_1 as soon as the value of r exceeds the given threshold. Given the indifference region $[0.49, 0.51]$ and level of significance 0.05, we find that the approximate values to be used[3] in Algorithm F.1 are: $-\log(p_1/p_0) =$

[1] M is a timed variant of the model proposed in [123] and used to demonstrate bias in a naive distributed approach to SMC.

[2] The performance of sequential testing has been subject to significant studies and is well-understood [118]. The aim here is to demonstrate that our UPPAAL SMC engine is a useful tool for obtaining such an insight.

[3] Those values are obtained by observing Wald's ratio on several application of the SMC algorithm to the same problem, and then take the average of the observations.

$\log(1-p_0/1-p_1) = 0.01715$ and $\log((1-\beta)/\alpha) = -\log(\beta/(1-\alpha)) = 1.2787$ ($\approx 1.262 + 0.017$). In the model of Figure F.3 we are using scaled integer constants for these values. Now, looking at the estimation of `Pr[#<=20000] (◇ Master.H1)` in Figure F.4, we find – as expected – that the probability of accepting H_1 (H_0) tends to 1 (0) as the number of steps increases. We also see that the average number of runs is estimated to 481.4. The "mismatch" with the experimentally found average 408.6 is due to early termination when the threshold for H_0 is exceeded.



(a) $\texttt{Pr}[\# \leq 80000](\Diamond\ \texttt{Master.H}_1)$ (b) $\texttt{Pr}[\texttt{runs} \leq 20000](\Diamond\ \texttt{Master.H}_1)$

Figure F.4: Cumulative probability plots over number of steps and runs.

3 Distributed Statistical Model-Checking in UPPAAL

SMC suffer from the fact that high confidence required by an answer may demand a large number of simulation runs, each of which may itself be time consuming. As an example, the first hypothesis test shown later in this section can generate between 14,000 and 2,390,000 runs if the parameters α, β, δ range between 0.01 and 0.001. A possible way to leverage this problem is to use several computers working in parallel using a master/slaves architecture: one or more slave processes register their ability to generate simulation with a single master process that is used to collect those simulations and perform the statistical test. For an estimation algorithm, this collection is trivially performed as the number of simulations to perform is known in advance and can be equally distributed between the slaves. When working with sequential algorithms, the situation gets more complicated. Indeed, we need to avoid introducing bias when collecting the results produced by the slave computers. This means that results should not be collected arbitrarily as illustrated by considering the model of Sec-

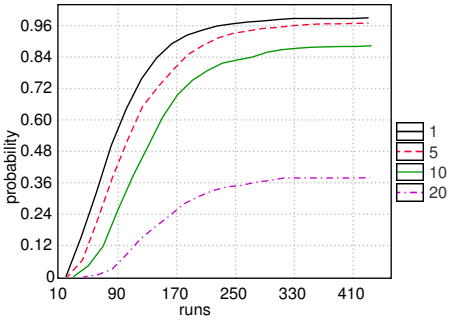


Figure F.5: Probability distributions obtained with 1 (top), 5, 10, and 20 (bottom) generator nodes.

tion 2.3 with several instances of the Generator template. Checking the property `Pr[runs<=20000](◇ Master.H1)` Figure F.5 shows that different distributions can be obtained with different numbers of generator nodes, hence revealing a bias in the results. In fact the probability of accepting H_1 tends (incorrectly) to 0 when the number of Generator components increases.

A solution, which was proposed in [123], consists in observing that Wald's ratio r is updated as a function of the Bernoulli random variable x as $r+ = x*r_{acc}+(1-x)*r_{rej}$ with r_{acc} and r_{rej} being constants depending on the tested hypothesis. To reduce blocking and still update r, the non-biased algorithm updates two safe approximations for r (r_1 and r_2). If x is unknown then it updates with $r_1+ = r_{rej}$ and $r_2+ = r_{acc}$, and then testing if $r_1 \leq I$ to accept H_0 or if $r_2 \geq S$ to accept H_1[4]. When all outcomes of a round are known then we can reset $r_1 := r_2 := r$. This allows us to accept H_0 even if some accepting outcomes are missing or conversely to accept H_1 if some rejecting outcomes are missing. We generalise [123] by aggregating the outcomes x by *batches* (of size B) and also by implementing a *buffer* (of size K) of incoming results. The batch is used to reduce communication by sending B aggregate results. The buffer is used to improve concurrency since the nodes are more loosely synchronised and they can be K runs ahead of the slowest node. Figure F.6 illustrates our algorithm at the master node that receives asynchronous messages from all other nodes in a buffer. A message is an aggregate result containing the outcome of B runs. The master may take a decision as soon as $r_1 \leq I$ or $r_2 \geq S$. When all outcomes at the bottom line of the buffer are known we reset $r_1 := r_2 := r$ with the exact updated value of r with those outcomes, and free the bottom line of the buffer. In practice, our algorithm is calibrated to count the runs up to a certain depth in the buffer. Indeed, the outcomes are weighted by B so few missing aggregated outcomes can prevent the algorithm from deciding. We have implemented this algorithm with asynchronous communications (using OpenMPI). There can be at most K pending messages due to the size of the buffer. If a slave tries to send more messages, then the communication will block waiting for a "slot" to be free. The experiment performed in the remainder of the paper has been carried out on varying numbers of nodes on a cluster with dual Xeons 5650 (hexa-cores at 2.66GHz) interconnected with infiniband.

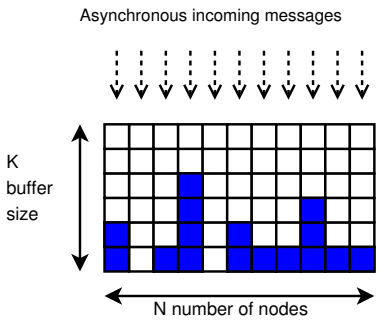


Figure F.6: Buffer of results at the master node.

We first make two types of experiments to exhibit the performance characteristics of our algorithm. The experiments are carried out using the train-gate example available as a demo of UPPAAL. This model comprises a number of

[4] $I = \log(\beta/(1-\alpha))$ and $S = \log((1-\beta)/\alpha)$ as stated in Alg. 1.

trains crossing a bridge with only one track. A gate controller stops and restart the trains to ensure mutual exclusion on the bridge and absence of starvation for the trains. Our first experiment concerns 6 trains and the property of being in a state where train 5 is crossing while all the other trains are stopped.

```
Pr[<=100](<> Train(5).Cross and
     (forall (i : id_t) i != 5 imply Train(i).Stop)) >= 0.46188
```

The runs are relatively short with few components so they will be cheap to compute and we expect the throughput of messages to be high. In addition, the hypothesis we are testing is not deterministic, which means that the outcomes and computation times of the runs will vary. The property is checked with high confidence (99.999%) and small indifference regions (+/- 0.00001) to have a precise and reliable result – and to stress our distributed algorithm.

Our second experiment considers a "large" instance with 20 trains, where we check if the model satisfies mutual exclusion on the bridge, expressed by the property

```
Pr[<=1000]([] forall (i : id_t) forall (j : id_t)
    Train(i).Cross and Train(j).Cross imply i == j) >= 0.9999
```

Here, the runs are random but bounded by the same large bound and since the inner property `[]forall(i : id_t)forall(j : id_t)...` holds by model-checking, all the runs will all reach their bounds. In addition, we have 20 trains and the runs are long (1000 time units) so they are relatively expensive to generate. This means that all the runs are implicitly synchronised and small deviations are amortised by the long runs. The throughput of messages will be low, which means a low overhead compared to the actual useful work of generating the runs.

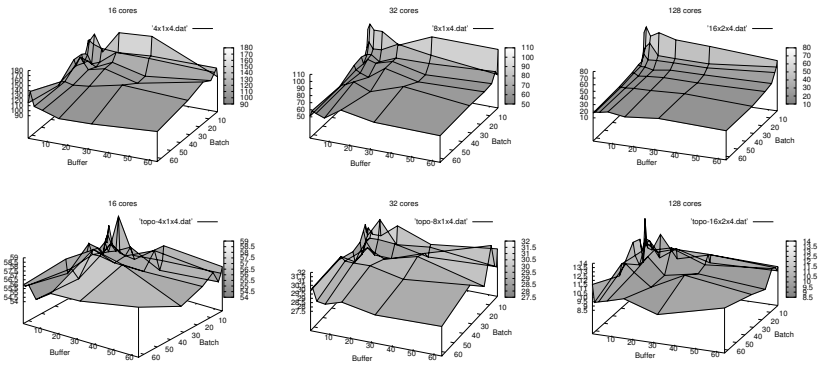


Figure F.7: Verification times on 16, 32, and 128 cores in function of B and K for the "small" model (first row) and the "large" model (second row).

Figure F.7 shows our results for different number of cores. The solution in [120] corresponds to the particular case with K and B are equal to one, exhibiting in all the experiments the worst verification time, and with performance

deteriorating with increasing number of cores (i.e. for 128 cores performance loss is a factor of 4). Though the impact of the buffer size is less, the experiments indicate that a buffer of size 2-4 will suffice. The results also demonstrate linear scalability of our distributed implementation: for $B = 32$ and $K = 2$ the verification times for 16, 32 and 128 cores are 108, 56 and 19 seconds (respectively).

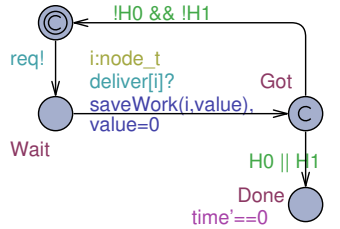
3.1 Distributing Estimation

Distributing the estimation algorithm is much simpler than distributing sequential hypothesis testing. We need a *fixed* number of runs to compute an estimate of the probability value with given confidence level. This is an embarrassingly parallel problem since we can simply divide the work equally and gather the result at the end. The speed of generating the runs has no influence here since every computer node is allocated the same number of runs and if this number is high enough (and the computers identical), the running time on every node will be closely the same. This latter claim is indeed confirmed by our experiments since we observe that the performance scales almost linearly with the number of nodes. Interestingly, the loss in efficiency in the later cases exhibits the overhead of starting up all the processors (around 3-4 seconds), which would be compensated for much longer runs. The jobs here are too small and fast, which is an extreme case. Table F.1 shows running time and relative efficiency for estimating a few probabilities on the Firewire and LMAC protocol[5] with confidence 99.9% and uncertainty interval 0.005. Time is not very sensitive to the physical placement of the cores here because there is essentially one communication round when all the nodes have finished their jobs.

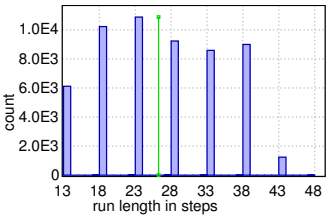
	Firewire					LMAC				
PxC/N	1	2	4	8	16	1	2	4	8	16
1x1	621.7s	316.7s	160.2s	81.1s	44.7s	279.3s	140.7s	73.0s	37.0s	19.5s
	1.00	*0.98*	*0.97*	*0.96*	*0.87*	*1.00*	*0.99*	*0.96*	*0.94*	*0.90*
1x2	300.9s	162.2s	80.5s	47.6s	24.3s	144.3s	71.0	37.5s	19.2s	10.4s
	1.03	*0.96*	*0.97*	*0.82*	*0.80*	*0.97*	*0.98*	*0.93*	*0.91*	*0.84*
1x4	161.2s	84.0s	44.8s	24.1s	16.0s	74.2s	36.1s	19.3s	9.6s	8.1s
	0.96	*0.93*	*0.87*	*0.81*	*0.61*	*0.94*	*0.97*	*0.90*	*0.91*	*0.54*
2x4	85.1s	46.5s	23.1s	14.1s	8.5	35.5s	19.6s	10.1s	10.2s	6.4s
	0.91	*0.84*	*0.84*	*0.69*	*0.57*	*0.98*	*0.89*	*0.86*	*0.43*	*0.34*

Table F.1: Time in seconds and efficiency (italic) to estimate probabilities on the Firewire and LMAC model in function of the number of nodes (N), processors per node (P) and cores per processor (C).

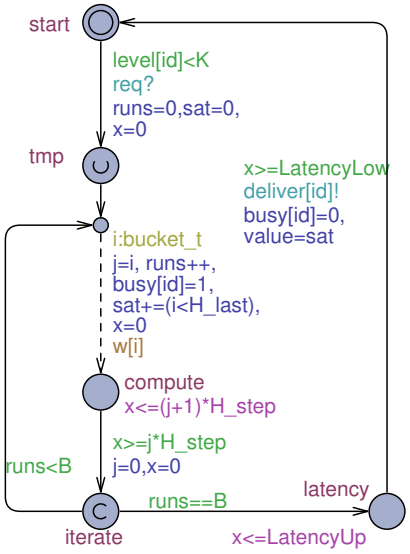
[5]The model and properties are available on `http://people.cs.aau.dk/~adavid/smc/`http://people.cs.aau.dk/~adavid/smc/.



(a) Master.



(b) Histogram of verification steps.



(c) Slave.

Figure F.8: Timed automata model of a statistical model checking process.

4 Analysing Distributed SMC in UPPAAL

In this section we model the implemented distributed algorithm of sequential hypothesis testing and we check it using the SMC engine of UPPAAL. The goal is to estimate the verification time and processor utilisation, check for bias in the distributed algorithm, and explore the parameters of our distributed SMC algorithm in an analytical manner.

Modelling.

We model the master and slave processes described in Section 3 as shown in Figure F.8. The master sends a broadcast request req! to verify batches of runs (of size B). We use a standard modelling pattern to synchronise on the corresponding req? as soon as possible. The master gathers the results with its saveWork function and loops again if neither H_0 nor H_1 is accepted. Listing F.3 shows this saveWork function that implements the distributed hypothesis testing algorithm of Section 3. UPPAAL uses floating point numbers that are not available in the modelling language. Instead we encode fixed point arithmetics with integers and we use precomputed tables for logarithm values (shown in Listing F.2).

Once the master accepts H_0 or H_1, it moves to the location Done and stops the clock `time`.

Slave processes proceed to compute their batches if their communication buffers are not full (level [id] < K) or wait for the condition to hold. The

```
typedef int[-1024*1024*1024, 1024*1024*1024] long; // high precision
const int N = 128; // number of processors
const int B = 32; // number of runs in a batch
const int K = 4; // maximum difference of batches among processors
const int LatencyLower = 19; // latency lower bound
const int LatencyUpper = 20; // latency higher bound
typedef int[0, N-1] node_t; // node id type
typedef struct { // SMC verification parameters:
    long scale; // denominator for fixed point operations
    long alpha; // probability of false positive (multiplied by
    scale)
    long beta;  // probability of false negative (multiplied by
    scale)
    long theta; // the probability for hypothesis testing
    long deltaM;// lower probability deviation
    long deltaP;// upper probability deviation
    long p1;    // lower probability bound [theta-deltaM]
    long p0;    // upper probability bound [theta+deltaP]
    long valAcc;// value accumulation [log(p1/p0)]
    long valRef;// value reference  [log((1-p1)/(1-p0))]
    long logInf;// log infimum (lower value bound) [log(beta/(1-
    alpha))]
    long logSup;// log supremum (upper value bound) [log((1-
    beta)/alpha))]
} SMC_params_t; // the structure is passed to Master upon
    instantiation
const int H_first = 13;// the start of the histogram
const int H_last = 48; // the end of the histogram
const int H_step = 1;  // time step of one bucket
typedef int[H_first, H_last] bucket_t; // type of integer with
    specific range
const int w[bucket_t] = { 6207,0,0,0,0,10463,0,0,0,5,10903,0,0,0,4,
     9133,0,0,0,3,8569,0,0,0,2,8837,0,0,0,1,1233,0,0,0,1,64469 };
broadcast chan req; // master requests
broadcast chan deliver[node_t]; // slave delivers
int level[node_t]; // level of the batch queue for each node
bool busy[node_t]; // encodes whether the node is computing
```

Listing F.1: DSMC model global declarations.

```
int value; // shared variable for transfering results
const SMC_params_t p[3] = {
//scale  alpha  beta theta +delta -delta   p1    p0    valAcc
    valRef logInf logSup
 {100000,  100,  100, 46188,  100,  100, 46088, 46288,  -433,  372,-
    690675,690675},
 {100000, 1000, 1000, 46188, 1000, 1000, 45188, 47188, -4331, 3717,-
    459512,459512},
 {100000, 5000, 5000, 46188, 5000, 5000, 41188, 51188,-21736,18637,-
    294444,294444}
};
master = Master(p[1], value);
slave(const node_t id) = Slave(id, value);
system master, slave, Global;
```

Listing F.2: DSMC instantiation and system declaration.

```
// buffer portion for early termination:
const int P = (K<=4)?K : ((K<=8)?5 : ((K<=16)?8 : ((K<=32)?10 :
    12)));
bool H0 = false, H1 = false; // for hypothesis H0 and H1
int batch[N][K]; // buffer of batches (K batches for N nodes)
long satisfied=0, unsatisfied=0; // information about filled lines
long sat=0, unsat=0, unknown=N*P*B; // early results in unfilled
    lines
long logRatio = 0, ratioLow = 0, ratioUp = 0; // scaled by p.scale
void saveWork(const node_t node, const int value) {
    if (level[node]<=P) { // entered the early results portion
        sat += value; unsat += B-value; unknown -= B;
    }
    batch[node][level[node]] = value; level[node]++; // store
    if (level[node]==1) { // entered at the lowest level
        bool filled = forall(i:node_t) level[i]>0;
        if (filled) { // line at the lowest level has been filled
            int L;
            for (i: node_t) { // shift all queues one by one
                satisfied += batch[i][0]; // count as firm results
                unsatisfied += B-batch[i][0];
                sat -= batch[i][0]; // discount from early results
                unsat -= B-batch[i][0]; unknown += B;
                level[i]--; // remove from buffer
                for (L=0; L<level[i]; ++L) {
                    batch[i][L] = batch[i][L+1]; // shift
                    if (L==P) { // entered the early results
    portion
                        sat += batch[i][L+1]; unsat +=
    B-batch[i][L+1];
                    }
                }
                batch[i][level[i]]=0; // cleanup
            }
            logRatio = p.valAcc*satisfied + unsatisfied*p.valRef;
            if (logRatio <= p.logInf) H0 = true;
            if (logRatio >= p.logSup) H1 = true;
        }
    }
    ratioLow = p.valAcc*(satisfied   +sat+unknown) +
               p.valRef*(unsatisfied +unsat);
    ratioUp  = p.valAcc*(satisfied   +sat) +
               p.valRef*(unsatisfied +unsat+unknown);
    if (ratioUp  <= p.logInf) H0 = true;
    if (ratioLow >= p.logSup) H1 = true;
}
```

Listing F.3: Master code.

compute location models the computation time of a run, chosen according to the distribution shown in Figure F.8b. This is encoded using probabilistic edges with weights matching the distribution. The distribution comes from a real verification of the property in Section 3:

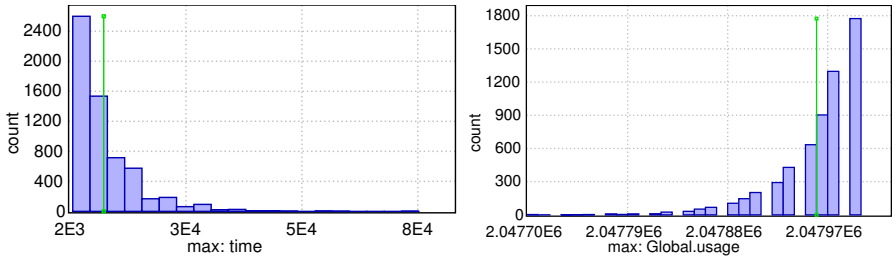
```
Pr[<=100](<> Train(5).Cross and
        (forall (i : id_t) i != 5 imply Train(i).Stop)) >= 0.46188
```

The last weighted edge (case `i=H`) is reserved for the runs that did not satisfy the property.

Verification.

In the hypothesis we test, the actual probability is very close to 0.46188. Since the real probability falls in the indifference region of our test, we would expect that a non-biased implementation would accept H_0 or H_1 equally often. Estimating the probability of confirming the hypothesis H_0 with the query `Pr[<=10000000](<> master.H0)` gives the probability 0.503±0.005 with 99.9% confidence, confirming that our algorithm is not biased as well as the validity of our model.

Similarly, we obtain the distribution of the verification time by the query `Pr[<=10000000](<> master.Done)` for a model with number of nodes $N = 128$, batch size $B = 64$, and buffer size $K = 4$. The result is 9557.6 time units in average and the distribution histogram is depicted in Figure F.9a. To estimate the processor usage time, we add another process with a single location with the invariant usage'==sum(i:node_t)busy[i]. Here, usage is a clock that grows with a rate equal to the number of busy nodes.



(a) Estimated time distribution.

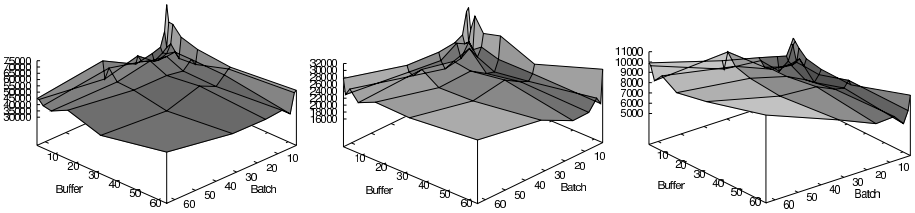
(b) Processor usage time distribution.

Figure F.9: Time estimation from 6000 runs of DSMC model.

The question is now to find a good settings for the parameters of our algorithm (B and K). We perform parameter sweep to estimate the verification time for values of B and K taking values in $1, 2, 4, 8, 16, 32, 64$ for three topologies with the number of processing nodes $N = 16, 32$, or 128. The results are depicted in Figure F.10, where it is visible that extremely small batch size requires more time. Large batch sizes can also be detrimental in a large cluster setting (Figure F.10c where too many runs are requested in bulk than actually needed to establish the result). Buffer size of one has a huge penalty of blocking with small blocks, but it is barely noticeable otherwise. This confirms the experimental findings of Section 3.

5 Lightweight Media Access Control

LMAC is a Lightweight Media Access protocol (studied in [45, 57]) used for scheduling communication in wireless sensor networks where the topology is determined by physical location and radio connectivity of the individual nodes.



(a) 16 nodes. (b) 32 nodes. (c) 128 nodes.

Figure F.10: Estimated verification times in model time units.

One of the goals of the LMAC protocol is to minimize the number of collisions in the network and to reconfigure the network to avoid further collisions.

The original model has been developed in [57] where topologies of 4-5 nodes are studied exhaustively using classical UPPAAL and a number of topologies are identified as problematic, containing perpetual collisions. In this paper we provide new insight as to the likelihood of perpetual collisions in different topologies. This insight could not be delivered by the use of classical UPPAAL and the experiment conducted is of unprecedented size. In LMAC communication media access time is discretized into time frames and each time frame is divided into time slots. The goal of the protocol is to allocate the time slots to each node efficiently. The challenge is that there is no central node distributing and assigning slots and nodes cannot themselves listen while transmitting, hence neighbours are responsible for detecting and informing each other about collisions. After waiting phase, the node moves to a discovery phase and listens for an entire time frame and notes which time slots are used by its neighbours. The collision counting expression collisions =++cc; is added on the edge from rec_one0 to done0 in Figure F.12b. After one time frame of discovery phase, the node chooses seemingly unused time slot and moves to an active phase. The node falls back to waiting phase if there are no neighbours (no signal received) or all slots are occupied. During active and discovery phases the node listens and notes any collisions (several receptions during the same slot). During active phase the node transmits information about collisions it has detected during its time slot and listens for collisions and information about collisions during other time slots. From the active phase the node may fall back to discovery phase if it is notified about the collisions on its time slot and falls back to the waiting phase if it detects that neighbours are gone.

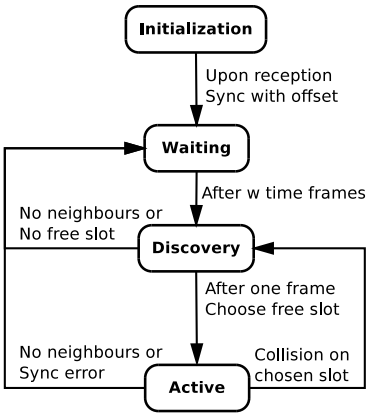
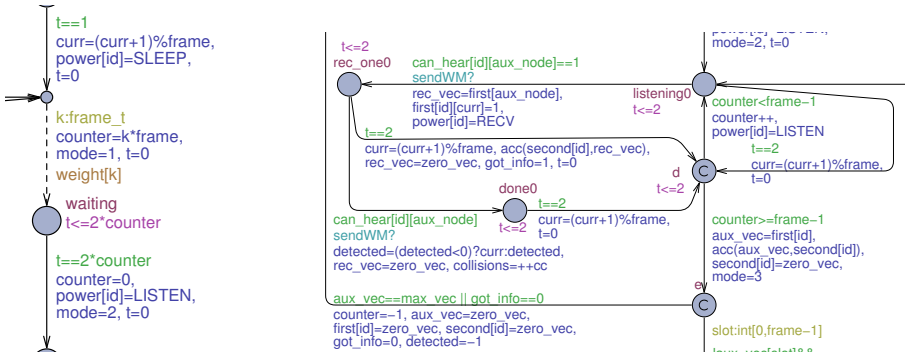


Figure F.11: LMAC protocol phases.

Figure F.11 shows the four phases of the protocol. Initially all nodes except the gateway are listening and waiting for a radio signal from its neighbourhood

during the initialization phase. The communication is triggered by a dedicated gateway node. Upon reception of signal, the node notes the relative time offset of the signal and moves to waiting phase, during which it chooses to wait for a random amount of time frames. The random delay is modeled using probabilistic branching (see Figure F.12a) with geometrical weights (`weight` array).



(a) Waiting. (b) Discovery.

Figure F.12: LMAC phases in the model.

Starting from the model[6] of [57], we removed the verification optimizations constraining the parallelism, annotated it with power consumption and collision counting (as cost variables). The model contains twice as many slots as nodes, whereas one slot per node is enough to schedule flawless communication in any topology if nodes were aware of each others choices.

First we examine the distribution of the first collisions over time. The first row of Figure F.13 is a result of a query `Pr[<=1000](◇ collision>0)` and it shows that most collisions happen early in time and in a ring topology some collisions may be discovered later (possibly when the first signal propagation meets at the opposite of the ring). In the second row of Figure F.13 the distribution of possible number of collisions is examined using a query `Pr[collisions<=100](◇ time>=1000)`: in a chain and a ring topologies the collisions are unlikely to occur ($> 90\%$ probability of 0 collisions), but in a star it is almost guaranteed to occur (only 8% probability of 0 collisions). The third row of Figure F.13 shows the probability distribution of collision counts after twice as long period of time (using query `Pr[collisions<=100](◇ time>=2000)`). Notice that the shape of distributions has not changed, but the small bumps have shifted to the right at exactly twice the number of collisions and almost identical probability density, which implies that those particular collisions are accumulating proportionally to the progress of time, and in other words it means that collisions are reoccurring perpetually without recovery. We checked these

[6]Thanks to Ansgar Fehnker and Angelika Mader.

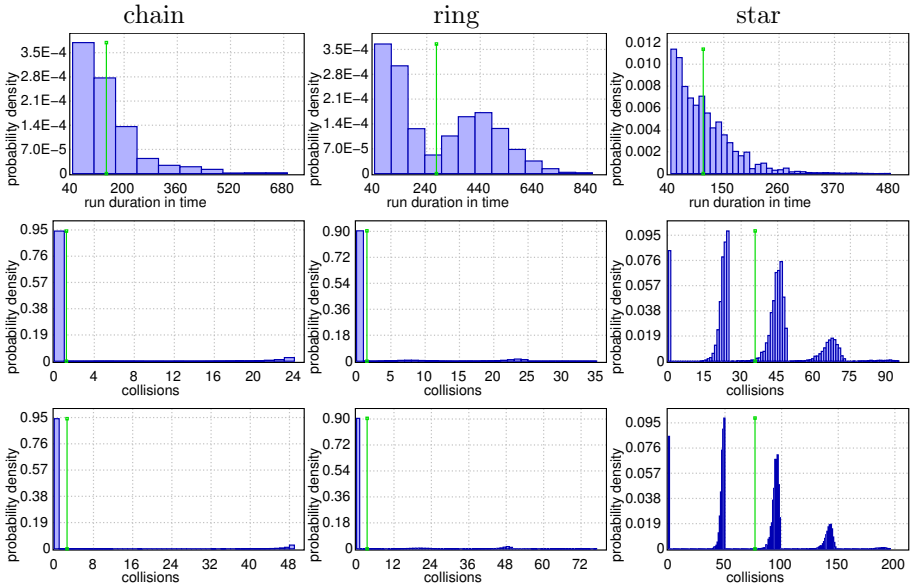


Figure F.13: Collision statistics in three different topologies, in rows: probability of a collision over time, probabilities of a number of collisions up to 1000 and up to 2000 time units.

three properties on a 128 cores cluster with high precision (with $\alpha = \beta = 0.0001$ and $\varepsilon = 0.0005$) in about 30 minutes, which generated around 19 million runs.

We have demonstrated how UPPAAL SMC can be used to identify problematic topologies and distributed implementation can provide a high degree of accuracy in spotting the reoccurring collisions.

6 Comparison with other toolsets

To the best of our knowledge, there are only two tools that implement distributed SMC algorithms, namely YMER [120] and PVESTA [4]. We could not conduct a direct comparison with distributed YMER as the implementation is currently broken[7]. However, we could compare the batch algorithm implemented in YMER with ours in the setting of UPPAAL (see Section 3).

PVESTA is a distributed statistical model checker based on VESTA [109]. It performs hypothesis testing but instead of using a sequential approach, it relies on a single sampling plan where the number, N, of samples needed to solve the qualitative question is pre-computed. As for the estimation algorithm a single sampling plan is thus trivially parallelisable. To compare PVESTA with our implementation we created a UPPAAL model of the cyclic polling server example of the PVESTA distribution.

[7]Discovered per personal communication with H. Youness

	Time (900 st.)			Samples (900 st.)			Time (9000 st.)	
Nodes	UPPAAL$_h$	PVESTA	UPPAAL$_e$	UPPAAL$_h$	PVESTA	UPPAAL$_e$	UPPAAL$_h$	UPPAAL$_e$
1	≤ 4	46.0	12.15	115	16906	18448	≤ 4	84.0
2	≤ 4	23.7	7.5	190	16906	18448	≤ 4	44.4
4	≤ 4	12.7	3.91	557	16906	18448	≤ 4	23.8
8	≤ 4	7.2	5.5	2340	16906	18448	≤ 4	12.5

Table F.2: Verification time for UPPAAL and PVESTA. The nodes column refers only to the nodes used for sample generation (PVESTA actually used $nodes+1$ processing units for the verification). The UPPAAL$_h$ column shows the verification time in seconds for hypothesis testing and UPPAAL$_e$ for estimation using UPPAAL.

We test for the hypothesis provided with the example, namely `P >= 0.5 [<> < 20.0 @0]` and its equivalent in UPPAAL and we estimate this probability in UPPAAL. The results for a configuration of 900 stations are given in Table F.2. We note that for 900, the computation time of UPPAAL is so short that we can only measure the overhead of starting and distributing the computation that takes less than 4 seconds, which is why we show ≤ 4 in the table. To see the scalability of UPPAAL, we experiment with 9000 stations in the model. We can only show the results for UPPAAL since PVESTA reached its time limit of one hour before it could give a result. The experiments show that UPPAAL and PVESTA scale almost linearly and UPPAAL is at least two orders of magnitude faster than PVESTA on for hypothesis testing.

7 Conclusion and Future work

In this paper we have developed, implemented, applied and evaluated a general and scalable framework for distributed SMC. We have thoroughly investigated the distribution of sequential algorithms where bias can be introduced when collecting the samples produced by slave computers. In particular, we have identified best choices of batch and buffer sizes both experimentally and analytically, with agreement in the findings of the two approaches. In the future, we plan to implement and distribute other SMC algorithms, principally the Bayesian algorithms introduced in [124, 81].

Finally, it is worth mentioning that we have tried to use other distributed SMC model checkers such as YMER [120] or PVESTA [4, 109]. Aside from the fact that the GUI of those two tools is quite restricted, we observed that YMER does not work anymore and that PVesta only distributes those algorithms where the number of simulations is precomputed in advance.

Bibliography

[1] P. A. Abdulla, B. Jonsson, M. Nilsson, J. d'Orso, and M. Saksena. Regular Model Checking for LTL(MSO). In *CAV*, volume 3114 of *LNCS*, pages 348–360. Springer, 2004.

[2] Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, New York, NY, USA, 2007. ISBN 0521875463.

[3] Gul Agha, José Meseguer, and Koushik Sen. PMaude: Rewrite-based Specification Language for Probabilistic Object Systems. *Electronic Notes in Theoretical Computer Science*, 153(2):213–239, 2006.

[4] Musab AlTurki and José Meseguer. PVeStA: A Parallel Statistical Model Checking and Quantitative Analysis Tool. In Andrea Corradini, Bartek Klin, and Corina Cîrstea, editors, *CALCO*, volume 6859 of *Lecture Notes in Computer Science*, pages 386–392. Springer, 2011. ISBN 978-3-642-22943-5.

[5] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[6] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In Mike Paterson, editor, *ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990. ISBN 3-540-52826-1.

[7] Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[8] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-Checking in Dense Real-time. *Inf. Comput.*, 104(1):2–34, 1993.

[9] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43:116–146, January 1996.

[10] Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal Paths in Weighted Timed Automata. In *HSCC'01*, pages 49–62. Springer, 2001.

[11] R.B. Ash and C. Doléans-Dade. *Probability and Measure Theory*. Harcourt/Academic Press, 2000. ISBN 9780120652020.

[12] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008. ISBN 978-0-262-02649-9.

[13] Paolo Ballarini, Hilal Djafri, Marie Duflot, Serge Haddad, and Nihal Pekergin. COSMOS: A Statistical Model Checker for the Hybrid Automata Stochastic Logic. In *Eighth International Conference on Quantitative Evaluation of Systems, QEST 2011, Aachen, Germany, 5-8 September, 2011*, pages 143–144. IEEE Computer Society, 2011. ISBN 978-1-4577-0973-9. doi:10.1109/QEST.2011.24.

[14] Benoît Barbot, Taolue Chen, Tingting Han, Joost-Pieter Katoen, and Alexandru Mereacre. Efficient CTMC Model Checking of Linear Real-Time Objectives. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *TACAS*, volume 6605 of *Lecture Notes in Computer Science*, pages 128–142. Springer, 2011. ISBN 978-3-642-19834-2.

[15] Naama Barkai and Stanislas Leibler. Biological rhythms: Circadian clocks limited by noise. *Nature*, 403:267–268, 2000.

[16] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Delahaye, and Axel Legay. Statistical Abstraction and Model-Checking of Large Heterogeneous Systems. In John Hatcliff and Elena Zucca, editors, *Formal Techniques for Distributed Systems*, volume 6117 of *Lecture Notes in Computer Science*, pages 32–46. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-13463-0. doi:10.1007/978-3-642-13464-7_4.

[17] Andreas Bauer and Patrik Haslum. LTL Goal Specifications Revisited. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 881–886. IOS Press, 2010. ISBN 978-1-60750-605-8.

[18] Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing LTL Semantics for Runtime Verification. *J. Log. Comput.*, 20(3):651–674, 2010.

[19] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime Verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, September 2011. ISSN 1049-331X. doi:10.1145/2000799.2000800.

[20] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn, and Frits W. Vaandrager. Minimum-Cost Reachability for Priced Timed Automata. In Maria Domenica Di Benedetto and Alberto L. Sangiovanni-Vincentelli, editors, *HSCC*, volume 2034 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2001. ISBN 3-540-41866-0.

[21] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. Developing UPPAAL over 15 years. *Softw., Pract. Exper.*, 41(2):133–142, 2011. doi:10.1002/spe.1006.

[22] S. Bensalem, B. Delahaye, and A. Legay. Statistical Model Checking: Present and Future. In *RV*. Springer-Verlag, 2010.

[23] Frank T. Bergmann and Herbert M. Sauro. Comparing simulation results of SBML capable simulators. *Bioinformatics*, 24(17):1963–1965, 2008. doi:10.1093/bioinformatics/btn319.

[24] Jonathan Bogdoll, Luis María Ferrer Fioriti, Arnd Hartmanns, and Holger Hermanns. Partial Order Methods for Statistical Model Checking and Simulation. In Roberto Bruni and Jürgen Dingel, editors, *Formal Techniques for Distributed Systems - Joint 13th IFIP WG 6.1 International Conference, FMOODS 2011, and 31st IFIP WG 6.1 International Conference, FORTE 2011, Reykjavik, Iceland, June 6-9, 2011. Proceedings*, volume 6722 of *Lecture Notes in Computer Science*, pages 59–74. Springer, 2011. ISBN 978-3-642-21460-8. doi:10.1007/978-3-642-21461-5_4.

[25] A. Bouajjani, A. Legay, and P. Wolper. Handling Liveness Properties in (Omega-)Regular Model Checking. In *INFINITY*, volume 138(3) of *ENTCS*. Elsevier, 2005.

[26] Abdeldjalil Boudjadar, Frits W. Vaandrager, Jean-Paul Bodeveix, and Mamoun Filali. Extending UPPAAL for the Modeling and Verification of Dynamic Real-Time Systems. In Farhad Arbab and Marjan Sirjani, editors, *FSEN*, volume 8161 of *Lecture Notes in Computer Science*, pages 111–132. Springer, 2013. ISBN 978-3-642-40212-8.

[27] Patricia Bouyer and Nicolas Markey. Costs Are Expensive! In *FORMATS*, volume 4763 of *LNCS*, pages 53–68. Springer, 2007.

[28] Patricia Bouyer, Kim Guldstrand Larsen, and Nicolas Markey. Model Checking One-Clock Priced Timed Automata. *Logical Methods in Computer Science*, 4(2):28, 2008.

[29] Patricia Bouyer, Nicolas Markey, Joël Ouaknine, and James Worrell. On Expressiveness and Complexity in Real-Time Model Checking. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 124–135. Springer, 2008. ISBN 978-3-540-70582-6.

[30] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A Model-Checking Tool for Real-Time Systems. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver,*

BC, Canada, June 28 - July 2, 1998, Proceedings, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer, 1998. ISBN 3-540-64608-6. doi:10.1007/BFb0028779.

[31] Marius Bozga, Mohamad Jaber, Nikolaos Maris, and Joseph Sifakis. Modeling Dynamic Architectures Using Dy-BIP. In *SC*, volume 7306 of *LNCS*, pages 1–16, 2012.

[32] Peter Bulychev, Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Danny Bøgsted Poulsen. Checking & Distributing Statistical Model Checking. In Alwyn E. Goodloe and Suzette Person, editors, *4th NASA FORMAL METHODS SYMPOSIUM*, volume 7226 of *LNCS*, pages 449–463. Springer, 2012. Paper F in this thesis.

[33] Peter Bulychev, Alexandre David, Kim G. Larsen, Axel Legay, and Marius Mikučionis. Computing Nash Equilibrium in Wireless Ad Hoc Networks: A Simulation-Based Approach. In Johannes Reich and Bernd Finkbeiner, editors, *Second International Workshop on Interactions, Games and Protocols*, volume 78 of *EPTCS*, pages 1–14, 2012. doi:10.4204/EPTCS.78.

[34] Peter E. Bulychev, Alexandre David, Kim G. Larsen, Axel Legay, Guangyuan Li, and Danny Bøgsted Poulsen. Rewrite-Based Statistical Model Checking of WMTL. In *RV*, volume 7687 of *LNCS*, pages 260–275, 2012. Paper B in this thesis.

[35] Peter E. Bulychev, Alexandre David, Kim Guldstrand Larsen, Axel Legay, Guangyuan Li, Danny Bøgsted Poulsen, and Amélie Stainer. Monitor-Based Statistical Model Checking for Weighted Metric Temporal Logic. In *LPAR*, volume 7180 of *LNCS*, pages 168–182, 2012. Paper A in this thesis.

[36] Franck Cassez and Kim Guldstrand Larsen. The Impressive Power of Stopwatches. In Catuscia Palamidessi, editor, *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2000. ISBN 3-540-67897-2.

[37] Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Di Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee, editors, *Software Engineering for Self-Adaptive Systems [outcome of a Dagstuhl Seminar]*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26.

Springer, 2009. ISBN 978-3-642-02160-2. doi:10.1007/978-3-642-02161-9_1.

[38] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer, 2002. ISBN 3-540-43997-8. doi:10.1007/3-540-45657-0_29.

[39] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.

[40] Edmund M. Clarke and E. Allen Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In Dexter Kozen, editor, *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981. ISBN 3-540-11212-X.

[41] Edmund M. Clarke, James R. Faeder, Christopher James Langmead, Leonard A. Harris, Sumit Kumar Jha, and Axel Legay. Statistical Model Checking in BioLab: Applications to the Automated Analysis of T-Cell Receptor Signaling Pathway. In *CMSB*, LNCS, pages 231–250, 2008.

[42] Costas Courcoubetis and Mihalis Yannakakis. Verifying Temporal Properties of Finite-State Probabilistic Programs. In *FOCS*, pages 338–345. IEEE Computer Society, 1988.

[43] Jean-Michel Couvreur. On-the-Fly Verification of Linear Temporal Logic. In *FM '99*, pages 253–271, 1999.

[44] Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Timed I/O automata: a complete specification theory for real-time systems. In *HSCC*, pages 91–100. ACM, 2010.

[45] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, Danny Bøgsted Poulsen, Jonas van Vliet, and Zheng Wang. Statistical Model Checking for Networks of Priced Timed Automata. In *FORMATS*, volume 6919 of *LNCS*, pages 80–96, 2011.

[46] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Zheng Wang. Time for Statistical Model Checking of Real-Time Systems. In *CAV*, volume 6806 of *LNCS*, pages 349–355. Springer, 2011.

[47] Alexandre David, Dehui Du, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical Model Checking for Stochastic Hybrid Systems. In Ezio Bartocci and Luca

Bortolussi, editors, *HSB*, volume 92 of *EPTCS*, pages 122–136, 2012. doi:10.4204/EPTCS.92.9.

[48] Alexandre David, DeHui Du, Kim G. Larsen, Marius Mikučionis, and Arne Skou. An evaluation framework for energy aware buildings using statistical model checking. *Science China Information Sciences*, 55:2694–2707, 2012. ISSN 1674-733X. doi:10.1007/s11432-012-4742-0.

[49] Alexandre David, Lasse Jacobsen, Morten Jacobsen, Kenneth Yrke Jørgensen, Mikael H. Møller, and Jirí Srba. TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets. In Cormac Flanagan and Barbara König, editors, *TACAS*, volume 7214 of *Lecture Notes in Computer Science*, pages 492–497. Springer, 2012. ISBN 978-3-642-28755-8.

[50] Alexandre David, Kim Guldstrand Larsen, Axel Legay, and Marius Mikučionis. Schedulability of Herschel-Planck Revisited Using Statistical Model Checking. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA (2)*, volume 7610 of *Lecture Notes in Computer Science*, pages 293–307. Springer, 2012. ISBN 978-3-642-34031-4. doi:10.1007/978-3-642-34032-1_28.

[51] Alexandre David, Kim Guldstrand Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, and Sean Sedwards. Runtime Verification of Biological Systems. In *ISoLA (1)*, pages 388–404, 2012. doi:10.1007/978-3-642-34026-0_29.

[52] Alexandre David, Kim G. Larsen, Axel Legay, and Danny Bøgsted Poulsen. Statistical Model Checking of Dynamic Networks of Stochastic Hybrid Automata. *ECEASST*, 66, 2013. Paper C in this thesis.

[53] Alexandre David, Kim G. Larsen, Axel Legay, Guangyuan Li, and Danny Bøgsted Poulsen. Quantified Dynamic Metric Temporal Logic for Dynamic Networks of Stochastic Hybrid Automata. In *14th International Conference on Application of Concurrency to System Design, ACSD 2014, Tunis La Marsa, Tunisia, June 23-27, 2014*, pages 32–41. IEEE, 2014. ISBN 978-1-4799-4281-7. doi:10.1109/ACSD.2014.21. Paper D in this thesis.

[54] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical model checking for biological systems. *STTT*, 17(3):351–367, 2015. doi:10.1007/s10009-014-0323-4. Paper E in this thesis.

[55] Doron Drusinsky. The Temporal Rover and the ATG Rover. In *SPIN*, pages 323–330, 2000.

[56] James R. Faeder, Michael L. Blinov, and William S. Hlavacek. Rule-Based Modeling of Biochemical Systems with BioNetGen. *Systems Biology*, 500, January 2009. doi:10.1007/978-1-59745-525-1_5.

[57] Ansgar Fehnker, Lodewijk van Hoesel, and Angelika Mader. Modelling and Verification of the LMAC Protocol for Wireless Sensor Networks. In Jim Davies and Jeremy Gibbons, editors, *Integrated Formal Methods*, volume 4591 of *LNCS*, pages 253–272. Springer Berlin / Heidelberg, 2007.

[58] José Luiz Fiadeiro and Antónia Lopes. A Model for Dynamic Reconfiguration in Service-Oriented Architectures. In *ECSA*, volume 6285 of *LNCS*, pages 70–85. Springer, 2010.

[59] Jasmin Fisher, Thomas A. Henzinger, Dejan Nickovic, Nir Piterman, Anmol V. Singh, and Moshe Y. Vardi. Dynamic Reactive Modules. In *CONCUR*, volume 6901 of *LNCS*, pages 404–418. Springer, 2011.

[60] Marc Geilen. An Improved On-the-fly Tableau Construction For a Real-Time Temporal Logic. In *In International Conference on Computer Aided Verification*, pages 276–290. Springer, 2003.

[61] Marc Geilen and Dennis Dams. An On-the-Fly Tableau Construction for a Real-Time Temporal Logic. In *FTRTFT*, pages 276–290, 2000.

[62] Thomas Gibson-Robinson, Philip J. Armstrong, Alexandre Boulgakov, and A. W. Roscoe. FDR3 - A Modern Refinement Checker for CSP. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 2014. ISBN 978-3-642-54861-1. doi:10.1007/978-3-642-54862-8_13.

[63] D. T. Gillespie. Exact Stochastic Simulation of Coupled Chemical Reactions. *Journal of Physical Chemistry*, 81:2340–2361, 1977. doi:10.1021/j100540a008.

[64] Haijun Gong, Paolo Zuliani, Anvesh Komuravelli, James R. Faeder, and Edmund M. Clarke. Computational Modeling and Verification of Signaling Pathways in Cancer. In *ANB*, volume 6479 of *LNCS*, pages 117–135. Springer, 2010.

[65] R. Grosu and S. A. Smolka. Monte Carlo Model Checking. In *Proc. of 11th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 3440 of *LNCS*, pages 271–286. Springer, 2005.

[66] J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods*. Methuen, 1975.

[67] Klaus Havelund and Kim Guldstrand Larsen. The Fork Calculus. *Nord. J. Comput.*, 1(3):346–363, 1994.

[68] Klaus Havelund and Grigore Rosu. Synthesizing Monitors for Safety Properties. In *TACAS*, LNCS, pages 342–356. Springer, 2002.

[69] Klaus Havelund, Arne Skou, Kim Guldstrand Larsen, and K. Lund. Formal modeling and analysis of an audio/video protocol: an industrial case study using UPPAAL. In *RTSS*, pages 2–13. IEEE Computer Society, 1997.

[70] Klaus Havelund, Kim Guldstrand Larsen, and Arne Skou. Formal Verification of a Power Controller Using the Real-Time Model Checker UPPAAL. In Joost-Pieter Katoen, editor, *ARTS*, volume 1601 of *Lecture Notes in Computer Science*, pages 277–298. Springer, 1999. ISBN 3-540-66010-0.

[71] Thomas A. Hennzinger. *The Temporal Specification and Verification of Real-Time Systems*. PhD thesis, Stanford University, 1991.

[72] Thomas A. Henzinger. The Theory of Hybrid Automata. In *LICS*, pages 278–292. IEEE Computer Society, 1996. ISBN 0-8186-7463-6.

[73] Thomas A. Henzinger and Vlad Rusu. Reachability Verification for Hybrid Automata. In Thomas A. Henzinger and Shankar Sastry, editors, *Hybrid Systems: Computation and Control, First International Workshop, HSCC'98, Berkeley, California, USA, April 13-15, 1998, Proceedings*, volume 1386 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 1998. ISBN 3-540-64358-3. doi:10.1007/3-540-64358-3_40.

[74] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A Model Checker for Hybrid Systems. *STTT*, 1(1-2):110–122, 1997. doi:10.1007/s100090050008.

[75] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate Probabilistic Model Checking. In *VMCAI*, volume 2937 of *LNCS*, pages 73–84. Springer, 2004.

[76] Robert C. Hilborn and Jessie D. Erwin. Stochastic coherence in an oscillatory gene circuit model. *Journal of Theoretical Biology*, 253(2):349 – 354, 2008. ISSN 0022-5193. doi:10.1016/j.jtbi.2008.03.012.

[77] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985. ISBN 0-13-153271-5.

[78] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[79] Gerad J. Holzmann. The Model Checker Spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[80] Cyrille Jegourel, Axel Legay, and Sean Sedwards. A Platform for High Performance Statistical Model Checking – PLASMA. In Cormac Flanagan and Barbara König, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *Lecture Notes in Computer Science*, pages 498–503. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-28755-8. doi:10.1007/978-3-642-28756-5_37.

[81] Sumit Kumar Jha, Edmund M. Clarke, Christopher James Langmead, Axel Legay, André Platzer, and Paolo Zuliani. A Bayesian Approach to Model Checking Biological Systems. In *CMSB*, volume 5688 of *LNCS*, pages 218–234. Springer, 2009.

[82] Robert M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, July 1976. ISSN 0001-0782. doi:10.1145/360248.360251.

[83] Ron Koymans. Specifying Real-Time Properties with Metric Temporal Logic. *Real-Time Systems*, 2(4):255–299, 1990.

[84] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 2.0: A Tool for Probabilistic Model Checking. In *QEST*, pages 322–323. IEEE Computer Society, 2004. ISBN 0-7695-2185-1.

[85] Kim Guldstrand Larsen and Arne Skou. Bisimulation Through Probabilistic Testing. In *POPL*, pages 344–352, 1989.

[86] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *STTT*, 1(1-2):134–152, 1997.

[87] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical Model Checking: An Overview. In *RV*, volume 6418 of *LNCS*, pages 122–135. Springer, 2010.

[88] Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal design and analysis of a gear controller. *STTT*, 3(3):353–368, 2001.

[89] Oded Maler, Dejan Nickovic, and Amir Pnueli. Real Time Temporal Logic: Past, Present, Future. In Paul Pettersson and Wang Yi, editors, *FORMATS*, volume 3829 of *Lecture Notes in Computer Science*, pages 2–16. Springer, 2005. ISBN 3-540-30946-2.

[90] Oded Maler, Dejan Nickovic, and Amir Pnueli. From MITL to timed automata. In *FORMATS'06*, pages 274–289. Springer, 2006.

[91] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980. ISBN 3-540-10235-3.

[92] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, I. *Inf. Comput.*, 100(1):1–40, 1992.

[93] Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, II. *Inf. Comput.*, 100(1):41–77, 1992.

[94] Faron Moller and Perdita Stevens. The Edinburg Concurrency Workbench user manual (Version 7.1). Technical report, University of Edinbrugh, 1999. URL `http://homepages.inf.ed.ac.uk/perdita/cwb/doc/manual.pdf`.

[95] Dejan Nickovic and Nir Piterman. From Mtl to Deterministic Timed Automata. In *FORMATS*, volume 6246 of *LNCS*, pages 152–167. Springer, 2010.

[96] Peter Olofsson. *Probability, statistics, and stochastic processes*. Wiley, 2005.

[97] Joël Ouaknine and James Worrell. On the Decidability of Metric Temporal Logic. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 188–197. IEEE Computer Society, 2005. ISBN 0-7695-2266-1. doi:10.1109/LICS.2005.33.

[98] Joël Ouaknine and James Worrell. On the decidability and complexity of Metric Temporal Logic over finite words. *Logical Methods in Computer Science*, 3(1), 2007. doi:10.2168/LMCS-3(1:8)2007.

[99] Andrew Phillips. *A Visual Process Calculus for Biology*. Jones and Bartlett Publishers, In Press, 2009. URL `http://research.microsoft.com/apps/pubs/default.aspx?id=79914`.

[100] Amir Pnueli. The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32. URL `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4567914`.

[101] Jean-Pierre Queille and Joseph Sifakis. Specification and Verification of Concurrent Systems in Cesar. In Orna Grumberg and Helmut Veith, editors, *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *Lecture Notes in Computer Science*, pages 216–230. Springer, 2008. ISBN 978-3-540-69849-4. doi:10.1007/978-3-540-69850-0_13.

[102] Diana El Rabih and Nihal Pekergin. Statistical Model Checking Using Perfect Simulation. In *ATVA*, volume 5799 of *LNCS*, pages 120–134. Springer, 2009.

[103] G. E. H. Reuter. Denumerable Markov Processes and the associated contracting semigroups onL. *Acta Mathematica*, 97(1-4):1–46, 1957.

[104] Grigore Rosu and Klaus Havelund. Rewriting-Based Techniques for Runtime Verification. *Autom. Softw. Eng.*, 12(2):151–197, 2005.

[105] S. Schivo, J. Scholma, B. Wanders, R.A. Urquidi Camacho, P.E. van der Vet, M. Karperien, R. Langerak, J. van de Pol, and J.N Post. Modelling biological pathway dynamics with Timed Automata. *Biomedical and Health Informatics, IEEE Journal of*, PP(99):1–1, 2013. ISSN 2168-2194. doi:10.1109/JBHI.2013.2292880.

[106] Stefano Schivo, Jetse Scholma, Brend Wanders, Ricardo A. Urquidi Camacho, Paul E. van der Vet, Marcel Karperien, Rom Langerak, Jaco van de Pol, and Janine N Post. Modelling biological pathway dynamics with Timed Automata. In *Proceedings of the 2012 IEEE 12th International Conference on Bioinformatics & Bioengineering (BIBE)*, pages 447–453, 2012.

[107] K. Sen, M. Viswanathan, and G. Agha. On Statistical Model Checking of Stochastic Systems. In *CAV*, LNCS 3576, pages 266–280, 2005.

[108] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Statistical Model Checking of Black-Box Probabilistic Systems. In *CAV*, LNCS 3114, pages 202–215, 2004.

[109] Koushik Sen, Mahesh Viswanathan, and Gul A. Agha. VESTA: A Statistical Model-checker and Analyzer for Probabilistic Systems. In *Second International Conference on the Quantitative Evaluaiton of Systems (QEST 2005), 19-22 September 2005, Torino, Italy*, pages 251–252. IEEE Computer Society, 2005. ISBN 0-7695-2427-3. doi:10.1109/QEST.2005.42. URL `http://dx.doi.org/10.1109/QEST.2005.42`.

[110] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003. doi:10.1101/gr.1239303.

[111] Amir Molzam Sharifloo and Paola Spoletini. LOVER: Light-Weight fOrmal Verification of adaptivE Systems at Run Time. In *FACS*, volume 7684 of *LNCS*, pages 170–187, 2012.

[112] Mihaela Sighireanu and Tayssir Touili. Bounded Communication Reachability Analysis of Process Rewrite Systems with Ordered Parallelism. *ENTCS*, 239:43–56, 2009.

[113] IEEE Computer Society. *Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs).* 2003.

[114] Jun Sun, Yang Liu, and Jin Song Dong. Model Checking CSP Revisited: Introducing a Process Analysis Toolkit. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA 2008, Porto Sani, Greece, October 13-15, 2008. Proceedings*, volume 17 of *Communications in Computer and Information Science*, pages 307–322. Springer, 2008. ISBN 978-3-540-88478-1. doi:10.1007/978-3-540-88479-8_22.

[115] Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *26th Annual Symposium on Foundations of Computer Science*, pages 327 –338, october 1985.

[116] Björn Victor and Faron Moller. The mobility workbench — A tool for the π-Calculus. In DavidL. Dill, editor, *Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer Berlin Heidelberg, 1994. ISBN 978-3-540-58179-6. doi:10.1007/3-540-58179-0_73.

[117] José M. G. Vilar, Hao Yuan Kueh, Naama Barkai, and Stanislas Leibler. Mechanisms of noise-resistance in genetic oscillators. *Proceedings of the National Academy of Sciences*, 99(9):5988–5992, 2002. doi:10.1073/pnas.092133899.

[118] A. Wald. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, 1945.

[119] R. Wald. *Sequential Analysis*. Dove Publisher, 2004.

[120] Håkan L. S. Younes. Ymer: A Statistical Model Checker. In Kousha Etessami and Sriram K. Rajamani, editors, *CAV*, volume 3576 of *LNCS*, pages 429–433, 2005. ISBN 3-540-27231-3.

[121] Håkan L. S. Younes and Reid G. Simmons. Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling. In *CAV*, LNCS 2404, pages 223–235. Springer, 2002. ISBN 3-540-43997-8. doi:10.1007/3-540-45657-0_17.

[122] Håkan L. S. Younes, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Numerical vs. statistical probabilistic model checking. *STTT*, 8 (3):216–228, 2006.

[123] Håkan L. S. Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Carnegie Mellon University, 2005.

[124] Paolo Zuliani, André Platzer, and Edmund M. Clarke. Bayesian statistical model checking with application to Simulink/Stateflow verification. In Karl Henrik Johansson and Wang Yi, editors, *HSCC*, pages 243–252. ACM ACM, 2010. ISBN 978-1-60558-955-8.

Acronyms

WMTL$_{[a,b]}$ weighted metric temporal logic$_{[a,b]}$.

WMTL$_{\leq}$ weighted metric temporal logic$_{\leq}$.

cdf cumulative distribution function.

CTL computational tree logic.

CTMC continuous time Markov chain.

DMTL dynamic metric temporal logic.

DTMC discrete time Markov chain.

FSM finite state machine.

LTL linear temporal logic.

LTS labelled transition system.

MC model checking.

MTL metric temporal logic.

MWTA monitoring weighted timed automaton.

NPTA Network of priced timed automata.

pdf probability density function.

PMC probabilistic model checking.

pmf probability mass function.

PTA priced timed automaton.

QDTML quantified dynamic metric temporal logic.

RV runtime verification.

SHA stochastic hybrid automaton.

SMC statistical model checking.

SPRT sequential probability ratio test.

STA stochastic timed automaton.

TA timed automaton.

TIOTS timed IO transition system.

TLTS timed labelled transition system.

WTA weighted timed automaton.

ISSN (online): 2246-1248
ISBN (online): 978-87-7112-527-6

AALBORG UNIVERSITY PRESS