Contributions to Statistical Model Checking

Axel Legay

Université de Rennes 1

Contributions to Statistical Model Checking

Année académique
2014 - 2015

Habilitation á diriger des recherches présentée par

Axel Legay

Composition du jury:

- J1
- J2
- J3

ABSTRACT

Statistical Model Checking (SMC) is a powerful and widely used approach that consists in estimating the probability for a system to satisfy a temporal property. This is done by monitoring a finite number of executions of the system, and then extrapolating the result by using statistics. The answer is correct up to some confidence that can be parameterized by the user. It is known that SMC mitigates the state-space explosion problem and allows us to handle requirements that cannot be expressed in classical temporal logics. The approach has been implemented in several toolsets, and successfully applied in a wide range of diverse areas such as systems biology, robotic, or automotive. Unfortunately, SMC is not a panacea and many important classes of systems and properties are still out of its scope. Moreover, In addition, SMC still indirectly suffers from an explosion linked to the number of simulations needed to converge when estimating small probabilities. Finally,the approach has not yet been lifted to a professional toolset directly usable by industry people.

In this thesis we propose several contributions to increase the efficiency of SMC and to wider its applicability to a larger class of systems. We show how to extend the applicability of SMC to estimate the probability of rare-events. The probability of such events is so small that classical estimators such as Monte Carlo would almost always estimate it to be null. We then show how to apply SMC to those systems that combine both non-deterministic and stochastic aspects. Contrary to existing work, we do not use a learning-based approach for the non-deterministic aspects, but rather exploit a smart sampling strategy. We then show that SMC can be extended to a new class of problems. More precisely, we consider the problem of detecting probability changes at runtime. We solve this problem by exploiting an algorithm coming from the signal processing area. We also propose an extension of SMC to real-time stochastic system. We provide a stochastic semantic for such systems, and show how to exploit it in a simulation-based approach. Finally, we also consider an extension of the approach for Systems of Systems.

Our results have been implemented in Plasma Lab, a powerful but flexible toolset. The thesis illustrates the efficiency of this tool on several case studies going from classical verification to more quixotic applications such as robotic.

Contents

Abstract v

Contents vii

1 Introduction 3
- 1.1 Context . . . 3
  - 1.1.1 The stochastic world: towards SMC . . . 5
- 1.2 Statistical model checking: a brief technical introduction . . . 6
  - 1.2.1 Stochastic systems and temporal logic . . . 7
  - 1.2.2 Verifying BLTL properties: a simulation approach . . . 9
  - 1.2.3 Qualitative answer using statistical model checking . . . 10
  - 1.2.4 Quantitative answer using statistical model checking and Estimation . . . 11
  - 1.2.5 On expected number of simulations . . . 11
- 1.3 Our contributions . . . 12
  - 1.3.1 The plasma-lab framework . . . 14

2 Rare-Event Simulation 19
- 2.1 Introduction . . . 19
- 2.2 Statistical model checking rare events . . . 20
  - 2.2.1 Importance Sampling . . . 20
  - 2.2.2 Importance splitting . . . 21
- 2.3 Score functions . . . 23
  - 2.3.1 Decomposition of a temporal logic formula . . . 24
- 2.4 Importance splitting algorithms . . . 26
  - 2.4.1 Fixed level algorithm . . . 26
  - 2.4.2 Adaptive level algorithm . . . 27
  - 2.4.3 Optimized adaptive level algorithm . . . 27
- 2.5 Case study: dining philosophers protocol . . . 30
  - 2.5.1 comparison between fixed and adaptive algorithm . . . 33
  - 2.5.2 Comparison with the optimized adaptive algorithm . . . 34
- 2.6 Conclusion . . . 35

3 Change Detection and Statistical Model Checking 37
- 3.1 Introduction . . . 37

3.2 Two new Problems . . . 38
3.2.1 The Optimization problem . . . 38
3.2.2 Change detection problem . . . 38
3.3 Quantitative verification and change detection: a statistical model checking approach . . . 39
3.3.1 Optimization . . . 39
3.3.2 change detection with CUSUM . . . 40
3.4 Plasma lab and simulink integration . . . 42
3.5 A Pig Shed Case study . . . 43
3.5.1 Quantitative Verification and Optimization . . . 44
3.5.2 Change Detection: Detection and Calibration . . . 46
3.6 Conclusion . . . 48

4 Motion planning in crowds using statistical model checking to enhance the social force model 49
4.1 Introduction . . . 49
4.2 Preliminaries . . . 51
4.2.1 Overview of the approach . . . 51
4.2.2 The social force model . . . 52
4.3 SMC–based Motion Planner . . . 54
4.4 Simulations . . . 56
4.5 Conclusion . . . 59

5 Statistical Model Checking for Markov Decision Processes 63
5.1 Introduction . . . 63
5.1.1 The Problems of schedulers and state explosion . . . 64
5.2 Related work . . . 66
5.3 Lightweight verification of MDPs . . . 67
5.3.1 General schedulers using hash functions . . . 67
5.3.2 An efficient iterative hash function . . . 68
5.3.3 Hypothesis testing multiple schedulers . . . 69
5.3.4 Estimating multiple schedulers . . . 70
5.4 Smart sampling . . . 72
5.4.1 Maximising the probability of seeing a good scheduler . . . 72
5.4.2 Smart estimation . . . 73
5.4.3 Smart hypothesis testing . . . 73
5.5 Experiments . . . 74
5.5.1 IEEE 802.11 Wireless LAN protocol . . . 75
5.5.2 IEEE 802.3 CSMA/CD protocol . . . 76
5.5.3 Choice coordination . . . 76
5.5.4 Network virus infection . . . 76
5.5.5 Gossip protocol . . . 77
5.6 Convergence and counterexamples . . . 78
5.7 Conclusion . . . 80

6 SMC for Stochastic Adaptive Systems **85**
6.1 Context . . . 85
6.2 Modeling Stochastic Adaptive Systems . . . 86
6.3 A logic for SAS properties . . . 88
6.3.1 Probabilistic Adaptive Bounded Linear Temporal Logic . . . 88
6.3.2 Verifying SAS Properties using SMC . . . 89
6.3.3 Verifying unbounded SAS Properties using SMC . . . 89
6.4 A software engineering point of view . . . 90
6.4.1 Adaptive RML systems as a high level formalism for SAS . . . 91
6.4.2 A contract language for SAS specification . . . 92
6.5 Experiments with SAS . . . 94
6.5.1 CAE model . . . 94
6.5.2 Checking requirements . . . 96
6.6 Conclusion . . . 97

7 Statistical model Checking for Priced timed stochastic automata **99**
7.1 Introduction . . . 99
7.2 Network of Priced Timed Automata . . . 101
7.3 Probabilistic Semantics of NPTA . . . 103
7.4 Statistical Model Checking for NPTA . . . 106
7.5 Statistical Model-Checking for comparisons . . . 106
7.6 Case Studies . . . 108
7.7 Conclusion and Extensions of the results . . . 112

8 Conclusion and Perspectives **115**
8.1 Perspectives for Statistical Model Checking . . . 115
8.2 Systems of Systems: a Vision . . . 116
8.2.1 Our Grand Challenge . . . 117
8.2.2 Potential Impact . . . 119

9 Appendix **121**
9.1 Curriculum Vitae . . . 121
9.1.1 Brief presentation of Axel Legay . . . 121
9.1.2 Supervision . . . 123
9.1.3 Services to Community . . . 124
9.1.4 Program committees . . . 125
9.1.5 Publications . . . 125
9.2 Brief Description of other works by the author . . . 126
9.2.1 Interface and Contract theories . . . 126
9.2.2 Software Product Lines . . . 127
9.2.3 Information Quantification flow . . . 127

Bibliography **129**

List of Figures

1.1 Plasma Lab architecture 15
1.2 Exponential scaling of numerical model checking vs. linear scaling of Plasma Lab SMC, considering a fairness property of the probabilistic dining philosophers protocol. 16
1.3 Scaling of Plasma Lab distributed algorithm applied to dining philosophers. Numbers are quantity of simulation nodes. Local simulation scaling is shown for reference. 16

2.1 Abstract dining philosopher. 30
2.2 Empirical number of levels. 30

3.1 Probability estimation with SMC of satisfying Φ_1 (left) and Φ_2 (right) 45
3.2 Probability estimation with SMC of satisfying Φ_3 without failures (left) and with failures (right) 45
3.3 Optimization of the thresholds parameters without failures (left) and with failures (right) 46

4.1 Diagrammatic overview of the motion planning framework. The sensor board detects the current state of objects in the environment. This state is used by the social force model to generate plausible future paths of the user and other pedestrians. The distribution of paths is verified against the global objectives of the user in order to suggest an optimal course. 52
4.2 Scenarios used to test the algorithm, *scenario 1* (a) and, *scenario 2* (b). 58
4.3 Scenario 2. Distances of the agents with respect to the user during one particular run of *scenario 2*. In this case the safety distance has been set to 0.5 m (dashed line) and has been violated once. 60

5.1 Model of network virus infection. 64
5.2 Fragment of a Markov decision process. 64
5.3 MDP with different optima for general and memoryless schedulers. . . 65
5.4 Empirical cumulative distribution of estimates from Algorithm 7 applied to MDP of Fig. 5.3. 72
5.5 Estimated maximum and minimum probabilities of second collision in WLAN protocol. Shaded regions denote true values ± 0.01. 75

5.6 Minimum probability of network infection. 77
5.7 Maximum probability of network infection. 77
5.8 Estimated probabilities that maximum path length is < 4 in gossip protocol model. Shaded regions denote ± 0.01 of true values. 78
5.9 Convergence of Algorithm 8 with exponentially distributed scheduler probabilities (Fig. 5.10) and per-iteration budget of 10^6 simulations. . 79
5.10 Theoretical linear (blue) and exponential (red) scheduler densities. Both have probability mass ≈ 0.0144 and zero density at scheduler probability 0.2. 80
5.11 Performance of smart sampling (black dots) applied to self-stabilising models of [IJ90]. Red shaded areas denote true values ± 0.01. 80

6.1 Illustration of SAS Methodology 86
6.2 SAS verification flow 90
6.3 Components and Views in the CAE model 95

7.1 Three composable NPTAs: A, B and T; A, B_r and T; and AB and T. 103
7.2 Cumulative probabilities for time and cost-bounded reachability of T_3. 105
7.3 Template of a train (a) and probability density distributions for $\Diamond_{T \leq t}$`Train(0).Cross` and $\Diamond_{T \leq t}$`Train(5).Cross`. 109
7.4 Comparing trains 0 and 5. 113
7.5 Collision probabilities when using exponential and uniform weights in chain and ring topologies, a) cumulative probability of collision over time and b) probability of having various numbers of collisions. . . . 113
7.6 Total energy consumption. 114
7.7 Rectangles are busy states and circles are for waiting when resources are not available. There are $r_1 = 5$ and $r_2 = 3$ resources available. . . 114

List of publications

The work presented in this thesis is based on the following publications:

- Cyrille Jegourel, Axel Legay, Sean Sedwards: A Platform for High Performance Statistical Model Checking - PLASMA. TACAS 2012: 498-503.

- Cyrille Jegourel, Axel Legay, Sean Sedwards: Cross-Entropy Optimisation of Importance Sampling Parameters for Statistical Model Checking. CAV 2012: 327-342.

- Alessio Colombo, Daniele Fontanelli, Axel Legay, Luigi Palopoli, Sean Sedwards: Motion planning in crowds using statistical model checking to enhance the social force model. CDC 2013: 3602-3608.

- Cyrille Jegourel, Axel Legay, Sean Sedwards: Importance Splitting for Statistical Model Checking Rare Properties. CAV 2013: 576-591.

- Benot Boyer, Kevin Corre, Axel Legay, Sean Sedwards: PLASMA-lab: A Flexible, Distributable Statistical Model Checking Library. QEST 2013: 160-164.

- Cyrille Jegourel, Axel Legay, Sean Sedwards: An Effective Heuristic for Adaptive Importance Splitting in Statistical Model Checking. ISoLA (2) 2014: 143-159.

Chapter 1

Introduction

This thesis presents a summary of some of the main results obtained by the author in the area Statistical Model Checking. The author also made fundamental contributions in the area of privacy, interface theory, and variability. Those topics will not be the main focus of this thesis, but a connection will be established at the end of the manuscript.

1.1 Context

Computers play a central role in modern life and their errors can have dramatic consequences. For example, such mistakes could jeopardize the banking system of a country or, more dramatically, endanger human life through the failure of some safety systems. It is therefore not surprising that proving the correctness of computer systems is a highly relevant problem.

The most common method to ensure the correctness of a system is *testing* (see [BJK+05] for a survey). After the computer system is constructed, it is tested using a number of *test cases* with predicted outcomes. Testing techniques have shown effectiveness in bug hunting in many industrial problems. Unfortunately, testing is not a panacea. Indeed, since there is, in general, no way for a finite set of test cases to cover all possible scenarios, errors may remain undetected.

There are also methods that can ensure the full correctness of a system. Those methods, also called *formal methods*, use mathematical techniques to check whether the system will behave correctly for all possible scenarios. There are several mathematical representations for a system. In this thesis, we will consider (extensions of) *Transition Systems*. The behaviors of a transition system can be represented by (possibly infinite) sequences of state changes and time stamps, which we call *executions*. The relation between successive states being obtained by a so-called *transition relation*. This relation may not be finite; it may also be implicit.

There is a long history of formal methods, going from logical proofs and invariants to model checking [BK08]. In this thesis, we focus on the second approach. It consists in checking that each behavior of the system satisfies a given requirement by exploring its state-space. In early work on the subject, requirements are

often expressed in some temporal logic such as Linear Temporal Logic [Pnu77], or computational Tree Logic [CE81]. Those logics extend classical Boolean logics with (quantification of) temporal operators that allows us to reason on the temporal dimension of a given execution.

It can be shown that solving the model checking problem boils down to compute a (repeated) set of reachable states [CGP99]. A simple state-space exploration technique starts the exploration from the set of initial states and then adds new reachable states by applying the reachability relation. If the number of states is finite, repeating this operation will eventually produce a stable set, that is the set of reachable states of the system. However, even for simple systems, finite-state spaces can be much too large to be computed and represented with realistic amounts of computer resources. For several decades now, researchers have been looking at ways to reduce the computational burden associated with these state space exploration based techniques.

A first family of strategies developed for coping with large state spaces is to exploit similarities and repetitive information. Among such techniques, one finds the so-called partial reduction [WG93,FG05]. This approach avoids the exploration of sequences of states by showing that their effect is already captured by another sequence. Another technique is called bisimulation reduction [DPP04]. It exploits equivalence classes of bisimilar states (i.e., states that generate the same behaviors) to reduce the state space. Predicate abstraction techniques [BMR05] extend bisimulation reduction by abstracting sets with a given predicate that subsumes their behaviors. The difficulty being to find the predicate that do not blow up the set of behaviors artificially. Predicate abstraction based techniques can be combined with CounterExample approaches used to calibrate the precision of the abstraction [CV03].

In addition to compute state-space, one of the major difficulties in model checking is to represent sets of state in an efficient way. One of the very first family of strategies developed for coping with large state spaces is based on symbolic methods which use symbolic representation to manipulate set of states implicitly rather than explicitly. Symbolic methods have managed to broaden the applicability of simple analysis methods, such as state space exploration, to systems with impressively large sets of states. One of the most used symbolic representation is known as Binary Decision Diagrams (BDD in short) [Bry92]. In BDDs, the states of the system are encoded with fixed-length bit vectors. In such a context, a finite set of states can be viewed as the set of solutions of a Boolean formula for which a BDD provides a representation that is often more compact than conjunctive or disjunctive normal form. This representation, algorithmically easy to handle, allows to efficiently represent the regular structure that often appears in the set of reachable states of finite state-transition systems. The BDD-based approach has been used to verify systems with more than 10^{20} reachable states [BCM+92], and it is now well-admitted that Boolean formal verification of large-size systems can be performed. Over the last decade, BDD have been replaced (or combined with) logical representation. Those consists in representing the sequence of states via formulas, and then use a sat-solvers

to check for a reachable state [BCCZ99, GPS02].

For two decades, logics and formal models did not exploit and model informations such as real-time or probabilities. This is however needed to reason large class of systems such as Embedded systems, Cyber physical systems, or systems biology. There, one is more interested in computing the level of energy needed to stay above a certain threshold, or the time needed to reach a given state. Motivated by this observation, the research community extended transitions systems with the ability to handle quantitative features. This includes, e.g., the formalism of timed automata [A.99] that exploits real-time informations to guide the executions, stochastic systems that can capture uncertainty in the executions, or weighted automata which permits to quantify the weight of a set of transitions [DG07]. In a similar fashion, LTL/CTL were extended with timed and quantitative informations. Those formalisms have been largely discussed in the literature, and have extended to other classes such as energy automata, or hybrid systems. It has been observed that reasoning on quantities amplifies the state-space explosion problem. However, tools such as Uppaal or Prism provided efficient approaches to partly overcome those problems. In this work, we focus on the stochastic aspects.

1.1.1 The stochastic world: towards SMC

Among the prominent extensions of transitions sytems, one finds quantitative sytems whose transitions are equipped with a probability distribution. This category includes, e.g., both discrete and continuous timed Markov Chains[1]. Our main interest will be in computing the probability to satisfy a given property of a stochastic system. This quantification replaces the Boolean world and permits us to quantify the impact of changes made on a given system.

Like classical transition systems, quantitative properties of stochastic systems are usually specified in linear temporal logics that allow one to compare the measure of executions satisfying certain temporal properties with thresholds. The model checking problem for stochastic systems with respect to such logics is typically solved by a numerical approach that, like state-space exploration, iteratively computes (or approximates) the exact measure of paths satisfying relevant subformulas. The algorithm for computing such measures depends on the class of stochastic systems being considered as well as the logics used for specifying the correctness properties. Model checking algorithms for a variety of contexts have been discovered [BHHK03, CY95, CG04] and there are mature tools (see e.g. [KNP04, CB06]) that have been used to analyze a variety of systems in practice.

Despite the great strides made by numerical model checking algorithms, there are many challenges. Numerical algorithms work only for special systems that have certain structural properties. Further the algorithms require a lot of time and space, and thus scaling to large systems is a challenge. In addition, the logics for which model checking algorithms exist are extensions of classical temporal logics, which are often not the most popular among engineers. Finally, those numerical techniques do

[1] As we shall see later, stochastic systems may deal with additional quantities such as real-time.

not allows us to consider extended stochastic models whose semantics also depends on other quantities such as real-time, or energy.

Another approach to verify quantitative properties of stochastic systems is to *simulate* the system for finitely many runs, and use techniques coming from the area of statistics to infer whether the samples provide a *statistical* evidence for the satisfaction or violation of the specification [YS02b]. The crux of this approach is that since sample runs of a stochastic system are drawn according to the distribution defined by the system, they can be used to get estimates of the probability measure on executions. Those techniques are known under the name of Statistical Model Checking (SMC).

The SMC approach enjoys many advantages. First, these algorithms only require that the system be simulatable (or rather, sample executions be drawn according to the measure space defined by the system). Thus, it can be applied to larger class of systems than numerical model checking algorithms including black-box systems and infinite state systems. Second the approach can be generalized to a larger class of properties, including Fourier transform based logics. Finally, the algorithm is easily parallelizable, which can help scale to large systems. In case the problem is undecidable or too complex, SMC is often the only viable solution. As we shall see latter, SMC has been the subject of intensive researches. SMC algorithms have been implemented in a series of tools such as Ymer [You05a], Prism [KNP11], or UPPAAL [DLL+11]. Recently, we have implemented a series of SMC techniques in a flexible and modular toolset called Plasma Lab [BCLS13]. As we shall see latter, this tool will be the main achievement of this thesis.

Unfortunately, SMC is not a panacea and many important classes of systems and properties are still out of its scope. Moreover, In addition, SMC still indirectly suffers from an explosion linked to the number of simulations needed to converge when estimating small probabilities. Finally,the approach has not yet been lifted to a professional toolset directly usable by industry people. Consequently, it remains unclear whether the approach can handle applications that are beyond the academic world.

In the rest of this chapter, we introduce the basic SMC algorithm that existed prior to our work, discuss the situation of SMC in 2008, and we present our contribution. We then conclude the section with a brief presentation of Plasma Lab as it will be used through the rest of the thesis.

1.2 Statistical model checking: a brief technical introduction

In this section, we introduce the basic notations that will be used in this thesis. We also briefly survey SMC algorithms used in foundation works on the topic. Those algorithms will be extended in the contribution chapters.

1.2.1 Stochastic systems and temporal logic

We consider a set of states S and a time domain $T \subseteq \mathbb{R}$. We first introduce the general definition of stochastic systems.

Definition 1.1.[Stochastic system] A *stochastic system* over S and T is a family of random variables $\mathcal{X} = \{X_t \mid t \in T\}$, each random variable X_t having range S.

The definition of a stochastic system as a family of random variables is quite general and includes systems with both continuous and discrete dynamics. In this thesis, we will focus our attention on a limited, but important, class of stochastic system: stochastic discrete event systems, which we note $\mathcal{S} = (S, T)$. This class includes any stochastic system that can be thought of as occupying a single state for a duration of time before an event causes an instantaneous state transition to occur. An *execution* for a stochastic system is any sequence of observations $\{x_t \in S \mid t \in T\}$ of the random variables $X_t \in \mathcal{X}$. It can be represented as a sequence $\omega = (s_0, t_0), (s_1, t_1), \ldots, (s_n, t_n) \ldots$, such that $s_i \in S$ and $t_i \in T$, with time stamps monotonically increasing, e.g. $t_i < t_{i+1}$. Let $0 \leq i \leq n$, we denote $\omega^i = (s_i, t_i), \ldots, (s_n, t_n)$ the suffix of ω starting at position i. Let $\overline{s} \in S$, we denote $Path(\overline{s})$ the set of executions of $\mathcal{X}$ that starts in state $(\overline{s}, 0)$ (also called initial state) and $Path^n(\overline{s})$ the set of executions of length n.

In [You05a], Youness showed that the executions set of a stochastic system is a measurable space, which defines a probability measure μ over $Path(\overline{s})$. The precise definition of μ depends on the specific probability structure of the stochastic system being studied.

We now instantiate our model into three models. Each model is obtained from stochastic discrete event systems by adding extra artifacts. Any such artifact being exploited in a specific SMC algorithm in the next chapters.

We first introduce a model in where each state is assigned to a set of variable. We assume a set of states variables SV. Assume that each state variable $x \in SV$ is assigned to domain $\mathbb{D}_x$, and define the valuation function V, such that $V(s, x) \in \mathbb{D}_x$ is the value of x in state s. Consider also

define the general structure for *stochastic discrete event systems*.

Definition 1.2.[Variable-state Stochastic Discrete Event System (VSDES)] A *variable-state stochastic discrete event system* is a stochastic system extended with initial state and variable assignments, i.e., $\mathcal{S}ys = \langle S, I, T, SV, V \rangle$, where (S, T) is a stochastic system, $I \subseteq S$ is the set of initial states, SV is a set of state variables and V is the valuation function.

We now consider (a restricted version of) discrete and continuous time Markov Chains. Markov Chains are particular classes of Stochastic systems in where the transition relation is finite.

Remark 1.3. The main difference between our definition of Markov chains and SDES is in the discretisation of the transition relation and the labeling of states with actions. We could have adopted another approach in where all the models are unified into one definition. We have preferred to keep those definitions separated in order to emphasize the features of the algorithms presented in the next chapters.
□

We first recap the definition of discrete transition systems, a class of transition systems where both the number of states and the transition relation are finite.

Definition 1.4. A (labeled) *discrete transition system* is a tuple $\mathcal{T} = (S, I, \Sigma, \rightarrow, AP, L)$ where S is a finite set of states, $I \subseteq S$ is the initial state, Σ is a finite set of actions, $\rightarrow: S \times \Sigma \times S$ is the transition relation, AP is a set of atomic propositions, and $L : S \rightarrow 2^{AP}$ is a state labeling function that assigns to each state the set of propositions that are valid in the state.

We denote by $s \xrightarrow{a} s'$ the transition $(s, a, s') \in \rightarrow$. We are now ready to introduce Discrete Markov Chains (MC).

Definition 1.5. A (labeled) DTMC is a tuple $\mathcal{D} = (S, I, \Sigma, \rightarrow, AP, L, \mathbf{P})$ where:

- $(S, I, \Sigma, \rightarrow, AP, L)$ is a labeled discrete transition system,
- $\mathbf{P} : S \times S \rightarrow [0, 1]$ is a *transition probability matrix*, such that $\sum_{s' \in S} \mathbf{P}(s, s') = 1$ for all $s \in S$,
- $\rightarrow$ is such that $s \xrightarrow{a} s'$ iff $\mathbf{P}(s, s') > 0$, and for each state s there is at most one action $a \in \Sigma$ such that $s \xrightarrow{a} s'$ for some s'.

In continuous time Markov chains (CTMCs) transitions are given a rate. The sum of rates of all enabled transitions specifies an exponential distribution that determines a real value for the time spent in the current state. The ratio of the rates then specifies which discrete transition is chosen. Formally, we have the following definition.

Definition 1.6. A (labeled) CTMC is a tuple $\mathcal{C} = (S, I, \Sigma, \rightarrow, AP, L, \mathbf{R})$ where:

- $(S, \overline{s}, \Sigma, \rightarrow, AP, L)$ is a labeled discrete transition system,
- $\mathbf{R} : S \times S \rightarrow \mathbb{R}_{\geq 0}$ is a *transition rate matrix*,
- $\rightarrow$ is such that $s \xrightarrow{a} s'$ iff $\mathbf{R}(s, s') > 0$, and there is a unique $a \in \Sigma$ such that $s \xrightarrow{a} s'$.

Requirements. In this thesis, except if explicitly mentioned,Properties over traces of $\mathcal{S}ys$ are defined via the so-called Bounded Linear Temporal Logic (BLTL). BLTL restricts Linear Temporal Logic by bounding the scope of the temporal operators. The syntax of BLTL is defined as follows:

$$\varphi = \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathrm{F}^{\leq t}\varphi \mid \mathrm{G}^{\leq t}\varphi \mid \varphi\,\mathrm{U}^{\leq t}\varphi \mid \mathrm{X}\varphi \mid \alpha$$

$\vee, \wedge$ and $\neg$ are the standard logical connectives and α is a Boolean constant or an atomic proposition constructed from numerical constants, state variables and relational operators. X is the *next* temporal operator: $\mathrm{X}\varphi$ means that φ will be true on the next step. F, G and U are temporal operators bounded by time interval $[0, t]$, relative to the time interval of any enclosing formula. We refer to this as a *relative interval*. F is the *finally* or *eventually* operator: $\mathrm{F}^{\leq t}\varphi$ means that φ will be true at least once in the relative interval $[0, t]$. G is the *globally* or *always* operator: $G^{\leq t}\varphi$ means that φ will be true at all times in the relative interval $[0, t]$. U is the *until* operator: $\psi \mathrm{U}_{\leq t}\varphi$ means that in the relative interval $[0, t]$, either φ is initially true or ψ will be true until φ is true. Combining these temporal operators creates complex properties with interleaved notions of *eventually* (F), *always* (G) and *one thing after another* (U).

1.2.2 Verifying BLTL properties: a simulation approach

Consider a stochastic system (S, T) and a property φ. *Statistical model checking* refers to a series of simulation-based techniques that can be used to answer two questions: (1) **Qualitative:** Is the probability that (S, T) satisfies φ greater or equal to a certain threshold? and (2) **Quantitative:** What is the probability that (S, T) satisfies φ? Contrary to numerical approaches, the answer is given up to some correctness precision. As we shall see latter, SMC solves those problems with two different approaches, while classical numerical approaches only solve the second problem, which implies the first one, but is harder.

In the rest of the section, we overview several statistical model checking techniques. Let B_i be a discrete random variable with a Bernoulli distribution of parameter p. Such a variable can only take 2 values 0 and 1 with $Pr[B_i = 1] = p$ and $Pr[B_i = 0] = 1 - p$. In our context, each variable B_i is associated with one simulation of the system. The outcome for B_i, denoted b_i, is 1 if the simulation satisfies φ and 0 otherwise. The latter is decided with the help of a monitoring[2] procedure [HR02]. The objective of an SMC algorithm is to generate simulations and exploit the Bernouili outcomes to extract a global confidence on the system.

In the next subsections, we present three algorithms used in history work on SMC to solve both the quantitative and the qualitative problems. Extension of those algorithms to unbounded temporal operators [SVA05, HCZ11] and to nested probabilistic operators exist [You05b]. As shown in [JKO+07] those extensions or debatable and often slower than their . Consequently, we will not discuss them.

[2]This thesis is not concerned with the definition of efficient monitoring procedures.

1.2.3 Qualitative answer using statistical model checking

The main approaches [You05a, SVA04] proposed to answer the qualitative question are based on *hypothesis testing*. Let $p = Pr(\varphi)$, to determine whether $p \geq \theta$, we can test $H : p \geq \theta$ against $K : p < \theta$. A test-based solution does not guarantee a correct result but it is possible to bound the probability of making an error. The *strength* (α, β) of a test is determined by two parameters, α and β, such that the probability of accepting K (respectively, H) when H (respectively, K) holds, called a Type-I error (respectively, a Type-II error), is less or equal to α (respectively, β).

A test has *ideal performance* if the probability of the Type-I error (respectively, Type-II error) is exactly α (respectively, β). However, these requirements make it impossible to ensure a low probability for both types of errors simultaneously (see [You05a] for details). A solution to this problem is to relax the test by working with an *indifference region* (p_1, p_0) with $p_0 \geq p_1$ ($p_0 - p_1$ is the *size of the region*). In this context, we test the hypothesis $H_0 : p \geq p_0$ against $H_1 : p \leq p_1$ instead of H against K. If the value of p is between p_1 and p_0 (the indifference region), then we say that the probability is sufficiently close to θ so that we are indifferent with respect to which of the two hypotheses K or H is accepted. The thresholds p_0 and p_1 are generally defined in terms of the single threshold θ, e.g., $p_1 = \theta - \delta$ and $p_0 = \theta + \delta$. We now need to provide a test procedure that satisfies the requirements above. In the next two subsections, we recall two solutions proposed by Younes in [You05a, You06].

Single Sampling Plan. This algorithm is more for history than for direct usage. However, it is still exploited in subsequent algorithms. To test H_0 against H_1, we specify a constant c. If $\sum_{i=1}^{n} b_i$ is larger than c, then H_0 is accepted, else H_1 is accepted. The difficult part in this approach is to find values for the pair (n, c), called a *single sampling plan (SSP in short)*, such that the two error bounds α and β are respected. In practice, one tries to work with the smallest value of n possible so as to minimize the number of simulations performed. Clearly, this number has to be greater if α and β are smaller but also if the size of the indifference region is smaller. This results in an optimization problem, which generally does not have a closed-form solution except for a few special cases [You05a]. In his thesis [You05a], Younes proposes a binary search based algorithm that, given p_0, p_1, α, β, computes an approximation of the minimal value for c and n.

Sequential Probability Ratio Test (SPRT). The sample size for a single sampling plan is fixed in advance and independent of the observations that are made. However, taking those observations into account can increase the performance of the test. As an example, if we use a single plan (n, c) and the $m > c$ first simulations satisfy the property, then we could (depending on the error bounds) accept H_0 without observing the $n - m$ other simulations. To overcome this problem, one can use the *sequential probability ratio test (SPRT in short)* proposed by Wald [Wal45a]. The approach is briefly described below.

In SPRT, one has to choose two values A and B ($A > B$) that ensure that the strength of the test is respected. Let m be the number of observations that have been made so far. The test is based on the following quotient:

$$\frac{p_{1m}}{p_{0m}} = \prod_{i=1}^{m} \frac{Pr(B_i = b_i | p = p_1)}{Pr(B_i = b_i | p = p_0)} = \frac{p_1^{d_m}(1-p_1)^{m-d_m}}{p_0^{d_m}(1-p_0)^{m-d_m}}, \tag{1.1}$$

where $d_m = \sum_{i=1}^{m} b_i$. The idea behind the test is to accept H_0 if $\frac{p_{1m}}{p_{0m}} \geq A$, and H_1 if $\frac{p_{1m}}{p_{0m}} \leq B$. The SPRT algorithm computes $\frac{p_{1m}}{p_{0m}}$ for successive values of m until either H_0 or H_1 is satisfied; the algorithm terminates with probability 1 [Wal45a]. This has the advantage of minimizing the number of simulations. In his thesis [You05a], Younes proposed a logarithmic based algorithm SPRT that given p_0, p_1, α and β implements the sequential ratio testing procedure.

SPRT has been largely used in the formal methods area. In this thesis, we shall show that the approach extends to a much larger class of problems that the one originally foreseen.

1.2.4 Quantitative answer using statistical model checking and Estimation

In the case of estimation, existing SMC algorithms rely on classical Monte Carlo estimation. More precisely, they calculate a priori the required number of simulations according to a Chernoff bound [Oka59] that allows the user to specify an error ε and a probability δ that the estimate $\hat{p}$ will not lie outside the true value $\pm\varepsilon$. Given that a system has true probability p of satisfying a property, the Chernoff bound ensures $\mathrm{P}(\mid \hat{p} - p \mid \geq \varepsilon) \leq \delta$. Parameter δ is related to the number of simulations N by $\delta = 2e^{-2N\varepsilon^2}$ [Oka59], giving

$$N = \left\lceil (\ln 2 - \ln \delta)/(2\varepsilon^2) \right\rceil . \tag{1.2}$$

1.2.5 On expected number of simulations

The efficiency of the above algorithms is characterized by the number of simulations needed to obtain an answer. This number may change from executions to executions and can only be estimated (see [You05a] for an explanation). However, some generalities are known. For the qualitative case, it is known that, except for some situations, SPRT is always faster than SSP. When $\theta = 1$ (resp. $\theta = 0$) SPRT degenerates to SSP; this is not problematic since SSP is known to be optimal for such values. Monte Carlo can also be used to solve the qualitative problem, but it is always slower than SSP [You05a]. If θ is unknown, then a good strategy is to estimate it using Monte Carlo with a low confidence and then validate the result with SPRT and a strong confidence.

1.3 Our contributions

This thesis presents some of the main contributions to SMC made by the author starting from 2008. SMC, which is informally used in industry since decades, was first introduced by Haakan Younes [YS02a], Koushik Sen, and Sylvain Peyronnet independently. The main contributions of Younes were mainly to show that a simulation approach can act as a unifying paradigm to reason on various classes of stochastic systems as well as to solve planing problems in robotics [SVA05], a subject that we will also tackle in the thesis. There was no effort to minimize the number of simulations and the applications were rather limited. The work of Sen explored more technical aspects of SMC such as unbounded properties or Black-box systems, but there was again no effort to minimize the number of simulations, or to handle complex problems. The work of Peyronnet was more on exploiting classical Monte Carlo simulation (as described below) than to make a strong contribution on SMC. Other contributors to SMC include Radu Grosu and Scott Smolka who showed how simulation can be used to approximate the LTL model checking problem [GS05]. Since 2008 other research teams have made significant contribution to SMC. They will be introduced gradually in the next chapters.

The main contributions we made comparing to history contributors is to 1. extend the applicability of SMC to a larger class of systems, 2. handle new problems, 3. show that SMC can be applied on large scaled problems, and 4. develop a professional toolset directly available at the industry level. In the rest of this section, we briefly summarize the contributions made during this period. We then introduce the Plasma Lab toolset [BCLS13], which is in fact our main contribution.

- Chapter 2 presents an extension of SMC that can be used to detect rare events. As we said above, SMC avoids the intractable growth of states associated with probabilistic model checking by estimating the probability of a property from simulations. Rare properties are often important, but pose a challenge for simulation-based approaches: the relative error of the estimate is unbounded. A key objective for statistical model checking rare events is thus to reduce the variance of the estimator. *Importance splitting* achieves this by estimating a sequence of conditional probabilities, whose product is the required result. To apply this idea to model checking it is necessary to define a *score function* based on logical properties, and a set of *levels* that delimit the conditional probabilities. In this chapter we motivate the use of importance splitting for statistical model checking and describe the necessary and desirable properties of score functions and levels. We illustrate how a score function may be derived from a property and give two importance splitting algorithms: one that uses fixed levels and one that discovers optimal levels adaptively.
- Chapter 3 presents an algorithm that can be used to monitor changes in the probability distribution to satisfy a bounded-time property at runtime. Concretely, the algorithm constantly monitors the execution of the deployed system, and rise a flag when it observes that the probability has changed significantly. This is done by extending the applicability of the CUSUM algorithm

used in signal processing into the formal validation setting. In this chapter, we also show how the programming interface of Plasma Lab can be exploited in order to make SMC technology directly available in toolsets used by designers. This integration is done by exploiting simulation facilities of design tools. Our approach thus differs from the one adopted by other SMC/formal verification toolsets which assume the existence of formal semantics for the design language, as well as a compiling chain to the rather academic one used by validation tool. The concept is illustrated with an integration of Plasma Lab as a library of the Simulink toolset. The contributions are illustrated with a pig shed case study.

- In Chapter 4, we consider the unpredictable social and physical interactions in Crowded environments. They pose a challenge to the comfort and safety of those with impaired ability. To address this challenge we have developed an efficient algorithm that may be embedded in a portable device. to assist such people. The algorithm anticipates undesirable circumstances in real time, by verifying simulation traces of local crowd dynamics against temporal logical formulae. The model incorporates the objectives of the user, pre-existing knowledge of the environment and real time sensor data. The algorithm is thus able to suggest a course of action to achieve the user's changing goals, while minimising the probability of problems for the user and others in the environment. To demonstrate our algorithm we have implemented it in an autonomous computing device that we show is able to negotiate complex virtual environments. The performance of our implementation demonstrates that our technology can be successfully applied in a portable device or robot.

- Chapter 5 presents an extension of SMC to model check Markov Decision Processes (MDP). MDP are useful to model optimisation problems in concurrent systems but verifying them with numerical methods is often intractable. Existing approximative approaches also do not scale well, hence we have developed the basis of scalable Monte Carlo verification for Markov decision processes. By devising an $\mathcal{O}(1)$ memory representation of history-dependent schedulers, we facilitate scalable learning techniques and the use of massively parallel verification. We also show that this results can be extended to model check properties with rewards.

- Complex systems such as systems of systems result from the combination of several components that are organized in a hierarchical manner. One of the main characteristics of those systems is their ability to adapt to new situations by modifying their architecture. Those systems have recently been the subject of a series of works in the software engineering community. Most of those works do not consider quantitative features. The objective of Chapter 6 is to propose a modeling language for adaptive systems whose behaviors depend on stochastic features. This language relies on an extension of stochastic transition systems equipped with (1) an adaptive operator that allows to reason about the probability that a system has to adapt its architecture over time,

and (2) dynamic interactions between processes. As a second contribution, the chapter propose a contract-based extension of probabilistic linear temporal logic suited to reason about assumptions and guarantees of such systems. The formalism has been implemented in Plasma Lab. First experiments on a large case study coming from the industrial driven European project DANSE give encouraging results.

- Chapter 7 offers a natural stochastic semantics of Networks of Priced Timed Automata (NPTA) based on races between components. The semantics provides the basis for satisfaction of Probabilistic Weighted Computational Tree Logic properties (PWCTL), conservatively extending the classical satisfaction of timed automata with respect to TCTL. In particular the extension allows for hard real-time properties of timed automata expressible in TCTL to be refined by performance properties, e.g. in terms of probabilistic guarantees of time- and cost-bounded properties. A second contribution of the chapter is the application of Statistical Model Checking (SMC) to efficiently estimate the correctness of non-nested PWCTL model checking problems with a desired level of confidence, based on a number of independent runs of the NPTA. In addition to applying classical SMC algorithms, we also offer an extension that allows to efficiently compare performance properties of NPTAs in a parametric setting. The third contribution is an efficient tool implementation of our result and applications to several case studies.

1.3.1 The plasma-lab framework

Dedicated SMC tools, such as YMER[3], VESPA, APMC[4] and COSMOS[5], have been joined by statistical extensions of established tools such as PRISM[6] and UPPAAL[7]. In the case of UPPAAL-SMC, this has required the definition of stochastic timed semantics. The tool MRMC[8] has both numerical and statistical functionality, but takes as input a low level textual description of a Markov chain. Many other tools are available or under development, with most using a single high level modeling language related to a specific semantics. Our first implementation of SMC algorithms [JLS12a] suffered the same limitation, prompting us to develop a radically new tool with modular architecture.

Plasma Lab is a compact, efficient and flexible platform for statistical model checking of stochastic models. The tool offers a series of SMC algorithms which includes those presented above as well as those that will be presented in the rest of this dissertation. The main difference between Plasma Lab and other SMC tools is that Plasma Lab proposes an API abstraction of the concepts of stochastic model simulator, property checker (monitoring) and SMC algorithm. In other words, the

[3] www.tempastic.org/ymer
[4] sylvain.berbiqui.org/apmc
[5] www.lsv.ens-cachan.fr/~barbot/cosmos/
[6] www.prismmodelchecker.org
[7] www.uppaal.org
[8] www.mrmc-tool.org

tool has been designed to be capable of using external simulators, input languages, or SMC algorithms. This not only reduces the effort of integrating new algorithms, but also allows us to create direct plug-in interfaces with industry used specification tools. The latter being done without using extra compilers.

Fig. 1.1 presents Plasma Lab architecture. More specifically, the relations between model simulators, property checkers, and SMC algorithms components. The simulators features include starting a new trace and simulating a model step by step. The checkers decide a property on a trace by accessing to state values. They also control the simulations, with a *state on demand* approach. A SMC algorithm component, such as the SPRT algorithm, is a runnable object. It collect samples obtained from a checker component. Depending on the property language, their checker either returns Boolean or numerical values. The algorithm then notifies progress and sends its results through the Controller API.

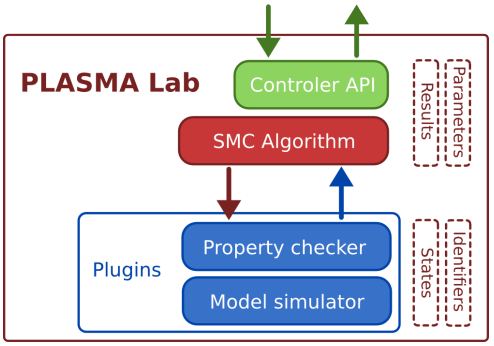


Figure 1.1: Plasma Lab architecture

In coordination with this architecture, we use a plugin system to load models and properties components. It is then possible to support new model or property languages. Adding a simulator, a checker or an algorithm component is pretty straightforward as they share a similar plugin architecture. Thus, it requires only a few classes and methods to get a new component running. Each plugin contains a factory class used by Plasma Lab to instantiate component objects. These components implement the corresponding interface defining their behavior. Some companion objects are also required (results, states, identifiers) to allow communication between components and the Controller API.

Languages Plasma Lab offers several input languages that permit to describe a wide range of systems. Some of those will be introduced in Chapters 5, 6, 3, and 7.

Properties Plasma Lab accepts properties described in a form of bounded linear temporal logic (BLTL) extended with custom temporal operators based on concepts such as *minimum*, *maximum* and *mean* of a variable over time. Some of those concepts will be illustrated in chapters 5 and 6.

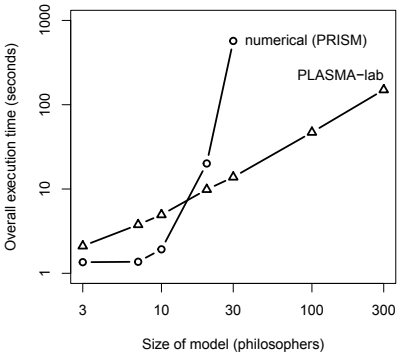


Figure 1.2: Exponential scaling of numerical model checking vs. linear scaling of Plasma Lab SMC, considering a fairness property of the probabilistic dining philosophers protocol.

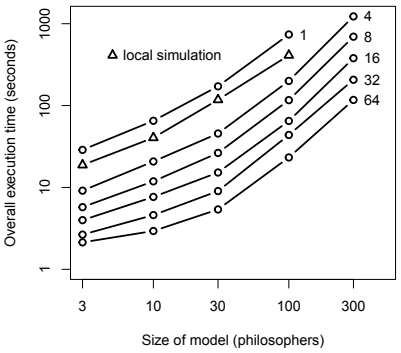


Figure 1.3: Scaling of Plasma Lab distributed algorithm applied to dining philosophers. Numbers are quantity of simulation nodes. Local simulation scaling is shown for reference.

Model checking Algorithms Plasma Lab offers all the SMC algorithms described above as well as several algorithms that will be described in Chapters 2 3, and 7.

Usage Plasma Lab may be invoked from the command line or embedded in other software as a library. Plasma Lab is provided as a pre-compiled jar file (plasmalab.jar) and a source template (Simulator.java) to create the simulator class. The minimum requirement is to implement the methods newTrace() and nextState(), that initiate a new simulation and advance the simulation by one step, respectively.

Graphical user interface The GUI provides an integrated development environment (IDE) to facilitate the use of Plasma Lab as a standalone statistical model checker with multiple 'drop-in' modeling languages. To demonstrate this, we have included a biochemical language and a language based on reactive modules. The website `https://project.inria.fr/plasma-lab/` includes other examples. The GUI implements the notion of a project file, that links the description of a model to a specific modeling language simulator and a set of associated properties and experiments. The GUI also provides 2D and 3D graphical output of results and implements a distributed algorithm that will work with any of its associated modeling languages.

Distributed algorithm The administrative time needed to distribute SMC on parallel computing architectures is often a deterrent. To overcome this, the Plasma Lab GUI implements a simple and robust client-server architecture, based on Java Remote Method Invocation (RMI) using IPv4/6 protocols. The algorithm will work on dedicated clusters and grids, but can also take advantage of ad hoc networks of heterogeneous computers. The minimum requirement is that the IP address of the GUI is available to the clients. Plasma Lab implements the SMC distribution algorithm of [You05b], which avoids the statistical bias that might otherwise occur

from load balancing. Distributed performance in illustrated in Fig. 1.3. The user selects the distributed mode via the GUI and publishes the IP address of the instance of Plasma Lab GUI that is acting as server. Clients (instances of the Plasma Lab service application) willing to participate respond by sending a message to the published IP address. The server sends an encapsulated version of the model and property to each of the participating clients, which then wait to be told how many simulations to perform. When sufficient clients are available, the user initiates the analysis by causing the server to broadcast the simulation requirements to each client.

Applications Our tool has been applied to several case studies. This includes those that will be presented in this thesis as well as some breakthrough in systems biology or in the verification of Systems of Systems.

Chapter 2

Rare-Event Simulation

2.1 Introduction

Statistical Model Checking can be used to estimate the probability to satisfy a property. So far, as we have seen in the introduction, this has been done with Monte Carlo method [MU49], which takes advantage of robust statistical techniques to bound the error of the estimated result (e.g., [Che52, Wal45b]). To quantify a property it is necessary to observe the property and increasing the number of observations generally increases the confidence of the estimate. Rare properties thus pose a problem to statistical model checking, since they are difficult to observe and often highly relevant to system performance (e.g., system failure is usually required to be rare). Fortunately, many Monte Carlo methods for rare events were devised in the early days of computing. In particular, *importance sampling* [Kah50, KM53] and *importance splitting* [KH51, KM53, RR55] may be successfully applied to statistical model checking.

Importance sampling and importance splitting have been widely applied to specific simulation problems in science and engineering. Importance sampling works by estimating a result using biased simulations and compensating for the bias. Importance splitting works by reformulating the rare probability as a product of less rare probabilities conditioned on levels that must be achieved. This chapter is dedicated to importance splitting, but the limitation of importance sampling are briefly presented.

Earlier work [VAVA91, GHSZ99] extended the original applications of importance splitting to more general problems of computational systems. Recent work has explicitly considered the use of importance sampling in the context of statistical model checking [RdBSH12, JLS12a, BHP12, JLS12b]. In what follows, we describe some of the limitations of importance sampling and motivate the use of importance splitting applied to statistical model checking, linking the concept of levels and score functions to temporal logic.

The remainder of the chapter is organised as follows. Section 2.2 discusses statistical model checking rare events and introduces importance sampling and splitting. Section 2.3 defines the important properties of score functions (required to define

levels) and describes how such functions may be derived from logical properties. Section 2.4 gives three importance splitting algorithms, while Section 2.5 illustrates their use on a case study.

2.2 Statistical model checking rare events

We consider a discrete-event stochastic system $\mathcal{S}$ and a temporal logic property φ that may be true or false with respect to an execution trace. Our objective is to calculate the probability γ that an arbitrary execution trace ω satisfies φ, denoted $\gamma = \mathrm{P}(\omega \models \varphi)$. To decide the truth of a particular trace ω', we define a model checking function $z(\omega) \in \{0, 1\}$ that takes the value 1 if $\omega' \models \varphi$ and 0 if $\omega' \not\models \varphi$.

Let Ω be the set of paths induced by $\mathcal{S}$, with $\omega \in \Omega$ and f a probability measure over Ω. Then

$$\gamma = \int_{\Omega} z(\omega)\, \mathrm{d}f \qquad (2.1) \qquad \text{and} \qquad \gamma \approx \frac{1}{N} \sum_{i=1}^{N} z(\omega_i)$$

N denotes the number of simulations and ω_i is sampled according to f. Note that $z(\omega_i)$ is effectively the realisation of a Bernoulli random variable with parameter γ. Hence $\mathrm{Var}(\gamma) = \gamma(1-\gamma)$ and for $\gamma \to 0$, $\mathrm{Var}(\gamma) \approx \gamma$.

When a property is not rare there are useful bounding formulae (e.g., the Chernoff bound [Che52]) that relate *absolute* error, confidence and the required number of simulations to achieve them. As the property becomes rarer, however, absolute error ceases to be useful and it is necessary to consider *relative error*, defined as the standard deviation of the estimate divided by its expectation. For a Bernoulli random variable the relative error is given by $\sqrt{\gamma(1-\gamma)}/\gamma$, that is unbounded as $\gamma \to 0$. In standard Monte Carlo simulation, γ is the expected fraction of executions in which the rare event will occur. If the number of simulation runs is significantly less than $1/\gamma$, as is necessary when γ is very small, no occurrences of the rare property will likely be observed. A number of simulations closer to $100/\gamma$ is desirable to obtain a reasonable estimate. Hence, the following techniques have been developed to reduce the number of simulations required or, equivalently, to reduce the variance of the rare event and so achieve greater confidence for a given number of simulations.

2.2.1 Importance Sampling

Importance sampling works by biasing the system dynamics in favour of a property of interest, simulating under the new dynamics, then unbiasing the result to give a true estimate. Referring to (2.1), let f' be another probability measure over Ω, absolutely continuous with respect to zf, then (2.1) can be rewritten

$$\gamma = \int_{\Omega} z(\omega) \frac{\mathrm{d}f(\omega)}{\mathrm{d}f'(\omega)}\, \mathrm{d}f' = \int_{\Omega} L(\omega) z(\omega)\, \mathrm{d}f'$$

where $L = \mathrm{d}f/\mathrm{d}f'$ is the *likelihood ratio*. We can thus estimate γ by simulating under f' and compensating by L: $\gamma \approx \frac{1}{N} \sum_{i=1}^{N} L(\omega_i) z(\omega_i)$. $L(\omega_i)$ may be calculated

with little overhead during individual simulation runs.

In general, the importance sampling measure f' is chosen to produce the rare property more frequently, but this is not the only criterion. The optimal importance sampling measure, denoted f^* and defined as f conditioned on the rare event, is exactly the distribution of the rare event: $f^* = zf/\gamma$. The challenge of importance sampling is to find a good *change of measure*, i.e., a measure f' that is close to f^*. An apparently good change of measure may produce the rare property more frequently (thus reducing the variance with respect to the *estimated* value) but increase the variance with respect to the *true* value.

It remains an open problem with importance sampling to quantify the performance of apparently 'good' distributions. A further challenge arises from properties and systems that require long simulations. In general, as the length of a path increases, its probability diminishes exponentially, leading to very subtle differences between f and f' and consequent problems of numerical precision.

As said above, Importance sampling will not be part of this thesis. The reader who is interested is redirected to [JLS12b] where we describe an efficient algorithm to find a change of measure that avoids this phenomenon.

2.2.2 Importance splitting

The earliest application of importance splitting is perhaps that of [KH51], where it was used to calculate the probability that neutrons would pass through certain shielding materials. This physical example provides a convenient analogy for the more general case. The system comprises a source of neutrons aimed at one side of a shield of thickness T. It is assumed that neutrons are absorbed by random interactions with the atoms of the shield, but with some small probability γ it is possible for a neutron to pass through the shield. The distance travelled in the shield can then be used to define a set of increasing levels $l_0 = 0 < l_1 < l_2 < \cdots < l_n = T$ that may be reached by the paths of neutrons, with the property that reaching a given level implies having reached all the lower levels. Though the overall probability of passing through the shield is small, the probability of passing from one level to another can be made arbitrarily close to 1 by reducing the distance between the levels.

These concepts can be generalised to simulation models of arbitrary systems, where a path is a simulation trace. By denoting the abstract level of a path as l, the probability of reaching level l_i can be expressed as $\mathrm{P}(l > l_i) = \mathrm{P}(l > l_i \mid l > l_{i-1})\mathrm{P}(l > l_{i-1})$. Defining $\gamma = \mathrm{P}(l > l_n)$ and observing $\mathrm{P}(l > l_0) = 1$, it is possible to write

$$\gamma = \prod_{i=1}^{n} \mathrm{P}(l > l_i \mid l > l_{i-1}) \tag{2.2}$$

Each term of the product (2.2) is necessarily greater than or equal to γ. The technique of importance splitting thus uses (2.2) to decompose the simulation of a rare event into a series of simulations of conditional events that are less rare. There have been many different implementations of this idea, but a generalised procedure

is as follows.

Assuming a set of increasing levels is defined as above, a number of simulations are generated, starting from a distribution of initial states that correspond to reaching the current level. The procedure starts by estimating $\mathrm{P}(l \geq l_1 | l \geq l_0)$, where the distribution of initial states for l_0 is usually given (often a single state). Simulations are stopped as soon as they reach the next level; the final states becoming the empirical distribution of initial states for the next level. Simulations that do not reach the next level (or reach some other stopping criterion) are discarded. In general, $\mathrm{P}(l \geq l_i | l \geq l_{i-1})$ is estimated by the number of simulation traces that reach l_i, divided by the total number of traces started from l_{i-1}. Simulations that reached the next level are continued from where they stopped. To avoid a progressive reduction of the number of simulations, the generated distribution of initial states is sampled to provide additional initial states for new simulations, thus replacing those that were discarded.

In physical and chemical systems, distances and quantities may provide a natural notion of level that can be finely divided. In the context of model-checking arbitrary systems, variables may be Boolean and temporal properties may not contain an obvious notion of level. To apply importance splitting to statistical model checking it is necessary to define a set of levels based on a sequence of temporal properties, φ_i, that have the logical characteristic

$$\varphi = \varphi_n \Rightarrow \varphi_{n-1} \Rightarrow \cdots \Rightarrow \varphi_0$$

Each φ_i is a strict restriction of the property φ_{i-1}, formed by the conjunction of φ_i with property ψ_i, such that $\varphi_i = \varphi_{i-1} \wedge \psi_i$, with $\varphi_0 \equiv \top$. Hence, φ_i can be written $\varphi_i = \bigwedge_{j=1}^{i} \psi_j$. This induces a strictly nested sequence of sets of paths $\Omega_i \subseteq \Omega$:

$$\Omega_n \subset \Omega_{n-1} \subset \cdots \subset \Omega_0$$

where $\Omega_i = \{\omega \in \Omega : \omega \models \varphi_i\}$, $\Omega_0 \equiv \Omega$ and $\forall \omega \in \Omega, \omega \models \varphi_0$. Thus, for arbitrary $\omega \in \Omega$,

$$\gamma = \prod_{i=1}^{n} \mathrm{P}(\omega \models \varphi_i \mid \omega \models \varphi_{i-1}),$$

that is analogous to (2.2).

A statistical model checker implementing bounded temporal logic will generally assign variables to track the status of the time bounds of temporal operators. Importance splitting requires these variables to be included as part of the state that is stored when a trace reaches a given level.

The choice of levels is crucial to the effectiveness of importance splitting. To minimise the relative variance of the final estimate it is desirable to choose levels that make $\mathrm{P}(\omega \models \varphi_i \mid \omega \models \varphi_{i-1})$ the same for all i (see, e.g., [DM04]). A simple decomposition of a property may give levels with widely divergent conditional probabilities, hence Section 2.3 introduces the concept of a *score function* and techniques that may be used to increase the possible resolution of levels. Given sufficient resolution, a further challenge is to define the levels. In practice, these are often guessed

or found by trial and error, but Section 2.4.2 gives an algorithm that finds optimal levels adaptively.

2.3 Score functions

Score functions generalise the concept of levels described in Section 2.2.2.

Definition 2.1. Let $J_0 \supset J_1 \supset \ldots \supset J_n$ be a set of nested intervals of $\mathbb{R}$ and let $\varphi_0 \Leftarrow \varphi_1 \Leftarrow \cdots \Leftarrow \varphi_n = \varphi$ be a set of nested properties. The mapping $\Phi : \Omega \to \mathbb{R}$ is a *level-based* score function of property φ if and only if $\forall k : \omega \models \varphi_k \iff \Phi(\omega) \in J_k$ and $\forall i, j \in \{0, \ldots, |\omega|\} : i < j \Rightarrow \Phi(\omega_{\leq i}) \leq \Phi(\omega_{\leq j})$

In general, the aim of a score function is to discriminate good paths from bad with respect to a property. In the case of a level-based score function, paths that have a higher score are clearly better because they satisfy more of the overall property. Given a nested sequence of properties $\varphi_0 = \top \Leftarrow \varphi_1 \Leftarrow \cdots \Leftarrow \varphi_n = \varphi$, a simple score function may be defined

$$\Phi(\omega) = \sum_{k=1}^{n} \mathbf{1}(\omega \models \varphi_k) \tag{2.3}$$

$\mathbf{1}(\cdot)$ is an indicator function taking the value 1 when its argument is true and 0 otherwise. Various ways to decompose a logical property are given in Section 2.3.1.

While a level-based score function directly correlates logic to score, in many applications the property of interest may not have a suitable notion of levels to exploit; the logical levels may be too coarse or may distribute the probability unevenly. For these cases it is necessary to define a more general score function.

Definition 2.2. Let $J_0 \supset J_1 \supset \ldots \supset J_n$ a set of nested intervals of $\mathbb{R}$ and $\Omega = \Omega_0 \supset \Omega_1 \supset \cdots \supset \Omega_n$ a set of nested subsets of Ω. The mapping $\Phi : \Omega \to \mathbb{R}$ is a *general* score function of property φ if and only if $\forall k : \omega \in \Omega_k \iff \Phi(\omega) \in J_k$ and $\omega \models \varphi \iff \omega \in \Omega_n$ and $\forall i, j \in \{0, \ldots, |\omega|\} : i < j \Rightarrow \Phi(\omega_{\leq i}) \leq \Phi(\omega_{\leq j})$

Informally, Definition 2.2 states that a general score function requires that the highest scores be assigned to paths that satisfy the overall property and that the score of a path's prefix is non-decreasing with increasing prefix length.

When no formal levels are available, an effective score function may still be defined using heuristics, that only loosely correlate increasing score with increasing probability of satisfying the property. For example, a time bounded property, not explicitly correlated to time, may become increasingly less likely to be satisfied as time runs out (i.e., with increasing path length). The heuristic in this case would assign higher scores to shorter paths. A score function based on coarse logical levels may be improved by using heuristics between the levels.

2.3.1 Decomposition of a temporal logic formula

Many existing uses of importance splitting employ a natural notion of levels inherent in a specific problem. Systems that do not have an inherent notion of level may be given quasi-natural levels by 'lumping' states of the model into necessarily consecutive states of an abstracted model. This technique is used in the dining philosophers example from Section 2.5.

For the purposes of statistical model checking, it is necessary to link levels to temporal logic. The following subsections describe various ways a logical formula may be decomposed into subformulae that may be used to form a level-based score function. The techniques may be used independently or combined with each other to give the score function greater resolution. Hence, the term 'property' used below refers both to the overall formula and its subformulae.

Since importance splitting depends on successively reaching levels, the initial estimation problem tends to become one of reachability (as in the case of numerical model checking algorithms). We observe from the following subsections that this does not necessarily limit the range of properties that may be considered.

Simple decomposition

When a property φ is given as an explicit conjunction of n sub-formulae, i.e., $\varphi = \bigwedge_{j=1}^{n} \psi_j$, a simple decomposition into nested properties is obtained by $\varphi_i = \bigwedge_{j=1}^{i} \psi_j, \forall i \in \{1, \ldots, n\}$, with $\varphi_0 \equiv \top$. The associativity and commutativity of conjunction make it possible to choose an arbitrary order of sub-formulae, with the possibility to choose an order that creates levels with equal conditional probabilities. Properties that are not given as conjunctions may be re-written using DeMorgan's laws in the usual way.

Natural decomposition

Many rare events are defined with a natural notion of level, i.e., when some quantity in the system reaches a particular value. In physical systems such a quantity might be a distance, a temperature or a number of molecules. In computational systems, the quantity might refer to a loop counter, a number of software objects, or the number of available servers, etc.

Natural levels are thus defined by nested atomic properties of the form $\varphi_i = (l > l_i), \forall i \in \{0, \ldots, n\}$, where l is a state variable, $l_0 = 0 < l_1 < \cdots < l_n$ and $\omega \models \varphi_n \iff l \geq l_n$. When rarity increases with decreasing natural level, the nested properties have the form $\varphi_i = l > l_i, \forall i \in \{0, \ldots, n\}$, with $l_0 = \max(l) > l_1 > \cdots > l_n$, such that $\omega \models \varphi_n \iff l \leq l_n$.

Time may be considered as a natural level if it also happens to be described by a state variable, however in the following subsection it is considered in terms of the bound of a temporal operator.

Decomposition of temporal operators

The following Propositions hold:

1. $(\varphi_n \Rightarrow \varphi_{n-1}) \Longrightarrow (\mathbf{F}^{\leq t}\varphi_n \Rightarrow \mathbf{F}^{\leq t}\varphi_{n-1})$
2. $(\varphi_n \Rightarrow \varphi_{n-1}) \Longrightarrow (\mathbf{G}^{\leq t}\varphi_n \Rightarrow \mathbf{G}^{\leq t}\varphi_{n-1})$
3. $(\varphi_n \Rightarrow \varphi_{n-1}) \Longrightarrow (\mathbf{X}\varphi_n \Rightarrow \mathbf{X}\varphi_{n-1})$
4. $(\varphi_n \Rightarrow \varphi_{n-1} \wedge \psi_m \Rightarrow \psi_{m-1}) \Longrightarrow (\varphi_n \mathbf{U} \psi_m \Rightarrow \varphi_{n-1}\mathbf{U}\psi_{m-1})$
5. $(\varphi_n \Rightarrow \varphi_{n-1}) \Longrightarrow (\mathbf{F}^{\leq t}\mathbf{G}^{\leq s}\varphi_n \Rightarrow \mathbf{F}^{\leq t}\mathbf{G}^{\leq s}\varphi_{n-1})$
6. $(\varphi_n \Rightarrow \varphi_{n-1}) \Longrightarrow (\forall \omega \models \mathbf{G}^{\leq t}\varphi_n : \exists t' \geq t \mid \omega \models \mathbf{G}^{\leq t'}\varphi_{n-1})$
7. $(\varphi_n \Rightarrow \varphi_{n-1}) \Longrightarrow (\forall \omega \models \mathbf{F}^{\leq t}\varphi_n : \exists t' \leq t \mid \omega \models \mathbf{F}^{\leq t'}\varphi_{n-1})$
8. $(t' \geq t) \Longrightarrow (\mathbf{F}^{\leq t}\mathbf{G}^{\leq s}\varphi_n \Rightarrow \mathbf{F}^{\leq t'}\mathbf{G}^{\leq s}\varphi_n)$
9. $(s' \leq s) \Longrightarrow (\mathbf{F}^{\leq t}\mathbf{G}^{\leq s}\varphi_n \Rightarrow \mathbf{F}^{\leq t}\mathbf{G}^{\leq s'}\varphi_n)$
10. $(t' \geq t \wedge s' \leq s) \Longrightarrow (\mathbf{F}^{\leq t}\mathbf{G}^{\leq s}\varphi_n \Rightarrow \mathbf{F}^{\leq t'}\mathbf{G}^{\leq s'}\varphi_n)$
11. $(\varphi_n \Rightarrow \varphi_{n-1}) \Longrightarrow (\forall \omega \models \mathbf{F}^{\leq t}\mathbf{G}^{\leq s}\varphi_n : \exists t' \leq t \wedge s' \geq s \mid \omega \models \mathbf{F}^{\leq t'}\mathbf{G}^{\leq s'}\varphi_{n-1})$

Temporal decomposition From Proposition 6, properties having the form $\varphi = \mathbf{G}^t\psi$ may be decomposed in terms of t. For an arbitrary suffix $\omega_{\geq k} = s_k \overset{t_k}{\rightarrow} s_{k+1} \overset{t_{k+1}}{\rightarrow} s_{k+2} \overset{t_{k+2}}{\rightarrow} \cdots$, we have $(\omega_{\geq k} \models \mathbf{G}^t\psi) \leftrightarrow (\omega_{\geq k} \models \psi) \wedge (\omega_{\geq k+1} \models \psi) \wedge \cdots \wedge (\omega_{k+m} \models \psi)$, for some m such that $\sum_{j=k}^{m+k} t_j \leq t \ \wedge \ \sum_{j=k}^{m+k+1} t_j > t$. This has the form required for a simple decomposition, giving nested properties of the form $\varphi_i = \mathbf{G}^{l_i}\psi, \forall i \in \{1, \ldots, n\}$, where $l_1 = 0 < l_2 < \cdots < l_n = t$, with $\varphi_0 \equiv \top$.

Properties having the form $\varphi = \mathbf{F}^t\psi$ evaluate to disjunctions in terms of time. From Proposition 7, it is plausible to construct nested properties of the form $\varphi_i = \mathbf{F}^{t+l_i}\psi, \forall i \in \{1, \ldots, n\}$, with $l_1 > l_2 > \cdots > l_n = 0$ and $\varphi_0 \equiv \top$. Some caution is required if t is the value given in the overall property. If trace ω satisfies $\mathbf{F}^{t'}$ but not $\mathbf{F}^t$, any prefix of ω does not satisfy $\mathbf{F}^t$. The requirement for $\mathbf{F}^{t'}$to have a lower score than $\mathbf{F}^t$ conflicts with the requirement of a score function $\forall i, j \in \{0, \ldots, |\omega|\} : i < j \Rightarrow \Phi(\omega_{\leq i}) \leq \Phi(\omega_{\leq j})$.

Heuristic decomposition

The decomposition of a property into logical levels may not necessarily result in an adequate score function: there may be insufficient levels, the levels may be irrelevant to the overall property or the levels may not evenly distribute the probability. In such cases it may be desirable to define intermediate levels based on heuristics – approximate correlations between a path and its probability to satisfy the property.

For example, $\varphi_i = \mathbf{F}^{t+l_i}\psi$ may not form legitimate nested properties with positive l_i, but may nevertheless be used as a heuristic with $l_i \in [-t, 0]$.

Note that a heuristic score function that respects Definition 2.2 will give an unbiased estimate when used with an unbiased importance splitting algorithm. The effectiveness of a heuristic is dependent on how well it correlates path prefixes with the probability of eventually satisfying the overall property.

2.4 Importance splitting algorithms

We give three importance splitting pseudo-algorithms based on [CMFA12]; in the first one, levels are fixed and defined a priori, the number of levels is an input of the algorithm; in the second one, levels are found adaptively with respect to a predefined probability, the number of levels is a random variable and is not an input anymore; the third one is an extension of the second where the probability to cross a level from a previous stage is set to its maximum. By N we denote the number of simulations performed at each level. Thresholds, denoted τ, are usually but not necessarily defined as values of score function $S(\omega)$, where ω is a path. τ_φ is the minimal threshold such that $S(\omega) \geq \tau_\varphi \iff \omega \models \varphi$. τ_k is the k^{th} threshold and ω_i^k is the i^{th} simulation on level k. $\tilde{\gamma}_k$ is the estimate of γ_k, the k^{th} conditional probability $\mathrm{P}(S(\omega) \geq \tau_k \mid S(\omega) \geq \tau_{k-1})$.

2.4.1 Fixed level algorithm

The fixed level algorithm follows from the general description given in Section 2.2.2. Its advantages are that it is simple, it has low computational overhead and the resulting estimate is unbiased. Its disadvantage is that the levels must often be guessed by trial and error – adding to the overall computational cost.

In Algorithm 1, $\tilde{\gamma}$ is an unbiased estimate (see, e.g., [DM04]). Furthermore, from Proposition 3 in [CMFA12], we can deduce the following $(1-\alpha)$ confidence interval:

$$CI = \left[\tilde{\gamma}\left(\frac{1}{1+\frac{z_\alpha \sigma}{\sqrt{N}}}\right), \tilde{\gamma}\left(\frac{1}{1-\frac{z_\alpha \sigma}{\sqrt{N}}}\right)\right] \qquad \text{with} \qquad \sigma^2 \geq \sum_{k=1}^{M} \frac{1-\gamma_k}{\gamma_k}, \tag{2.4}$$

where z_α is the $1 - \frac{\alpha}{2}$ quantile of the standard normal distribution. Hence, with confidence $100(1-\alpha)\%$, $\gamma \in CI$. For any fixed M, the minimisation problem

$$\min \sum_{k=1}^{M} \frac{1-\gamma_k}{\gamma_k} \qquad \text{with constraint} \qquad \prod_{k=1}^{M} \gamma_k = \gamma$$

implies that σ is reduced by making all γ_k equal.

For given γ, this motivates fine grained score functions. When it is not possible to define γ_k arbitrarily, the confidence interval may nevertheless be reduced by increasing N. The inequality for σ arises because the independence of initial states diminishes with increasing levels: unsuccessful traces are discarded and new

initial states are drawn from successful traces. Several possible ways to minimise these dependence effects are proposed in [CMFA12]. In the following, for the sake of simplicity, we assume that this goal is achieved. In the confidence interval, σ is estimated by the square root of $\sum_{k=1}^{M} \frac{1-\tilde{\gamma}_k}{\tilde{\gamma}_k}$.

Algorithm 1: Fixed levels

Let $(\tau_k)_{1 \leq k \leq M}$ be the sequence of thresholds with $\tau_M = \tau_\varphi$
Let *stop* be a termination condition
$\forall j \in \{1, \ldots, N\}$, set prefix $\tilde{\omega}_j^1 = \epsilon$ (empty path)
for $1 \leq k \leq M$ **do**
    $\forall j \in \{1, \ldots, N\}$, using prefix $\tilde{\omega}_j^k$, generate path ω_j^k until $(S(\omega_j^k) \geq \tau_k) \vee stop$
    $I_k = \{\forall j \in \{1, \ldots, N\} : S(\omega_j^k) \geq \tau_k\}$
    $\tilde{\gamma}_k = \frac{|I_k|}{N}$
    $\forall j \in I_k$, $\tilde{\omega}_j^{k+1} = \omega_j^k$
    $\forall j \notin I_k$, let $\tilde{\omega}_j^{k+1}$ be a copy of ω_i^k with $i \in I_k$ chosen uniformly randomly
end
$\tilde{\gamma} = \prod_{k=1}^{M} \tilde{\gamma}_k$

2.4.2 Adaptive level algorithm

The cost of finding good levels must be included in the overall computational cost of importance splitting. An alternative to trial and error is to use an adaptive level algorithm that discovers its own optimal levels.

Algorithm 2 is an adaptive level importance splitting algorithm presented first in [CG07]. It works by pre-defining a fixed number N_k of simulation traces to retain at each level. With the exception of the last level, the conditional probability of each level is then nominally N_k/N. Making N_k all equal minimizes the overall relative variance and is only possible if the score function has sufficient granularity.

2.4.3 Optimized adaptive level algorithm

Algorithm 3 defines an optimized adaptive level importance splitting algorithm. The variance of the estimate $\tilde{\gamma}$ is:

$$Var(\tilde{\gamma}) = \frac{p^2}{N}\left(n_0 \frac{1-\gamma_0}{\gamma_0} + \frac{1-r_0}{r_0} + o(N^{-1})\right)$$

and the function $f : \gamma_0 \longmapsto \frac{1-\gamma_0}{-\gamma_0 \log \gamma_0}$ is strictly decreasing on $]0, 1[$. Increasing γ_0 therefore decreases the variance. Ideally, this value is $\gamma_0 = 1 - \frac{1}{N}$ but it is more realistic to fix this value for each iteration k at $\gamma_0 = 1 - \frac{N_k}{N}$, with N_k the number of paths achieving the minimal score. Another advantage of this optimized version is that, although the number of steps before the algorithm terminates is more important, we only rebranch a few discarded traces (ideally only 1) per iteration.

Algorithm 2: Adaptive levels

Let $\tau_\varphi = \min\{S(\omega) \mid \omega \models \varphi\}$ be the minimum score of paths that satisfy φ
Let N_k be the pre-defined number of paths to keep per iteration
$k = 1$
$\forall j \in \{1, \ldots, N\}$, generate path ω_j^k
repeat
    Let $T = \left\{S(\omega_j^k), \forall j \in \{1, \ldots, N\}\right\}$
    Find maximum $\tau_k \in T$ such that $|\{\tau \in T : \tau > \tau_k\}| \geq N - N_k$
    $\tau_k = \min(\tau_k, \tau_\varphi)$
    $I_k = \{j \in \{1, \ldots, N\} : S(\omega_j^k) > \tau_k\}$
    $\tilde{\gamma}_k = \frac{|I_k|}{N}$
    $\forall j \in I_k$, $\omega_j^{k+1} = \omega_j^k$
    for $j \notin I_k$ **do**
        choose uniformly randomly $l \in I_k$
        $\tilde{\omega}_j^{k+1} = \max_{|\omega|} \left\{\omega \in \mathit{pref}(\omega_l^k) : S(\omega) < \tau_k\right\}$
        generate path ω_j^{k+1} with prefix $\tilde{\omega}_j^{k+1}$
    end
    $M = k$
    $k = k + 1$
until $\tau_k > \tau_\varphi$;
$\tilde{\gamma} = \prod_{k=1}^{M} \tilde{\gamma}_k$

Algorithm 3: Optimized adaptive levels

Let $\tau_\varphi = \min\{S(\omega) \mid \omega \models \varphi\}$ be the minimum score of paths that satisfy φ
$k = 1$
$\forall j \in \{1, \ldots, N\}$, generate path ω_j^k
repeat
  Let $T = \left\{S(\omega_j^k), \forall j \in \{1, \ldots, N\}\right\}$
  $\tau_k = \min T$
  $\tau_k = \min(\tau_k, \tau_\varphi)$
  $I_k = \{j \in \{1, \ldots, N\} : S(\omega_j^k) > \tau_k\}$
  $\tilde{\gamma}_k = \frac{|I_k|}{N}$
  $\forall j \in I_k$, $\omega_j^{k+1} = \omega_j^k$
  for $j \notin I_k$ **do**
    choose uniformly randomly $l \in I_k$
    $\tilde{\omega}_j^{k+1} = \max_{|\omega|} \left\{\omega \in \mathit{pref}(\omega_l^k) : S(\omega) < \tau_k\right\}$
    generate path ω_j^{k+1} with prefix $\tilde{\omega}_j^{k+1}$
  end
  $M = k$
  $k = k + 1$
until $\tau_k > \tau_\varphi$;
$\tilde{\gamma} = \prod_{k=1}^{M} \tilde{\gamma}_k$

2.5 Case study: dining philosophers protocol

Our work has been implemented as a prototype extension of Plasma Lab. We have adapted a case study from the literature to illustrate the use of heuristic-based score functions and of the optimized adaptive splitting algorithm with statistical model checking.

We have defined a rare event in the well known probabilistic solution [LR81] of Dijkstra's dining philosophers problem . In this example, there are no natural counters to exploit, so levels must be constructed by considering 'lumped' states.

A number of philosophers sit at a circular table with an equal number of chopsticks; a chopstick being placed within reach of two adjacent philosophers. Philosophers think and occasionally wish to eat from a communal bowl. To eat, a philosopher must independently pick up two chopsticks: one from the left and one from the right. Having eaten, the philosopher replaces the chopsticks and returns to thinking. A problem of concurrency arises because a philosopher's neighbour(s) may have already taken the chopstick(s). Lehmann and Rabin's solution [LR81] is to allow the philosophers to make probabilistic choices.

We consider a model of 150 'free' philosophers [LR81]. The number of states in the model is more than 10^{177}; 10^{97} times more than the estimated number of protons in the universe. The possible states of an individual philosopher can be abstracted to those shown in Fig. 2.1.

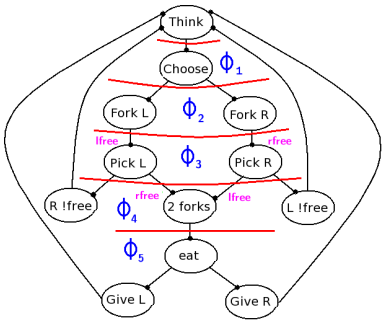


Figure 2.1: Abstract dining philosopher.

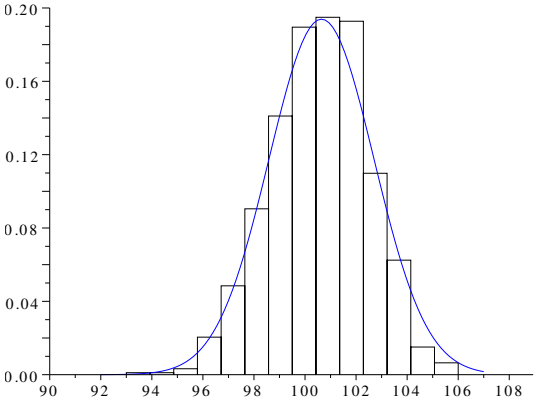


Figure 2.2: Empirical number of levels.

Think is the initial state of all philosophers. In state Choose, the philosopher makes a choice of fork he will try to get first. The transitions labelled by lfree or rfree in Fig. 2.1 are dependent on the availability of respectively left or right chopsticks. All transitions are controlled by stochastic rates and made in competition with the transitions of other philosophers. With increasing numbers of philosophers, it is increasingly unlikely that a specific philosopher will be satisfied (i.e., that the philosopher will reach the state eat) within a given number of steps from the initial state. We thus define a rare property $\varphi = \mathbf{F}^t\mathsf{eat}$, with t initially 30, denoting the property that a given philosopher will reach state eat within 30 steps. Thus, using the states of the abstract model, we decompose φ into nested properties $\varphi_0 = \mathbf{F}^t\mathsf{Think} =$

initial parameters n, $\tau_k - \tau_{k-1}$, γ_0	fixed alg.	adaptive alg.	optimized alg
Number n of path at first iteration	YES	YES	YES
Step between levels $(\tau_k - \tau_{k-1})$	YES	NO	NO
conditionnal probability γ_0	NO	YES	NO

Table 2.1: Parameters in each ISp algorithm.

$\top$, $\varphi_1 = \mathbf{F}^t\mathsf{Choose}$, $\varphi_2 = \mathbf{F}^t\mathsf{Try}$, $\varphi_3 = \mathbf{F}^t 1^{\mathsf{st}}\mathsf{stick}$, $\varphi_4 = \mathbf{F}^t 2^{\mathsf{nd}}\mathsf{stick}$, $\varphi_5 = \mathbf{F}^t\mathsf{eat}$. The red lines crossing the transitions indicate these formal levels on the graph.

Monte Carlo simulations with PLASMA statistical model checker

With such a large state space it is not possible to obtain a reference result with numerical model checking. We therefore performed extensive Monte Carlo simulations using the parallel computing capability of Plasma Lab. The experiment generated 300 million samples using 255 cores and took about 50 minutes. Our reference probability is thus approximately equal to 1.59×10^{-6} with 95%-confidence interval $[1.44 \times 10^{-6}; 1.72 \times 10^{-6}]$.

Recall and experiment protocol

Table 2.1 recalls, given a score function, that the parameters of each algorithm for an experiment are the number n of simulations used at the first iteration and the distance between levels (usually constant) in the fixed level algorithm or a probability between levels for the adaptive algorithms.

Note that the conditionnal probability in the optimized algorithm is a function of n more than an independent parameter.

Four types of importance splitting experiments are driven. The first one uses the simple score function and the fixed algorithm, the second uses the heuristic score function and the fixed-level algorithm (with different step values). The third algorithm uses the adaptive-level algorithm with different γ_0 parameters and finally the fourth set of experiments uses the optimized version of the adaptive algorithm.

For each set of experiments and chosen parameters, experiments are repeated 100 times in order to check the reliability of our results. In what follows, we remind which statistical notions are exploited and why:

- Number of experiments: used to estimate the variance of the estimator.
- Number of path per iteration: it is a parameter of the algorithm, equal to the number of paths that we use to estimate a conditionnal probability.
- Number of levels: known in the fixed algorithm, variable in the adaptive algorithms. In the second case, an average is provided.
- Time in seconds: the average of the 100 experiments is provided.

- The mean estimate is the estimator $\tilde{\gamma}$ of the probability of interest. The average of the 100 estimators is provided.
- The relative standard deviation of $\tilde{\gamma}$ is estimated with the 100 final estimators γ. A reliable estimator must have a low relative standard deviation (roughly ≤ 0.3).
- The mean value of γ_k is the average of the mean values of the conditionnal probabilities in an experiment. It is variable in the fixed algorithm and supposed to be a constant γ_0 in the adaptive algorithms. Because of the discreteness of the score function, the value is only almost constant and slightly lower than γ_0.
- The relative standard deviation of γ_k is the average of the relative standard deviations of the conditionnal probabilities in an experiment. By construction, the value in the adaptive algorithms must be low.

Comparison between logical and heuristic score function

Let ω be a path of length $t = 30$. For each prefix $\omega_{\leq j}$ of length j, we define the following function:

$$\Psi(\omega_{\leq j}) = \sum_{k=0}^{n} \mathbf{1}(\omega_{\leq j} \models \varphi_k) - \frac{\{\sum_{k=1}^{n} \mathbf{1}(\omega_{\leq j} \models \varphi_k)\} - j}{\sum_{k=1}^{n} \mathbf{1}(\omega_{\leq j} \models \varphi_k) - (t+1)}$$

We define score of ω as follows:

$$S(\omega) = \max_{1 \leq j \leq K} \Psi(\omega_{\leq j})$$

In the following experiment this score function is defined for any path of length $t+1$, starting in the initial state 'all philosophers think'. The second term of Ψ is a number between 0 and 1, linear in j such that the function gives a greater score to paths which satisfy a greater number of sub-properties φ_k and discriminates between two paths satisfying the same number of sub-properties by giving a greater score to the shortest path. A score in $]i-1, i]$ implies that a prefix of the path satisfied at most φ_i. We then compare results with the simple score function $S(\omega) = \sum_{k=1}^{n} \infty(\omega \models \varphi_k)$.

The experiments are repeated 100 times in order to demonstrate and improve the reliability of the results. Each conditional probability γ_k is estimated with a sample of 1000 paths.

For simplicity we consider a linear growing of score thresholds when we use the fixed-level algorithm. The simple score function thresholds increase by 1 between each level. When using the heuristic score function, we performed three sets of experiments involving an increase of 0.2, 0.1 and 0.05 of the thresholds. These partitions imply respectively 5, 20, 40 and 80 levels.

Table 2.2 shows that the simple score function likely gives a strong underestimation. It is due to the huge decrease of value of conditional probabilities between the

Statistics	Simple score function	Heuristic score function		
number of experiments	100	100	100	100
number of path per iteration	1000	1000	1000	1000
number of levels	5	20	40	80
Time in seconds (average)	6.95	13.42	16.64	21.56
mean estimate $\times 10^6$ (average)	0.01	0.59	1	1.37
mean value of $\tilde{\gamma_k}$	0.06	0.53	0.73	0.86
relative standard deviation of $\tilde{\gamma_k}$	1.04	0.36	0.22	0.15

Table 2.2: Comparison between fixed-level algorithms.

logical levels. All the estimated conditional probabilities are small and imply a large theoretical relative variance ($V(\tilde{\gamma})/E[\tilde{\gamma}]$). The final levels are difficult to cross and have probabilities close to 0. A sample size of 1000 paths is obviously not enough for the last step. On average $\tilde{\gamma}_5 = 0.003$ and in one case the last step is not satisfied by any trace, such that the estimate is equal to zero.

If a threshold is not often exceeded, it implies that traces will be rebranched from a very small set of first entrance states at the next level. This leads to significant relative variance between experiments. A further problem is that the conditional estimate is less efficient if γ_k is small. Increasing the number of evenly spaced levels decrease *a priori* more smoothly the conditional probabilities and reinforce the reliability of the results as soon as the relative standard deviation of conditional probabilities decreases enough. In the experiments, as expected, the mean value of conditional probabilities is positively correlated to the number of levels (respectively 0.06, 0.53, 0.73 and 0.86) and negatively correlated to the relative standard deviation of conditional probabilities. The results with 40 and 80 levels give results that are apparently close to the reference estimate, but are nevertheless consistently underestimates. This suggests that the number of simulations per level is too low.

Two questions arise: how to detect that the simulation is not efficient or robust and how to improve the results. In answer to the first, there are no general criteria for judging the quality of an importance splitting estimator. However, assuming that experiments are repeated a few times, a large relative error of the estimators (roughly ≥ 0.5), a very low value of conditional probability estimates, or a large relative error of conditional probability estimates (roughly ≥ 0.2) are good warnings. As for the second question, a way to improve results with the fixed level algorithm is simply to increase the number of paths per level or to increase the number of levels, for the reasons given above.

2.5.1 comparison between fixed and adaptive algorithm

The following section illustrates that adaptive algorithms give significantly more reliable results for slightly increased time. In the following set of experiments we use the adaptive algorithm with three predefined γ_0: 0.6, 0.75 and 0.9. Because of the granularity of the score function, conditional probabilities are not equal at each

γ_0	0.6	0.75	0.9
number of experiments	100	100	100
number of path per iteration	1000	1000	1000
number of levels (average)	22	34	65
Time in seconds (average)	14.53	16.78	20.05
mean estimate $\times 10^6$ (average)	0.78	1.14	1.58
relative standard deviation of $\tilde{\gamma}$	0.26	0.25	0.23
mean value of $\tilde{\gamma}_k$	0.55	0.68	0.83
relative standard deviation of $\tilde{\gamma}_k$	0.2	0.16	0.12

Table 2.3: Comparison between adaptive algorithms.

Statistics	Importance splitting				MC
number of experiments	100	100	100	100	1
number of path per iteration	100	200	500	1000	10 million
Time in seconds (average)	1.73	4.08	11.64	23.77	$>$ 5 hours
mean estimate $\times 10^6$ (average)	1.52	1.59	1.58	1.65	1.5
standard deviation $\times 10^6$	1.02	0.87	0.5	0.38	0.39
95%-confidence interval $\times 10^6$	[1.34; 1.74]	[1.48; 1.72]	[1.54; 1.63]	[1.64; 1.66]	[0.74; 2.26]

Table 2.4: Comparison between optimized adaptive algorithms.

iteration, but their values are kept under control because their relative standard deviation does not vanish (≤ 0.2). We use 1000 sample paths per level and repeat the experiments 100 times.

As we increased the desired γ_0, the number of levels and time increase. However, the final estimate with $\gamma_0 = 0.9$ matches the Monte Carlo estimator and the relative standard deviation is minimized. In this experiment the number of levels found adaptively is on average 65. Even with mean value of conditional probabilities smaller than in the 80-fixed-level experiment, the results show better convergence, a slightly better speed and lower standard deviation.

2.5.2 Comparison with the optimized adaptive algorithm

This section illustrates a set of experiments using the optimized adaptive algorithm. As previously, we repeated experiments 100 times to check reliability of our results. For each experiment we use a different number of initial paths: 100, 200, 500 and 1000. In order to give an idea of the gain of time, we also executed a Monte Carlo experiment using 10^7 paths. The 95%-confidence intervals are given by (2.4) for the importance splitting experiments and by the standard confidence interval $\left[\tilde{\gamma} \pm 1.96 \times \sqrt{\frac{\tilde{\gamma}(1-\tilde{\gamma})}{N}}\right]$ for Monte Carlo experiment. As the experiments are repeated several times, we approximate the relative standard deviation σ by the standard deviation of the estimates divided by the average of the estimates, instead

of assuming full independence between levels and so taking $\sigma \approx \sum_{k=1}^{m} \frac{1-\gamma_k}{\gamma_k}$. Our approach is more pessimistic and in practise requires the experiment to be repeated a few times. However, even doing so, the results are much more accurate than the Monte Carlo approach. For example, 100 initial paths are used in the first experiment. Roughly speaking, the paths cross on average 100 other levels and only 11% are rebranched each time. So, only 1200 paths are generated and provide in less than 2 seconds an estimate and a confidence interval strictly included in the Monte Carlo confidence interval. This represents a gain greater than 10^4 with respect to the Monte Carlo experiment.

Fig. 2.2 illustrates empirically the convergence of the number of levels to a Gaussian with low variance (4.23) with respect to the mean of levels (100.65). Although this fact is only empirical, knowing that the variance is low has some importance whenever the time budget is critical for more extensive experiments.

2.6 Conclusion

We have introduced the notion of using importance splitting with statistical model checking to verify rare properties. We have described how such properties must be decomposed to facilitate importance splitting and have demonstrated the procedures on several examples. We have described two importance splitting algorithms that may be constrained to give results within confidence bounds. Overall, we have shown that the application of importance splitting to statistical model checking has great potential. One future work consists conducting a deeper comparison between importance splitting and sampling, or even to try to combine the two approaches. In [CFGN09], the authors proposed to use importance sampling to retrieve the inputs of a concurrent systems. We should try to compare their work with an extension based on importance splitting. There are several other directions for future work. Due to the organization of the thesis and the dependency with other chapters, we have postponed them to the conclusion chapter.

Chapter 3

Change Detection and Statistical Model Checking

3.1 Introduction

In this chapter, we propose two new contributions to SMC. We focus on requirements that can be represented by bounded temporal properties. As we have seen in the introduction chapter, classical SMC algorithms are interested in estimating the probability to satisfy such a property starting from an initial state, which is done by monitoring a finite set of executions from this state. In this paper, we also consider the case where one can only observe the current execution of the system. In this context, we are interested in observing the evolution of the probability to satisfy the property at successive positions of the execution, and detecting positions where it drastically changes from original expectation. In summary, our first contribution is a methodology that can be used to monitor changes in probability distributions to satisfy a bounded property at runtime. Given a possibly infinite sequence of states that represents the continuous execution of the system, the algorithm monitors the property at each position and rise a flag when the proportion of satisfaction has changed significantly. The latter can be used to monitor, e.g., emergent behaviors. To achieve this objective, we adapt CUSUM [BN93], that is an algorithm that can be used to detect changes in signal monitoring. Our ambition is not to propose a new version of CUSUM, but rather to show how the algorithm can be used in the monitoring context.

Our second contribution is to show how the programming interface of Plasma Lab can be exploited in order to make SMC technology directly available in toolsets used by designers (which we will also show in another concept in Chapter 4). Our approach differs from the one adopted by other SMC/formal verification toolsets which assume the existence of formal semantics for the design language, as well as a compiling chain to the rather academic languages used by validation tool. The concept is illustrated with an integration of Plasma Lab as a library of the Simulink toolset. Concretely, we show that the recently developed Plasma Lab can directly be integrated as a Simulink library, hence offering the first in house tool

for the verification of stochastic Simulink models – this tool completes the panoply of validation toolsets already distributed with Simulink. Another advantage of our approach is that any advance on SMC that we will implement within Plasma Lab in the future will directly be available to the Simulink users.

Finally, our third contribution is to show the potential of our approach on a pig-shed case study.

Organisation of the chapter The chapter is organized as follows. In Section 2, we define the new statistical model checking problems we want to solve. Section 3 discusses solution to those problems. Section 4 discussed the integration of Plasma Lab within Simulink. Section 5 illustates our approach, while Section 6 concludes the paper.

3.2 Two new Problems

We consider a stochastic discrete event system $\mathcal{S}ys = \langle S, I, T, SV, V \rangle$ and define the two following problems.

3.2.1 The Optimization problem

We study the *optimization problem*, that is the one of finding an initial state that maximizes/minimizes the value of a given observation. Consider a set $\mathcal{O}$ of observations over $\mathcal{S}ys$. Each observation $o \in \mathcal{O}$ is a function $o : Path^n(\overline{s}) \to \mathbb{D}_o$ that associates to each run of length n and starting at $\overline{s}$ a value in a domain $\mathbb{D}_o$. We denote $(\tilde{o})_n$ the average value of $o(\omega)$ over all the executions $\omega \in Path^n(\overline{s})$. The *optimization problem* for $\mathcal{S}ys$ is to determine an initial state $\overline{s} \in I$ that minimizes or maximizes the value $(\tilde{o})_n$, for all $o \in \mathcal{O}$.

As an example, an observation can simply be the maximal value of a given parameter along an execution. The average observation then becomes the sum of those observations divided by the number of runs. In this context, the optimization could be to find the initial state that minimizes the value of the parameters.

3.2.2 Change detection problem

In this section, we consider the problem of detecting whether the probability to satisfy a given BLTL property φ changes at execution time. More precisely, we consider a (potentially infinite) execution $\omega = (s_0, t_0), (s_1, t_1), \ldots, (s_n, t_n), \ldots$ of a system $\mathcal{S}ys$. We monitor φ from each position (s_i, t_i) of this execution (the monitoring involves a finite sequence of states as BLTL formulas are time bounded) and we compute an ingenious proportion on the numbers of satisfaction and non satisfaction of the property. This proportion is used to detect changes in the probability to satisfy the property at a given point of the execution. Concretely, assuming that this probability is originally $p<k$, we detect a change index in the execution when the probability becomes $p \geq k$.

Example 3.1. Consider the firefighting services in city like London. Assume that under normal traffic conditions, the firemen can extinguish a fire within three hours with a probability greater than 0.7. It is expected that this probability decreases when the traffic increases. The challenge is to detect the time t when this change happens.

Formally, we consider a sequence of Bernoulli variables X_i such that $X_i = 1$ iff $\omega^i \models \varphi$. An execution ω satisfies a change $\tau = p \geq k$ where $p = Pr[\varphi]$ at time t, iff $Pr[X_i = 1] < k$ for $t_i < t$ and $Pr[X_i = 1] \geq k$ for $t_i \geq t$. Given an execution ω, we use $\tau!$ to denote the index $i = (s_i, t_i)$ in ω at which the execution is subject to the change. We assume an implicit change detection maximal time set by the user. If no change is detected after this time has passed, then we set up the evaluation of τ to ∞. In case the execution is subject to several changes, we take the first time. Using those notations, one can define Boolean propositions over changes and their respective time. One can also combine changes propositions with BLTL formulas, providing that those propositions are not in the scope of temporal operators. We now introduce extended BLTL change-based relations, an extension of BLTL that incorporates a change detection operator.

Definition 3.2. An extended BLTL change relation is defined as:

$$\begin{aligned} \mathit{change} &:= \quad p \star k \texttt{ where } p = Pr[\varphi] \\ \mathit{prop} &:= \quad \texttt{let } \tau = \mathit{change} \texttt{ and } \tau' = \mathit{change} \texttt{ and } \ldots \texttt{ in } \delta \\ \delta, \delta' &:= \quad \tau! \diamond \tau'! + t \mid \tau! \diamond t \mid \varphi \in \text{BLTL} \\ &\quad \mid \delta \vee \delta' \mid \delta \wedge \delta' \mid \neg\delta \mid \delta \Rightarrow \delta' \mid (\delta) \end{aligned}$$

with p is a probability identifier, $k \in]0, 1[$, $t \in \mathbb{Q}^+$, $\diamond \in \{<, \leq\}$, and φ is a BLTL formula.

This extension allows us, e.g., to express conditions such as "if a change occurs at time t, then the system shall reach a state x in less than 10 units of time". The semantics of extended BLTL change relation easily follows from the one of BLTL and the description of the change operator.

3.3 Quantitative verification and change detection: a statistical model checking approach

In this section, we detail our statistical model checking algorithmic solutions to the problems described in Section 3.2. SMC solutions to optimization problems is well-known and will only briefly be surveyed. SMC solution for extended BLTL change relations is new.

3.3.1 Optimization

We show that a simulation approach can also be used to perform an optimization of the model by varying the model parameters and evaluating the observable quantities

to optimize. We consider a stochastic state transition system $\mathcal{S}ys$, with a set of initial states I, and a set of observations $\mathcal{O}$.

For each initial state $\overline{s} \in I$ we perform N random simulations of the system $\mathcal{S}ys(\overline{s})$ and we compute the average value of the observed quantities at the end of the simulations. Therefore, for each observation $o \in \mathcal{O}$ we compute an estimation $\frac{1}{N}\sum_{i=1}^{N} o(\omega_i)$ of the average value $(\tilde{o})_n$ after runs of length n.

To solve the optimization problem, we must determine the configurations in I that optimize (minimize or maximize) these quantities. When the problem is defined with several observable quantities, we are faced with a multi-objective problem, and the best configurations are then selected by computing the Pareto frontier of the set of observations.

3.3.2 change detection with CUSUM

In this section, we consider SMC solutions for verifying extended BLTL properties with changes. We first present an SMC algorithm for change detection, and then briefly discuss the monitoring of extended BLTL. For change detection, we resort to the CUSUM algorithm [BN93], whose principles have already been formalized in other contexts [VHV08].

Assume a set of states variables SV and let $\mathcal{S}ys$ be a VSDES and $\omega = (s_0, t_0), (s_1, t_1), \ldots$ be an execution of $\mathcal{S}ys$. We consider the change $\tau = p \geq k$ `where` $p = Pr[\varphi]$ with φ a BLTL property and $k \in]0, 1[$. Let $X_1, \ldots X_N$ be a finite set of Bernoulli variables such that X_i takes the value 1 if ω satisfies φ starting at (s_i, t_i). We note p_i the probability value of X_i (contrary to the quantitative problem, this probability varies during the execution). We assume the probability initially satisfies $p_0 < k$. The problem is stated as:

- $H_0 : \forall\ 1 \leq i \leq N, p_i < k$ *i.e.* no change occurs
- $H_1 : \exists\ 1 \leq i \leq N$ such that the change occurs at time t: $\forall 1 \leq j \leq N$, we have $t_j < t \Rightarrow p_i < k$ and $t_j \geq t \Rightarrow p_i \geq k$.

Like SPRT, the CUSUM comparison is based on a likelihood-ratio test: it consists in computing the cumulative sum of the logarithm of the likelihood-ratio S_i over the sequence of samples $X_1, \ldots X_i$ and detecting the change decision as soon as S_i satisfies the stopping rule.

$$S_i = \sum_{j=1}^{i} s_j \qquad\qquad s_j = \begin{cases} \ln \frac{k}{p_j}, & \text{if } X_j = 1 \\ \ln \frac{1-k}{1-p_j}, & \text{otherwise} \end{cases}$$

The typical behavior of the log-likelihood ratio S_i is a global decreasing before the change, and an increasing shape after the change. Then the stopping rule purpose is to detect when the positive drift is sufficiently relevant to detect the change. It consists in saving $m_i = \min_{1 \leq j \leq i}\ S_j$, the minimal value of CUSUM, and compare

it with the current value. If the distance is sufficiently great, the stopping decision is taken, i.e., an alarm is raised at time $t_a = \min\{t_i : S_i - m_i \geq \lambda\}$, where λ is a sensitivity threshold.

The CUSUM proportion can only be computed during a finite amount of time, which is set by the user. In case there is no detection, we set $t_a = +\infty$. Note that we presented CUSUM monitoring for the case $p \geq k$, but it could be set up for $p \leq k$ by defining the stopping rule for the maximum value of CUSUM instead.

CUSUM Calibration: It is important to note that the likelihood-ratio test assumes that the considered samples must be independent. This assumption may be difficult to ensure over a single execution of a system, but several heuristic solutions exist to guarantee independence. One of them consists in finding a location frequently visited during the execution of the system. Collecting exactly one sample each time such a state is visited, ensures independence between samples. In our context, such a state can be the initial location from which the execution is constantly restarted. However this solution cannot be applied to continuous-time systems. Another solution is to introduce delays between the samples. In that case Monte Carlo SMC analyses can evaluate the correlation between the samples, and help to select appropriate delays.

The CUSUM sensitivity depends on the choice of the threshold λ. A smaller value increases the sensitivity, *i.e.* the false alarms rate. A false alarm is a change detection at a time when no relevant event actually occurs in the system. Conversely, big values may delay the detection of the changes. The false alarms rate of CUSUM is defined as $E[t_a]$, the expected time of an alarm raised by CUSUM while the system is still running before the change occurs. Ideally, this value must be the biggest as possible $E[t_a] \to +\infty$. The detection delay is defined as the expected time between the actual change of time t and the alarm time t_a raised by CUSUM: $E[t_a - t \mid t < t_a]$. Ideally, this value has to be small as possible. In Section 3.5, we will propose an heuristic that uses the quantitative model checking problem in order to calibrate the algorithm.

The empirical way to choose the stopping rule: One of the main difficulties in applying CUSUM is to compute the minimal duration needed to trigger an alarm. Indeed, the algorithm may be subjected to brief local changes that should not impact the final result. Theoretically, the properties of the CUSUM are based on the computation of the Average Run Length function (ARL) [BN93]. In a very few cases, this function may be computed or approximated using some approximating techniques (Wald or Siegmund) but most of the time, it is too complex to be used and to deduce λ. In this paper we propose a variant of the methodology proposed in [VHV08]. Our approach consists in exploiting $\mathcal{S}ys_0$, that is a version of the system for which the change does not occur. We first compute the probability p for this system to satisfy the property. We then compute several CUSUM on the modified system in order to compute the average frequency of a false alarm. The latter is obtained by observing the mean time between positive drift in the CUSUM as well

as its duration in term of samples (observation of the CUSUM quotient). We then compute the minimal sample duration to exceed the minimal change probability. This value is multiplied by the logarithm of the minimal change probability divided by p (i.e the minimal value of a drift).

Monitoring executions for Change Relation Satisfiability

We now briefly discuss the monitoring of extended BLTL with changes. Let us consider the change relation γ based on τ_1, ..., τ_n changes. Using the syntax introduced in Section 3.2.2, it is expressed as `let` τ_1 `and` ... `and` τ_n `in` γ, where γ contains Boolean operations over changes and BLTL formulas. We use the following monitoring procedure for each atom:

1. For each change τ_i, we set a CUSUM monitor that splits the monitoring into sub-monitors, one for each random variable, i.e., one to monitor the BLTL formula involved in the change from a given position of the execution.. Note that classical tableau-based heuristics allows us to reuse information between monitoring actions.

2. The proposition $\tau_i!$ holds iff $t_i \neq +\infty$. The proposition $\tau_i \diamond t$ holds iff $t_i \diamond t$. Similarly, the proposition $\tau_i \diamond \tau_j + t$ holds only if $t_i \diamond t_j + t$ but it is undefined if $t_i = t_j = +\infty$.

3. BLTL formulas can be monitored with classical techniques.

In practice, the tool generates monitors on demand for the given atoms and combines their answers in a Boolean manner.

3.4 Plasma lab and simulink integration

The results presented in Section 3.3 have been implemented in Plasma Lab. In this section, we now show how to integrate Plasma Lab within Simulink, hence lifting the power of our simulation approaches directly within the tool. We will focus on those Simulink models with stochastic information. The experienced reader shall observe that our approach is different from the one of Zuliani et al. [ZPC10] that consists in programming one SMC algorithm within the Matlab toolbox of Simulink. Indeed, the flexibility of our tool will allows us to incrementally add new algorithms to the toolbox without new programming efforts. Moreover, the user will directly use Plasma Lab within the Simulink interface, without third party. The approach is also different from the one in [DDL+12] that consists in translating parts of Simulink models into the Uppaal language (which makes it difficult for analysing counter examples). The reader shall observe that Plasma Lab for Simulink offers the first integrated verification tool for Simulink models with stochastic information.

Simulink is a block diagram environment for multidomain simulation and Model-Based Design approach. It supports the design and simulation at the system level, automatic code generation, and the testing and verification of embedded systems.

Simulink provides a graphical editor, a customizable set of block libraries and solvers for modeling and simulation of dynamic systems. It is integrated within MATLAB. The Simulink models we considered have special extensions to randomly behave like failures. By default the Simulink library provides some random generators that are not compatible with statistical model checking: they always generate the same random sequence of values at each execution. To overcome this limitation we use some C-function blocks call that generate independent sequences of random draws.

Our objective was to integrate Plasma Lab as a new Simulink library. For doing so, we developed a new simulator plugin. One of the key point of our integration has been to exploit MATLAB Control[1], a library that allows to interact with MATLAB from Java. This library uses a proxy object connected to a MATLAB session. MATLAB invokes, *e.g.* functions `eval`, `feval` ... as well as variables access, that are transmitted and executed on the MATLAB session through the proxy. This allowed us to implement the features of a model component, controlling a Simulink simulation, in MATLAB language. Calls to this implementation are then done in Java from the Plasma Lab plugin.

Regarding the monitoring of properties, we exploit the simulation output of Simulink. More precisely, BLTL properties are checked over the executions of a SDES, *i.e.* sequences of states and time stamps based on the set of state variables SV. This set must be defined by declaring in Simulink signals as log output. During the simulation these signals are logged in a data structure containing time stamps and are then retrieved as states in Plasma Lab. One important point is that Simulink discretizes the signals trace, its sample frequency being parameterized by each block. In terms of monitoring this means that the sample frequency must be configured to observe any relevant change in the model. In practice, the frequency can be set as a constant value, or, if the model mixes both continuous data flow and state flow, the frequency can be aligned on the transitions, *i.e.* when a state is newly visited.

3.5 A Pig Shed Case study

We illustrate the contributions of this paper on the model of a temperature controller in a pig shed. This model is inspired by similar studies [JRLD07, GTJ+10, DDL+13]. The system under control is a pig shed equipped with a fan and a heater to regulate the air temperature. Air temperature in the shed is subjected to random variations due to the variation of external temperature and the variation of the number of pigs that produce heat. The objective of the controller is to counter these variations such that the temperature remains within a given comfort zone. To do so, the controller can activate the heater to increase the temperature, and the fan to bring external air and therefore cool the shed. Then the temperature T of the shed is given by the following differential equation:

$$T' = T_{ext} * Q - T * Q + W_{heater} + W_{pigs}$$

[1] https://code.google.com/p/matlabcontrol/

where T_{ext} is the external temperature, $Q = Q_{min} + Q_{fan}$ is the air flow created by a minimal flow Q_{min}, and an additional flow Q_{fan} when the fan is activated, W_{heater} is the heat produced by the heater, when activated, and W_{pigs} is the heat produced by the pigs.

The controller that we study applies a *bang-bang* (also called *on-off*) strategy that is specified by four temperature thresholds, that is (1) when the temperature goes above TFanOn, the fan is turned on, (2) when the temperature returns below $\mathsf{TFanOff}$, the fan is turned off, (3) when the temperature goes below $\mathsf{THeaterOn}$, the heater is turned on, (4) when the temperature returns above $\mathsf{THeaterOff}$, the heater is turned off.

The fan and the heater are subjected to random failures when they are in use. Exponential distributions control the occurrence time of a failure. After a failure a reparation process allows to restart the fan or the heater, but it also takes a random time, exponentially distributed. Additionally, the failure rate increases with usage due to wear and tear. This continues until a replacement is performed, which resets the rate.

3.5.1 Quantitative Verification and Optimization

The controller goal is to maintain the temperature within a comfort zone specified by a minimum and a maximum temperature (resp. $\mathsf{T_{min}} = 15\,^{\circ}\mathrm{C}$ and $\mathsf{T_{max}} = 25\,^{\circ}\mathrm{C}$). We first consider the following values for the controller thresholds: $\mathsf{TFanOn} = 22\,^{\circ}\mathrm{C}$, $\mathsf{TFanOff} = 20\,^{\circ}\mathrm{C}$, $\mathsf{THeaterOn} = 18\,^{\circ}\mathrm{C}$ and $\mathsf{THeaterOff} = 20\,^{\circ}\mathrm{C}$.

We apply statistical model-checking to evaluate the efficiency of the controller both in the presence and absence of failures. The first BLTL property that we monitor checks that the system is not in discomfort for an excessive period of time. This is expressed by the following property:

$$\Phi_1 = G^{\leq t_1} F^{\leq t_2} \neg \mathsf{Discomfort}$$

where t_1 is the simulation time, t_2 is the accepted discomfort time, and $\mathsf{Discomfort}$ is a predicate that is true when the temperature of the system is outside the comfort zone. A dual of Φ_1 is to check for long periods without discomfort. This is possible with:

$$\Phi_2 = F^{\leq t_1} G^{\leq t_2} \mathsf{Discomfort}$$

Finally, a third BLTL property checks that each period of discomfort is followed by a period without discomfort:

$$\Phi_3 = G^{\leq t_1} \Big(G^{\leq t_2} \mathsf{Discomfort} \Rightarrow F^{\leq t_3} (G^{\leq t_4} \neg \mathsf{Discomfort}) \Big)$$

Here t_1 and t_2 are as previously, while $t_3 \geq t_2$ is the expected time at which the system returns to normal situation, and t_4 is the duration of the period without discomfort.

We use Plasma Lab to estimate the probability to satisfy these properties for different values of the timing constraints, on both models with and without failures.

Each property is evaluated over a period of time $t_1 = 12000$ $t.u.$ with precision $\epsilon = 0.01$ and confidence $\delta = 0.01$. Φ_1 and Φ_2 are evaluated for several values of t_2. Note that for $t_2 = 0$, Φ_1 resumes to checking $G_{\leq t_1} \neg$Discomfort. Φ_3 is evaluated with $t_2 = 25$ $t.u.$ and several values of t_3 and t_4.

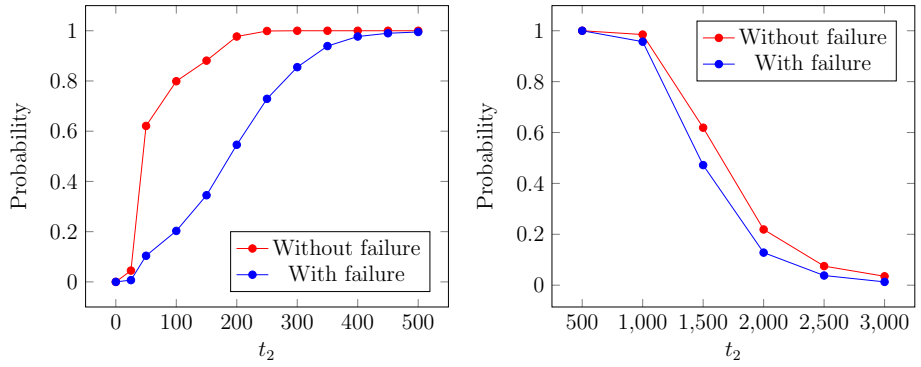


Figure 3.1: Probability estimation with SMC of satisfying Φ_1 (left) and Φ_2 (right)

The results for properties Φ_1 and Φ_2 are presented in Fig. 3.1. While the probabilities of satisfying Φ_1 show a significant difference between the models with and without failures, the results for Φ_2 are almost identical. This means that discomfort is as frequent in the two models, but it tends to last longer in the presence of failures. The results for Φ_3 are presented in Fig. 3.2. It shows again that the model without failures recovers quicker from a discomfort period.

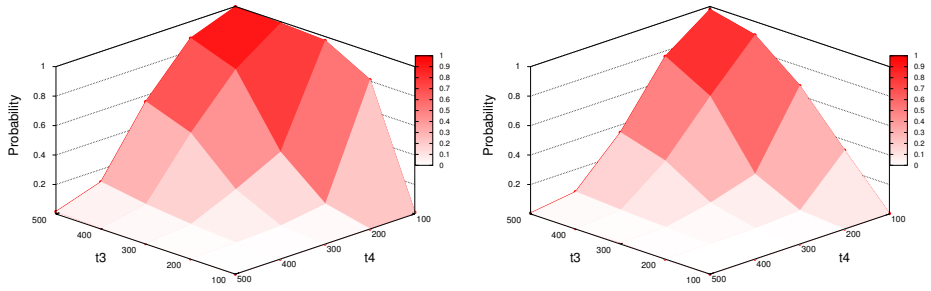


Figure 3.2: Probability estimation with SMC of satisfying Φ_3 without failures (left) and with failures (right)

Instead of estimating a probability using SMC techniques, we can compute the average value of two quantities in the model, namely the *discomfort time*, that is the cumulative time when the model is in a discomfort state, and the energy *cost*, computed with the duration of use of the heater and the fan. We aim at minimizing these two values by choosing adequate values of the model parameters.

Using Plasma Lab we can automatically instantiate the model with a range of values for the four temperature thresholds. We specify the ranges $[15, 20]$ for THeaterOn and THeaterOff, and $[20, 25]$ for TFanOn and TFanOff, with an increment

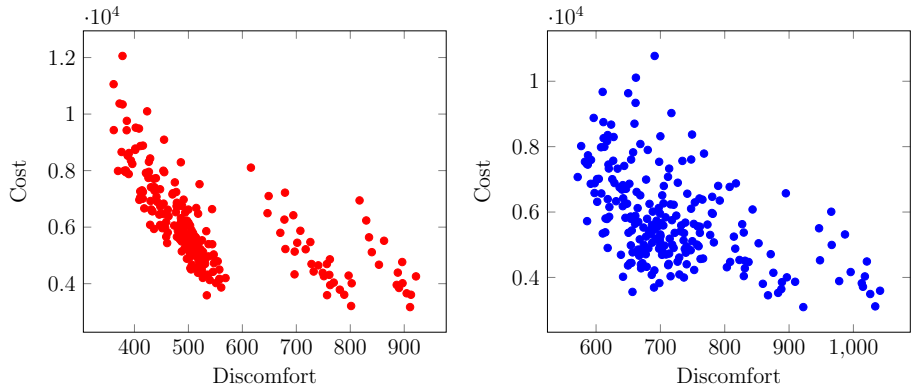


Figure 3.3: Optimization of the thresholds parameters without failures (left) and with failures (right)

of 1. We additionally specify the following constraints to select a subset of the possible values of the parameters: TFanOff $<$ TFanOn, THeaterOn $<$ THeaterOff, and THeaterOn $<$ TFanOn.

Using these constraints Plasma Lab generates a set of 225 possible configurations, for each variant of the models, with and without failures. Each configuration is automatically analyzed with 100 simulations. We then plot the average values of the cost and the discomfort in Fig. 3.3. These graphs helps to select the best values of the parameters by looking at the points that lie on the Pareto frontier of the data.

3.5.2 Change Detection: Detection and Calibration

In our pig shed, the equipment may sometimes fail (heater or fan may break). In such situation, the shed may be too frequently in the discomfort zone, which may lead to the death of several pigs.

As we have seen, the probability of being in the discomfort zone is nominally very low. However, to avoid problems, one should be able to rise a flag as soon as the probability to be in the discomfort zones crosses a given threshold. Our objective is to detect that when such a change happens, there is a maintenance procedure that moves the shed out of the discomfort zone. In our example, this maintenance feature is modeled as a procedure that is regularly applied to the pig shed. Initially, the time between each maintenance is set to a very large value (500000 $t.u.$). The final objective is to set this time value in order to have an acceptable maintenance delay when the death risk is to heavy for the pigs (emergent behavior). This will be done by detecting changes.

We modeled the property using the change property language we proposed and we used the CUSUM algorithm to check it. We first define τ to be the following change: " the probability to be in the discomfort zone more than $t_1 = 100$ $t.u.$ is greater than 0.35". We are now ready to propose a property that expresses that when the change occurs, then the maintenance must be done in less than $t_2 = 1000$ $t.u.$ Formally,

$$\varphi_4 = \left| \begin{array}{l} \texttt{let} \quad \tau = \ p \geq 0.35 \quad \texttt{where} \\ \quad p \ \texttt{is} \ Pr\big[G^{\leq t_1}\mathsf{Discomfort}\big] \\ \texttt{in} \quad \tau! \Rightarrow F^{\leq \tau + t_2}\mathsf{Reparation} \end{array} \right.$$

In order to perform the analysis, the CUSUM algorithm needs a calibration step. We first require an estimate of p_0, the initial probability of being in the discomfort zone before the change occurs, and we determine a minimum delay between the samples that ensures independence between the analyses. We disable failures of the temperature regulation system (fans + heaters) in the shed model and we estimate p_0 using a Monte Carlo simulation based on the monitoring of $G_{\leq t_1}\mathsf{Discomfort}$. The results differ with sampling delays lower than 100 $t.u.$, which indicates a correlation, but converge to 0 with 150 $t.u.$ and 200 $t.u.$. Therefore the property is checked every 200 $t.u.$ over execution traces of length 21000 $t.u.$ After 630 sec. of analysis, Plasma Lab returns $p_0 \in [0, 0.05]$ with a confidence of 0.9.

Next step is to set the stopping sensitivity λ, which is again done with a Monte Carlo approach. The objective is to observe several samples (value of the ratio) for several CUSUM. When there is no failure, the curve of samples should decrease. Indeed, it should only increase when failures happen, i.e., when the change happens. In practice, even without failure, the curve may locally increase for a short amount of time, which is due to the uncertainty introduced in the model. The objective is to characterize those local drifts to avoid false alarms.

To do so, we ran 100 executions of the CUSUM and observed 2000 samples (values of the quotient) per CUSUM (which corresponds to 201000 $t.u.$). From those experiments, we observed that the mean time (in CUSUM samples) between positive drifts is 127.88 $t.u.$ and the mean duration of positive drift is 1.2 samples. The frequency of positive drifts is thus $1.2/(127.88 + 1.2)$, which is in the interval $[0, 0.05]$ as predicted by Monte Carlo algorithm. In order to observe a real alarm one needs to push this quotient to 0.35, which is the probability one wants to observe. This amounts to varying the duration of a positive sample, i.e., to replace 1.2 by a higher value in the above quotient. Doing so, we concluded that the probability will become greater than 0.35 when the positive drift is longer that 52 samples. From the definition of CUSUM, we compute that the drift is $\ln \frac{0.35}{0.05}$ for each positive sample. We finally set the stopping rule to $\lambda = 52 * \ln \frac{0.35}{0.05} \approx 101$.

We then launched the CUSUM on the model with failures over an execution of 210000 $t.u.$ for the property $G_{\leq t_1}\mathsf{Discomfort}$ checked every 200 $t.u.$. We applied Plasma Lab and observed that the stopping rule was satisfied after the sample 901 that corresponds to the simulation time 103473.34 $t.u.$ We reproduced the experiment several times (20): we observed the change occurred at 102543.23 $t.u.$ in average and in earlier $\approx$ 101000 $t.u.$ We conclude that to satisfy Property φ_4 the maintenance operation must be scheduled at 100000 $t.u.$

3.6 Conclusion

This chapter presents two modest contributions to SMC. The first contribution takes the form of an algorithm used to detect changes on the probability to satisfy a bounded property at runtime. The second contribution illustrates the power of Plasma Lab via a Simulink library integration. This integration constitutes one of the first proof of concept that SMC can indeed be integrated as feature library in a tool largely used in industry. Other integration of Plasma will be introduced in the next chapter.

Future work includes an extension of the power of distributed computing to Plasma-Simulink. The latter is technically challenging as it would require to duplicate compiled code to avoid license duplication and costs.

Chapter 4

Motion planning in crowds using statistical model checking to enhance the social force model

This chapter presents an application of SMC to solve a planing problem. The experience reported in this chapter took place within the framework of the European project DALI.

4.1 Introduction

With unimpaired ability, pedestrians are able to negotiate crowded areas with few problems. With reduced ability or under panic conditions [HFV00b], finding a good strategy to proceed can be challenging. As a result, people afflicted by a decline in physical or cognitive abilities can be discouraged from attending crowded places, with a consequent negative impact on their physical condition (reduced exercise), on the quality of their nutrition (reduced fresh food) and on their psychological wellbeing (reduced social contact). Motivated by these considerations, the DALi project [DAL] aims to devise an intelligent 'walker' (an assistive wheeled device) that detects the presence of other pedestrians in the environment, anticipates their intent and plans an appropriate path that is suggested to the user via a combination of audio, visual and haptic interfaces. In this chapter we present an efficient algorithm that employs advanced modelling and verification techniques to address the path planning problem in a crowded and unfamiliar environment.

Succinctly, the problem is one of devising an online motion planning algorithm for an autonomous agent (the user) in a dynamic environment. The position of most fixed objects (e.g., buildings and rooms) are known a priori, but the algorithm must account for the possibility of changes, such as temporary obstructions. The environment contains moving objects (i.e., other pedestrians), whose positions and velocities cannot be known before they are encountered. The overall goal is to allow the user to visit pre-defined locations in the environment, while avoiding collisions, crowding and delays. The output of the algorithm is a *suggested* trajectory, so the

algorithm must be reactive to the potentially uncooperative response of the user. Practically, the algorithm will be implemented in a low power embedded computing device and must be sufficiently efficient to make course corrections in a time of the order of seconds. This time scale is dictated by the typical velocities of pedestrians and by the fact that frequent readings help to reduce the random errors produced by sensors.

Planning the trajectory of a human in a populated environment faces several challenges. One is the complex and continuous nature of the dynamics of human motion. A second is characterising the high level "logic" underlying the goals of different people and their interactions. A third challenge is the unpredictability of *individual* human behaviours. A further practical challenge arises from the need to account for variability in the environment and in the visibility cone of the sensors: in a crowded environment (where occlusions are frequent): the planning algorithm must be executed at least once every second to be reliable. Moreover, the requirements of the project mandate the use of low power computing devices emphasising the importance of computational efficiency.

Typically, a high level goal may be formulated in natural language as follows: *go toward a point of interest in the minimum possible time* and *remain at a safe distance from any other person* and *if somebody is heading in a direction that crosses your path, give way only if she is an old lady.* If the user moves to a densely crowded environment, no path might exist that is totally safe with respect to this goal. In such a case, the user has to decide whether to take a risk and move along a trajectory where some accident could occur, balanced against the expectation that other people will move out of the way as a result of social rules. Despite such expectation, some pedestrians may not respect social conventions or for other reasons just seem to move randomly. Overall, the decisions are complex and the acceptable level of risk is dependent on the urgency of the user's objectives and the user's level of stress.

While the behaviour of individual pedestrians may be arbitrary, people nevertheless tend to respect certain social rules that can be formalised. Hence, o Our solution to the problem outlined above this challenging problem is a two-tiered algorithm, comprising a low level predictive mathematical model of pedestrian dynamics, managed by a statistical model checking engine that checks temporal logical properties expressing the high level goals and constraints of the user. Given a pedestrian environment and a user with a number of places to visit, our algorithm uses the location of fixed objects and areas with known high probability of crowding to plan an optimal trajectory (the global plan). In this context, an "optimal" trajectory is one that enables the user to proceed with maximum efficiency and minimal stress (avoiding, e.g., collisions or being trapped). The algorithm uses dynamic input from sensors to reconstruct the user's position from fixed objects and to account is aware of the user's current position and, once the user embarks, the algorithm uses dynamic input from sensors to create a local plan that achieves the objectives and avoids problems (e.g., collisions). The sensor data is necessary to account for non-fixed objects, such as other pedestrians and temporary obstructions. The plan is periodically re-computed to allow for significant deviations of the user from the

plan previously suggested. (as in the case of GPS navigation systems).

To make optimal progress it is necessary to be pre-emptive in avoiding problems - to be *pro*active rather than *re*active. A reactive approach, such as "move forward along the suggested trajectory and slow down if there is an obstruction", is conceptually simple, but sub-optimal. The required sensor information and computation are minimal, but this strategy is not adaptive to changing environmental conditions and may result in the user's objectives not being met. A better approach is to choose a trajectory that avoids the *predicted* trajectories of other pedestrians, given that they will also be reacting to the behaviour of the user. Since the behaviour of real pedestrians (including the user) is not entirely predictable, it is necessary to include both predictive and reactive elements in the algorithm. Moreover, in order to be effective, the algorithm must operate efficiently on a compact portable computing device (an *embedded system*).

Our algorithm provides good coverage of the requirements Moreover, our 'hardware in the loop' implementation demonstrates that (1) the algorithm is indeed implementable within the strict constraints imposed by the choice of an embedded hardware platform, and (2) Plasma Lab can be customized on demand.

Organisation of the chapter The rest of the paper is organised as follows. Section 4.2.1 gives an overview of our approach. Section 4.2.2 introduces our mathematical model in detail. Section 4.3 gives a detailed description of our algorithm and implementation, while Section 4.4 describes the results of a number of experiments that demonstrate the utility of our approach. Section 4.5 discusses our choices and highlights areas of ongoing development.

4.2 Preliminaries

4.2.1 Overview of the approach

Fig. 4.1 gives a high level overview of the algorithm. At each iterative step the algorithm acquires the state of the system, comprising the position of static objects and the position and velocity of the user and of other people in the environment.

Given the current state, the algorithm hypothesises alternative courses of action using the *social force model*. Each hypothesised trajectory is formally verified (model-checked) against properties that express goals and constraints required for the user's trajectory (i.e., where the user wants to go, obeying the appropriate social rules). This leads to a statistical distribution of potentially successful trajectories. The algorithm uses this distribution to choose an immediate action that maximises the probability of achieving the user's objectives and minimises the probability of problems. In this probabilistic context, the measurement noise is considered as an additional source of stochasticity.

The social force model may be programmed with the user's objectives and is an efficient way to describe the continuous interactions that allow pedestrians to avoid collisions. The model also includes stochasticity to model the typical unpredictabil-

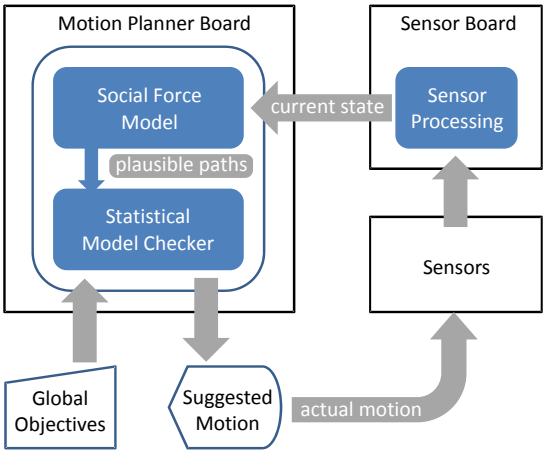


Figure 4.1: Diagrammatic overview of the motion planning framework. The sensor board detects the current state of objects in the environment. This state is used by the social force model to generate plausible future paths of the user and other pedestrians. The distribution of paths is verified against the global objectives of the user in order to suggest an optimal course.

ity of human behaviour. We use the stochasticity to generate a random sample of possible futures and choose the course of action that maximises the probability of success. Such trajectories respect the basic social and physical laws of pedestrian interactions and include the possibility of unpredicted behaviour. Their *distribution* allows the algorithm to choose a course of action that maximises the probability of success.

The stochasticity, while realistic, places an upper bound on the predictive accuracy of the model. Moreover, the model alone cannot account for the overall "mission" of the user. The predictive model needs to be managed *reactively*. Fortunately, the field of statistical model checking (SMC) encapsulates the technologies that we require to do this. SMC provides efficient algorithms to verify hypothesised trajectories against the user's constraints and objectives expressed in temporal logic. SMC can estimate the probability of success and bound the error of the estimation.

The key elements of our approach are (*i*) the social force model to hypothesise trajectories that respect low level social and physical "forces"; (*ii*) temporal logic to express the high level goals of the user and (*iii*) a statistical model checker to verify the traces with respect to the goals.

4.2.2 The social force model

The social force model [HM95, HFV00a, HFV00b, HFMV02] combines real and psychological forces to predict the behaviour of pedestrians in crowds, under normal and panic situations. The model recognises that pedestrians are constrained by the physical laws of motion and also by social 'laws' that can be modelled by external forces. The model considers an environment comprising fixed objects (walls)

and moving agents (pedestrians) that respond to attractive and repulsive forces, originated by social and physical interactions.

The model is constructed in two dimensions [HM95,HFV00a,HFV00b,HFMV02], with agents represented as circular discs. In what follows we adopt the convention of denoting vectors in bold type. Thus, agent i has mass m_i centred at position $\mathbf{x}_i \in \mathbb{R}^2$ in the environment, radius r_i and velocity $\mathbf{v}_i \in \mathbb{R}^2$. The linear model for the i-th agent is given by

$$\begin{cases} \dot{\mathbf{x}}_i = \mathbf{v}_i \\ \dot{\mathbf{v}}_i = \dfrac{\mathbf{v}_i^0 - \mathbf{v}_i}{\tau_i} + \dfrac{\mathbf{f}_i + \boldsymbol{\xi}_i}{m_i} \end{cases} \tag{4.1}$$

$\mathbf{v}_i^0$ is the *driving (desired) velocity* of agent i, represented by a product of speed amplitude v_i^0 and normalised direction $\mathbf{e}_i^0$, which is given by the direction of the line joining the initial and desired configurations. τ_i is the time taken to react to the difference between desired and actual velocity, while $\boldsymbol{\xi}_i$ is a *noise term* (a random variable) that models random fluctuations not accounted for by the deterministic part of the model. The inclusion of the noise term makes the model stochastic, such that a different trajectory is generated each time (4.1) is solved. This allows the application of SMC and serves to avoid deadlocks that might arise if, by chance, some of the deterministic forces are equal and opposite.

$\mathbf{f}_i$ is the force acting on agent i resulting from other objects in the environment and, hence, given by

$$\mathbf{f}_i = \sum_{j \neq i} [\mathbf{f}_{ij}^{\text{soc}} + \mathbf{f}_{ij}^{\text{att}} + \mathbf{f}_{ij}^{\text{ph}}] + \sum_{b} [\mathbf{f}_{ib}^{\text{soc}} + \mathbf{f}_{ib}^{\text{ph}}] + \sum_{c} \mathbf{f}_{ic}^{\text{att}}. \tag{4.2}$$

The first term on the right-hand side of (4.2) includes all the forces on agent i resulting from interactions with other agents: $\mathbf{f}_{ij}^{\text{soc}}$ is the repulsive social force that inhibits agents getting too close, $\mathbf{f}_{ij}^{\text{att}}$ is the attractive social force that brings friends together, $\mathbf{f}_{ij}^{\text{ph}}$ is the physical force that exists when two agents touch. The second summation includes the forces acting on agent i as a result of the boundaries (walls): $\mathbf{f}_{ib}^{\text{soc}}$ is the social force that inhibits agent i from getting too close to boundaries, $\mathbf{f}_{ib}^{\text{ph}}$ is the physical force that exists when agent i touches boundary b. Finally, $\mathbf{f}_{ic}^{\text{att}}$ is the attractive social force that draws agent i towards fixed objects of incidental interest (shops, cafs, toilets, etc.).

In general, the force acting on any agent is calculated with respect to the distance between its centre of mass and all other visible objects. Since the model mixes both notional (social) and real forces, the mass m_i is notionally the real mass of agent i. Other parameters can be used to model the unique characteristics of individual agents. For example, the latency factor τ_i can be used to model the possibly reduced mobility of agent i. Full details of these and other parameters can be found in [HFV00b]. In [CFG+13] we show how the model may be parametrised from captured motion.

4.3 SMC–based Motion Planner

Our motion planner is based on the scheme depicted in Fig. 4.1 and Algorithm 4. The planner assumes the existence of a pre-calculated *global plan* (GlobalPlan) that visits the user's objectives in an a priori optimal way, that is, considering all things known in advance. Typically, the global plan is computed with respect to a map of the static objects in the environment, the user's objectives and predicted anomalies (e.g., known crowded areas). Any contradiction of the a priori assumptions (e.g., an unforeseen blockage) triggers a recalculation of the global plan. We do not describe this recalculation in the present work.

The sensor board provides the current state of the local environment, located with respect to the global plan: the position and velocity of the user (state_{user}); the positions and velocities of other pedestrians ($\text{state}_{ped_1}, \text{state}_{ped_2}, \ldots$); the position of static objects (Map). The algorithm calculates a local way point $\mathbf{w}$, which is the user's point of greatest straight line progress along the global plan within the sensor range. $\mathbf{w}$ is used to calculate the user's driving velocity $\mathbf{v}^0$, assuming a constant desired speed. The driving velocities of the other pedestrians are estimated from their current velocities.

The algorithm uses the above information to construct social force models (4.1) of the local environment. Specific characteristics (e.g., τ_i) of other pedestrians are unknown to the algorithm, so it assumes the default values given in [HFV00b]. In the current implementation we construct the noise term $\boldsymbol{\xi}_i$ from two normal distributions; one for the magnitude and one for the direction. The results presented here are based on heuristically estimated parametrisations of these distributions, which appear to be adequate. In [CFG+13] we present a way of obtaining better parameters from captured motion. This topic is not the subject of the present thesis.

Algorithm 4: The planning algorithm

```
1  FindLocalPath(state_user, state_ped1, state_ped2, ..., Map, GlobalPlan, Formula, N)
2  Real P_curr, d_curr, P_best, d_best;
3  [P_best, d_best]=[0,∞];
4  for α_curr ∈ {0, ±25, ±50, ±75, ±90} do
5  |   [P_curr, d_curr] = SMC(N, Formula);
6  |   if is_better([P_curr, d_curr], [P_best, d_best]) then
7  |   |   α_best = α_curr;
8  |   |   [P_best, d_best] = [P_curr, d_curr];
9  |   end
10 end
11 if P_best == 0 then
12 |   return STOP;
13 else
14 |   return α_best;
15 end
```

The motion planner assumes the user will follow the global plan, but need to temporarily deviate to avoid collisions. The output of the algorithm is a suggested deviation, α_{best}, in the range ± 90 degrees relative to the user's direct path to $\mathbf{w}$. To

find α_{best}, the algorithm constructs models for each hypothesised deviation in the set $\{0, \pm 25, \pm 50, \pm 75, \pm 90\}$. These values are chosen to span ± 75 degrees using a tractable number of different values, with ± 90 included in case the user needs to sidestep an obstacle (see [CFG+13]). Each model is then investigates using statistical model checking.

The algorithm sets $\alpha_{curr} \in \{0, \pm 25, \pm 50, \pm 75, \pm 90\}$ and calls function SMC with arguments N and Formula. SMC estimates the probability of success P_{curr} for a particlar deviation α_{curr} by the proportion of N simulation traces that satisfy the BLTL property Formula. The value of α_{curr} is used as the *intial* deviation: the user's driving velocity is initially rotated by α_{curr}, but at each successive step of the simulation the deviation from a direct path to $\mathbf{w}$ is reduced to zero. This ensures that the user will eventually be close to the global plan.

BLTL is expressive enough to define complex sequences of high and low level requirements. For the results presented here, Formula merely expresses the basic constraints of the user:

$$\begin{aligned}(\mathrm{G}^{[0,T_{horizon}]} \bigwedge_{i \neq u} \|\mathbf{x}_u - \mathbf{x}_i\| > 0.5) \wedge \\ (\mathrm{F}^{[0,T_{horizon}]} \|\mathbf{x}_u - \mathbf{w}\| < 0.2)\end{aligned} \tag{4.3}$$

$\mathbf{x}_u$ denotes the position of the user and $\|\cdot\|$ denotes Euclidean distance. Intuitively, (4.3) means that "in the next $T_{horizon}$ time units the user will get no closer than 0.5m to any other pedestrian and will eventually be less than 0.2m from the global plan".

$T_{horizon}$ is chosen to be the expected time for the user to walk a distance equivalent to the range of the sensors. Using a higher value might produce impossible trajectories that pass through unseen fixed objects; using a lower value might exclude possible collisions. In our implementation we use $T_{horizon} = 4$s.

For each hypothesised deviation α_{curr}, the SMC function returns the probabillity of success P_{curr} and the expected distance from the global plan, d_{curr}. These are used by function is_better to decide α_{best}. is_better chooses the smallest $\mid \alpha_{curr} \mid$ which maximises P_{curr}. Ties are resolved by choosing the $alpha_{curr}$ with smallest d_{curr} or randomly if d_{curr} also ties. If $P_{best} == 0$ the user is required to stop (the global plan will be recalculated).

$T_{decision}$ is the actaul time the algorithm takes to make its predictions and must be less than the time period it is predicting, i.e., $T_{horizon}$. In practice $T_{decision}$ is bounded below by the performance of the hardware, the complexity of the environment (fixed and moving objects) and the confidence required (controlled by the number of simulations, N). In our implementation, $T_{decision} \approx 1s$.

At each decision point α_{best} is suggested to the user. The user may ignore this suggestion and move in a different direction, but the operation of the algorithm in the next decision period remains the same: α_{best} is calculated according to the global plan and the actual positions and velocities of the user and other pedestrians. Since the user specifies the global plan, when generating hypothesised traces the algorithm assumes that the user is compliant, however $\boldsymbol{\xi}_{user}$ may be used to model a lack of

compliance. The present work does not consider how the user's non-compliance might affect predictions.

Given an accurate stochastic model of the behaviour of pedestrians, the Chernoff bound [Che52] predicts that with $N = 10$ simulation runs the estimate of the probability of success has a maximum error of ± 0.3 with probability 0.7. With $N = 50$ the probability of success has a maximum error of ± 0.2 with probability 0.90. In general, the statistical confidence of the estimate increases with increasing N, but this only increases the probability of choosing the correct α_{best}. The predictive power of the model is bounded by its stochasticty. Thus, given finite computational power, we choose a value of N that balances the reactive and predictive aspects of the algorithm. That is, we choose a value of N that allows us to make $T_{decision}$ sufficiently small.

The algorithm solves (4.1) using a standard ODE solver [AM11], which produces traces comprising a sequence of states at discrete time points. Since the model given in Section 4.2.2 is based on continuous time and space, to guarantee properties that rely on the distance between objects it is necessary to choose time points that are sufficiently close. This is achieved by the ODE solver using adaptive time steps.

Simulating the traces accounts for most of the computational cost of the algorithm. We have found our chosen ODE solver to be efficient and presume its performance scales in a standard way with respect to the number of visible moving agents M and the complexity of their interactions. Since the forces in the model are dependent on the distances between agents, there is an additional $\mathcal{O}(M^2)$ cost, however M is bounded by the range of the sensors.

4.4 Simulations

To demonstrate our algorithm we have implemented a prototype on a off-the-shelf, low power embedded system, the Beagleboard xM[1]. It is a portable device that may run from battery power and provides performance comparable to a small computer. We use PLASMA-lab [INR12] as the statistical model checking library. To test the algorithm we have created a virtual environment that evolves according to the Social Force Model and contains fixed objects and other pedestrians that react to the user's presence.

The pedestrians are assigned individual global plans to simulate their objectives and individual parameters that reflect the variation seen in reality. The values of the parameters are based on the ones estimated in [HFV00b] and two different but correlated sets, one for the planner and one for the virtual environment, have been defined in order to increase the sense of reality. The noise term ξ has been differentiated as well, the standard deviation of the two normal distributions in the planner has been set as the double of the one in the virtual environment.

In this way we simulate pedestrians that are reactive to the user and each other, with behaviour that is realistically unpredictable. Moreover, the simulated device

[1] http://www.beagleboard.org

has limited omnidirectional sensing range, we suppose it is able to detect agents moving within a radius of 4 meters with respect to the current position of the user. In the final application, a sensor board connected to the single board computer will provide the real (estimated) positions and velocities of the user and nearby pedestrians.

We compared three different strategies:

- SMC with the Social Force Model ($SMC+SFM$): our novel approach, where Algorithm 4 computes the local plan. When detected, an agent is supposed to evolve according to the Social Force Model.
- SMC with a linear motion model ($SMC + LIN$): similar to $SMC + SFM$ but agents evolve according to a different and simpler model. When detected, an agent is suposed to keep moving with same speed and same direction.
- Social Force Model only (SFM): we analyze the evolution of the environment without any decision points ($T_{decision} = \infty$).

For $SMC+SFM$ and $SMC+LIN$ we use the temporal logic formula defined by Eq. (4.3). We also used the following parameters: $T_{horizon} = \{1, 2, 4, 6, 8\}$, $T_{decision} = 1$ and $N = 50$. We performed 500 independent runs for SFM and 500 for every combination of $T_{horizon}$ for $SMC + SFM$ and $SMC + LIN$. Our objective was to demonstrate that 1) the higher complexity of our approach leads to valuable payoff in terms of performance and 2) it can be implemented online on an embedded device with limited computing power.

Algorithm performance analysis We have devised two scenarios that challenge our algorithm and highlight significant features of its performance. In the first one (namely, *scenario 1*, depicted in Fig. 4.2(a)) the user moves on a straight line close to a fixed obstacle, while two agents are moving towards him following a straight line as well. In the second one (namely, *scenario 2*, showed in Fig. 4.2(b)) the user attempts to visit a market stall at the end of the market while some pedestrians (Agent 1-6) block the user's progress by entering the scenario and moving from one market stall to another. The user's global plan is a straight line from the left to the right of the market. Fig. 4.3 depicts the distances over time with respect to the user, respectively, for one particular run of *scenario 2*.

We defined 4 indicators to measure performance: 1) the time needed for the user to reach the right side of the scenario (T_{exit}), 2) the measured probability of respecting the minimum safety distance to agents (P_{safe}), 3) the average deviation in position from the global plan (ϵ_x) and 4) the average deviation of the orientation of the user with respect to the ideal orientation of a user perfectly following the global plan (ϵ_θ). These indicators are formally defined as follows.

Let $x(t)$ represent the cartesian coordinates of the position of the user after the planning for each time t, $\theta(t)$ represent its orientation with respect to a fixed frame, $\tilde{x}(t)$ the long term plan and $\tilde{\theta}(t)$ the orientation decided according to the long term plan. The integral error of the difference between the corrected plan and the global

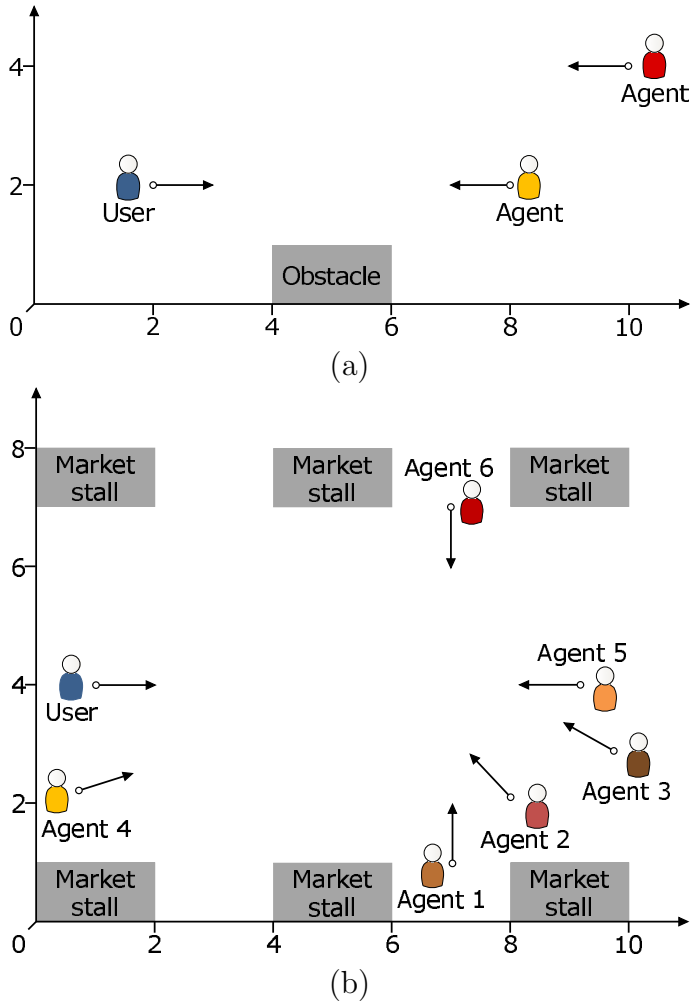


Figure 4.2: Scenarios used to test the algorithm, *scenario 1* (a) and, *scenario 2* (b).

plan can be defined as $\epsilon_x = E\left\{\sqrt{\frac{1}{T}\int_0^T |x(t) - \tilde{x}(t)|^2 \, dt}\right\}$. A similar performance indicator ϵ_θ is defined for the orientation $\theta(t)$.

Indicators ϵ_θ and P_{safe} can be used to quantify the "comfort" of the user. Indeed, frequent changes in the direction reduce the user experience, especially if elderly, and so does the probability of accidents. Table 4.1 and Table 4.2 reports the performance we obtained for *scenario 1* and *scenario 2*, respectively, using different values for $T_{horizon}$.

Scenario 1 is the most problematic for SFM due to the limitations of this model we discussed in [CFG+13]. The SMC-based strategies exhibit a higher P_{safe} and a lower ϵ_θ when $T_{horizon} \geq 6$. $SMC + SFM$, in turn, outperform $SMC + LIN$ on all indicators.

In *scenario 2*, from the safety and comfort point of view of the user, $SMC + SFM$ approach obtains a higher P_{safe} and a lower ϵ_θ with respect to SFM and

Table 4.1: Scenario 1: performance for $SMC + SFM$, $SMC + LIN$ and SFM strategies. 500 simulations each were conducted.

	Unit	$SMC + SFM$					$SMC + LIN$					SFM
$T_{horizon}$	[s]	1	2	4	6	8	1	2	4	6	8	-
T_{exit}	[s]	23.08	23.38	22.72	21.68	21.11	23.17	24.63	24.55	24.42	24.19	23.56
P_{safe}	-	0.7444	0.8923	0.9933	0.9981	0.9985	0.7511	0.8709	0.9565	0.9989	0.9925	0.7386
ϵ_x	[m]	0.3504	0.9914	1.4377	1.6131	1.7386	0.3322	0.9832	1.4761	1.9007	2.0384	0.3062
ϵ_θ	[DEG]	53.19	37.22	13.93	10.84	9.36	55.89	48.03	40.11	24.66	22.47	36.86

Table 4.2: Scenario 2: performance for $SMC + SFM$, $SMC + LIN$ and SFM strategies. 500 simulations each were conducted.

	Unit	$SMC + SFM$					$SMC + LIN$					SFM
$T_{horizon}$	[s]	1	2	4	6	8	1	2	4	6	8	-
T_{exit}	[s]	26.82	23.89	24.08	21.12	20.27	27.33	31.48	36.03	29.98	23.71	23.16
P_{safe}	-	0.9908	0.9998	0.9993	0.9977	0.9316	0.9882	0.9965	0.9977	0.9925	0.9486	0.9665
ϵ_x	[m]	0.7677	0.7927	0.6497	0.5701	0.5430	0.7902	1.4279	1.2607	0.8282	0.6760	0.3825
ϵ_θ	[DEG]	9.97	5.50	9.20	10.59	19.97	10.62	14.25	20.33	21.56	21.52	13.67

$SMC + LIN$, when $T_{horizon} \leq 6$. Nonetheless, P_{safe} decreases and ϵ_θ increases when $T_{horizon} > 6$. This is motivated by the fact that the tested temporal logic formula is less likely to be satisfied over a large horizon in a crowded environment. As a consequence, the planning algorithm suggests the user to stop and/or to change direction in order to avoid the unfeasible path, thus raising ϵ_θ.

The SFM strategy exhibits the lower ϵ_x because it tends to keep the user closer to the global plan. However, this reflects negatively on the "comfort" of the user, especially on P_{safe}, because the model doesn't have an explicit notion of *minimum safety distance to agents*. This behaviour is more evident in *scenario 1*.

Timings on the Beagleboard xM We measured the time needed by the Beagleboard xM to execute the scenarios presented in the previous section. We ran 500 simulations each and we timed the execution of every single decision step for the $SMC + SFM$ strategy, that is, the time needed for a single run of Algorithm 4 using the Social Force Model as the model for the agents. We also set $N = 50$. We computed both the average $\mu_1 = 228.9$ ms and $\mu_2 = 2026.1$ ms and standard deviations $\sigma_1 = 392.1$ ms and $\sigma_2 = 2432.1$ ms of the timings for *scenario 1* and *scenario 2*, respectively. If we allow a maximum 1000 ms latency to compute the decision step, the current implementation is able to satisfy it in 93.4% of the cases for *scenario 1* and in 40.9% of the cases for *scenario 2*.

4.5 Conclusion

The social force model is a generative model of pedestrian behaviour, which we have embedded in an efficient online motion planning algorithm. We parametrise the model with data from sensors in real time and thus hypothesise future trajectories.

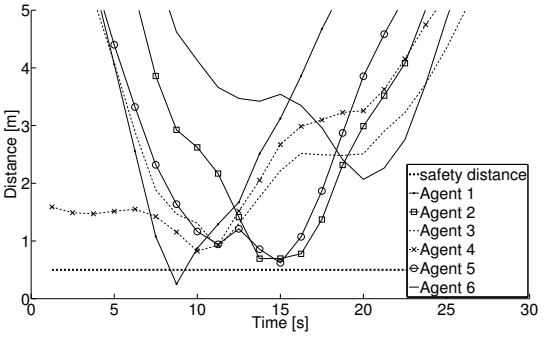


Figure 4.3: Scenario 2. Distances of the agents with respect to the user during one particular run of *scenario 2*. In this case the safety distance has been set to 0.5 m (dashed line) and has been violated once.

The model's stochastic element limits its predictive ability but allows us to consider a distribution of possible futures. The reactive part of the algorithm is provided by statistical model checking technology. Thus, the algorithm verifies the hypothesised trajectories against the user's goals and constraints expressed in temporal logic. In this way the algorithm finds the immediate course of action that maximises the user's probability of success.

The apparently random behaviour of pedestrians is often the result of deterministic choices on their part. We hope to to improve the performance of our algorithm by recognising these choices and thus replacing some of the stochasticity. To this end, in [CFG+13] we have identified behavioural templates that may be incorporated into the social force model. Also, equation (4.2) includes the possibility to explicitly model incidental attractive and repulsive forces that might, for example, arise from interesting shops and areas with high probability of crowding, respectively. Such forces apply to pedestrians in general and could enrich the existing model.

As part of our larger project [DAL], we also propose to include advanced sensor techniques to recognise known interesting or hostile people (e.g., using facial recognition) and to generally avoid people exhibiting hostile behaviour. Such forces apply asymmetrically and would obviously have to be included in an anisotropic version of the social force model [HFMV02].

A significant part of the challenge of our motion planning application is the performance of its implementation, especially the implementation of the SMC algorithms on small devices (a major challenge for our area). Current hardware performance forces us to accept the necessity of multiple boards to handle the overall computational burden, but there is a clear advantage if a portable device can be made to work on a single board. The embedded computing boards we have chosen for our implementation include high performance graphical processor units (GPUs) that can be used for general purpose parallel computing. Since statistical model checking requires multiple independent simulation runs, we propose to exploit the GPU to gain a significant increase in performance..

$T_{decision}$ is necessarily less than $T_{horizon}$, hence the algorithm predicts traces in time periods that overlap from one iteration to the next. While the predictions of older simulations are likely to be less accurate with respect to the current reality, data from the previous iterations may be employed, suitably weighted, to build a probabilistic map of the good and bad locations in the local environment. This map can be used to avoid simulations that explore directions that are unlikely to be successful and to provide haptic feedback if the user chooses to diverge from the proposed path.

Chapter 5

Statistical Model Checking for Markov Decision Processes

5.1 Introduction

The objective of this chapter is to lift SMC to a class of systems that mixes non-deterministic and stochastidc aspects. As outlined in [Var85], non-determinism can be used to represent communication between processes.

In this chapter, we consider Markov decision processes which describe systems that interleave nondeterministic *actions* and probabilistic transitions. This model has proved useful in many real optimisation problems [Whi85, Whi88, Whi93] and, in particular, may be used to represent concurrent probabilistic programs (see, e.g., [AdA95, BK08]). Such models comprise probabilistic subsystems whose transitions depend on the states of the other subsystems, while the order in which concurrently enabled transitions execute is nondeterministic. This order may radically affect the behaviour of a system or the probability that a system will satisfy a given property. In the latter case, it is usual to calculate the upper and lower bounds.

For example, consider the network of computational nodes depicted in Fig. 5.1 (relating to the case study in Section 5.5.4). Given that one of the nodes is infected by a virus, we would like to calculate the probability that a target node becomes infected. If we know the average probability that the virus will pass from one node to the next, we could model the system as a discrete time Markov chain and analyse it to find the average probability that any particular node will become infected. Such a model ignores the possibility that the virus might actually choose which node to infect, e.g., to maximise the probability of passing through the barrier layer. Under such circumstances, some nodes might be infected with near certainty or with only very low probability, but this would not be adequately captured by the Markov chain. By modelling the virus's choice of node as a nondeterministic transition in an MDP, the maximum and minimum probabilities of infection can be considered.

Fig. 5.2 shows a typical fragment of an MDP. Its execution semantics are as follows. In a given state (s_0), an action ($a_1, a_2, \dots$) is chosen nondeterministically to select a distribution of probabilistic transitions ($p_1, p_2, \dots$ or p_3, p_4, etc.). A

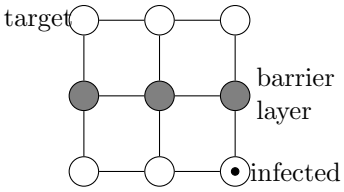


Figure 5.1: Model of network virus infection.

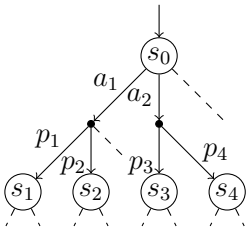


Figure 5.2: Fragment of a Markov decision process.

probabilistic choice is then made to select the next state $(s_1, s_2, s_3, s_4, \dots)$.

To calculate the expected probability of a sequence of states, as required by verification, it is necessary to define how the nondeterminism in the MDP will be resolved. In the literature this is often called a *strategy*, a *policy*, an *adversary* or a *scheduler*. Classic analysis of MDPs is concerned with finding schedulers that maximise or minimise rewards assigned to each of the actions [Bel57, Put94]. In this work we focus on MDPs in the context of *model checking* concurrent probabilistic systems to find schedulers that maximise or minimise the probability of a property. Model checking is an automatic technique to verify that a system satisfies a property specified in temporal logic [CES09]. *Probabilistic* model checking quantifies the probability that a probabilistic system will satisfy a property [HJ94].

Numerical model checking algorithms to solve purely probabilistic systems are costly in time and space. Finding extremal schedulers in MDPs is generally more so, because it is effectively necessary to consider all possible ways that the nondeterminism might be resolved [AdA95]. While memory-efficient ("lightweight") Mone Carlo techniques have been developed to address the probabilistic problem (i.e., *statistical* model checking), until recently [LST14] it has not been possible to address the nondeterministic problem in this way.

Building on the techniques introduced in [LST14], in this work we present "smart sampling" algorithms that make efficient use of a simulation budget and thus make significant improvements to lightweight verification of MDPs. We have implemented the algorithms in Plasma Lab and demonstrate their performance on a number of case studies from the literature. Lightweight verification of MDPs is nevertheless in its infancy, so we also highlight counterexamples that motivate future work.

5.1.1 The Problems of schedulers and state explosion

The classic algorithms to solve MDPs are *policy iteration* and *value iteration* [Put94]. Model checking algorithms for MDPs may use value iteration applied to probabilities [BK08, Ch. 10] or solve the same problem using linear programming [AdA95]. All consider *history-dependent* schedulers. Given an MDP with set of actions A, having a set of states S that induces a set of sequences of states $\Omega = S^+$, a history-dependent (general) scheduler is a function $\mathbf{S} : \Omega \to A$. A memoryless scheduler is a function $\mathbf{M} : S \to A$. Intuitively, at each state in the course of an execution,

a general scheduler (**S**) chooses an action based on the sequence of previous states and a memoryless scheduler (**M**) chooses an action based only on the current state. History-dependent schedulers therefore include memoryless schedulers.

Fig. 5.3 illustrates a simple MDP for which memoryless and history-dependent schedulers give different optima for logical property $\mathbf{X}(\psi \wedge \mathbf{X}\mathbf{G}^t \neg\psi)$ when $p_1 \neq p_2$ and $t > 0$. The property makes use of the linear temporal operators *next* (**X**) and *globally* (**G**). Intuitively, the property states that on the next step ψ will be true and, on the step after that, $\neg\psi$ will be remain true for $t+1$ time steps. The property is satisfied by the sequence of states $s_0 s_1 s_0 s_0 \cdots$. If $p_1 > p_2$, the maximum probability for $s_0 s_1$ is achieved with action a_2, while the maximum probability for $s_0 s_0$ is achieved with action a_1. Given that both transitions start in the same state, a memoryless scheduler will not achieve the maximum probability achievable with a history-dependent scheduler.

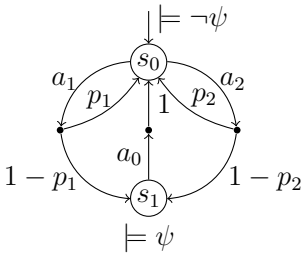


Figure 5.3: MDP with different optima for general and memoryless schedulers.

The principal challenge of finding optimal schedulers is what has been described as the 'curse of dimensionality' [Bel57] and the 'state explosion problem' [CES09]: the number of states of a system increases exponentially with respect to the number of interacting components and state variables. This phenomenon has led to the design of sampling algorithms that find 'near optimal' schedulers to optimise rewards in discounted MDPs [KMN02] and motivate our own work to develop lightweight verification algorithms to optimise probabilities.

The state explosion problem of model checking purely probabilistic systems has been well addressed by *statistical* model checking (SMC) [YS02a]. SMC uses an executable model to approximate the probability that a system satisfies a specified property by the proportion of simulation executions that individually satisfy it. The state space of the system is not constructed explicitly – states are generated on the fly during simulation – hence SMC is efficient for large, possibly infinite state, systems. Moreover, since the simulations are required to be statistically independent, SMC may be efficiently divided on parallel computing architectures.

SMC cannot be applied to MDPs without first resolving the nondeterminism. Since nondeterministic and probabilistic choices are interleaved in an MDP, schedulers are typically of the same order of complexity as the system as a whole and may be infinite. As a result, previous SMC algorithms for MDPs have considered only memoryless schedulers or have other limitations (see Section 5.2).

Organisation of the chapter. In Section 5.2, we briefly present some related work. Section 5.3 discusses our main idea for leightweight simulation via hash functions. Section 5.4 presents the main algorithms, while Section 5.5 and 5.7 discuss some experimental results and conclude the chapter, respectively.

5.2 Related work

Lifting SMC beyond pure stochastic systems is a long standing objective which has been the subject of a series of recent works. The Kearns algorithm [KMN02] is the classic 'sparse sampling algorithm' for large, infinite horizon, discounted MDPs. It constructs a 'near optimal' scheduler piecewise, by approximating the best action from a current state, using a stochastic depth-first search. Importantly, optimality is with respect to rewards, not probability (as required by standard model checking tasks). The algorithm can work with large, potentially infinite state MDPs because it explores a probabilistically bounded search space. This, however, is exponential in the discount. To find the action with the greatest expected reward in the current state, the algorithm recursively estimates the rewards of successive states, up to some maximum depth defined by the discount and desired error. Actions are enumerated while probabilistic choices are explored by sampling, with the number of samples set as a parameter. By iterating local exploration with probabilistic sampling, the discount guarantees that the algorithm eventually converges. The stopping criterion is when successive estimates differ by less than some error threshold.

There have been several recent attempts to apply SMC to nondeterministic models [BFHH11a, LP12, HMZ+12, HT13, LST14]. In [BFHH11a, HT13] the authors present on-the-fly algorithms to remove 'spurious' nondeterminism, so that standard SMC may be used. This approach is limited to the class of models whose nondeterminism does not affect the resulting probability of a property. The algorithms therefore do not attempt to address the standard MDP model checking problems related to finding optimal schedulers.

In [LP12] the authors first find a memoryless scheduler that is near optimal with respect to a reward scheme and discount, using an adaptation of the Kearns algorithm. This induces a Markov chain whose properties may be verified with standard SMC. By storing and re-using information about visited states, the algorithm improves on the performance of the Kearns algorithm, but is thus limited to tractable memoryless schedulers. The near optimality of the induced Markov chain is with respect to rewards, not probability, hence [LP12] does not address the standard model checking problems of MDPs.

In [HMZ+12] the authors present an SMC algorithm to decide whether there exists a memoryless scheduler for a given MDP, such that the probability of a property is above a given threshold. The algorithm has an inner loop that generates candidate schedulers by iteratively improving a probabilistic scheduler, according to sample executions that satisfy the property. The algorithm is limited to memoryless schedulers because the improvement process learns by counting state-action pairs. The outer loop tests the candidate scheduler against the hypothesis using SMC and is iterated until an example is found or sufficient attempts have been made. The approach has several problems. The inner loop does not in general converge to the true optimum, but is sometimes successful at finding an example because the outer loop randomly explores local maxima. This makes the number of samples used by the inner loop critical: too many may reduce the randomness of the outer loop's

exploration and thus significantly reduce the probability of finding examples. A further problem is that the repeated hypothesis tests of the outer loop will eventually produce erroneous results.

Very recently, the elements of lightweight verification for MDPs were introduced in [LST14]. By sampling directly from history-dependent scheduler space using only $\mathcal{O}(1)$ memory, [LST14] opens up the possibility of scalable verification of MDPs. It is nevertheless easy to construct examples for which the simple sampling strategies of [LST14] are not feasible, motivating the present work.

5.3 Lightweight verification of MDPs

Storing schedulers as explicit mappings does not scale, so we represent schedulers using uniform pseudo-random number generators (PRNG) that are initialised by a *seed* and iterated to generate the next pseudo-random value. In general, such PRNGs aim to ensure that arbitrary subsets of sequences of iterates are uniformly distributed and that consecutive iterates are statistically independent. PRNGs are commonly used to implement the uniform probabilistic scheduler, which chooses actions uniformly at random and thus explores all possible combinations of nondeterministic choices. Executing such an implementation twice with the same seed will produce identical executions. Executing the implementation with a different seed will produce an unrelated set of choices: individual schedulers cannot be identified, so it is not possible to estimate the probability of a property under a specific scheduler.

An apparently plausible solution is to use independent PRNGs to resolve nondeterministic and probabilistic choices. It is then possible to generate multiple probabilistic simulation executions per scheduler by keeping the seed of the PRNG for nondeterministic choices fixed while choosing random seeds for a separate PRNG for probabilistic choices. Unfortunately, the schedulers generated by this approach do not span the full range of general or even memoryless schedulers. Since the sequence of iterates from the PRNG used for nondeterministic choices will be the same for all instantiations of the PRNG used for probabilistic choices, the i^{th} iterate of the PRNG for nondeterministic choices will always be the same, regardless of the state arrived at by the previous probabilistic choices. The i^{th} chosen action can be neither state nor execution dependent, as required by scheduler functions **M** and **S**, respectively.

5.3.1 General schedulers using hash functions

We therefore construct a per-step PRNG seed that is a *hash* of the integer identifying a specific scheduler concatenated with an integer representing the sequence of states up to the present.

We assume that a state of an MDP is an assignment of values to a vector of system variables $v_i, i \in \{1, \ldots, n\}$. Each v_i is represented by a number of bits b_i, typically corresponding to a primitive data type (*int*, *float*, *double*, etc.). The state

can thus be represented by the concatenation of the bits of the system variables, such that a sequence of states may be represented by the concatenation of the bits of all the states. Without loss of generality, we interpret such a sequence of states as an integer of $\sum_{i=1}^{n} b_i$ bits, denoted $\overline{s}$, and refer to this in general as the *execution vector*. A scheduler is denoted by an integer σ, which is concatenated to $\overline{s}$ (denoted $\sigma : \overline{s}$) to uniquely identify a execution and a scheduler. Our approach is to generate a hash code $h = \mathcal{H}(\sigma : \overline{s})$ and to use h as the seed of a PRNG that resolves the next nondeterministic choice.

The hash function $\mathcal{H}$ thus maps $\sigma : \overline{s}$ to a seed that is deterministically dependent on the execution and the scheduler. The PRNG maps the seed to a value that is uniformly distributed but nevertheless deterministically dependent on the execution and the scheduler. In this way we approximate the schedulers functions **S** and **M** described in Section 5.1.1. Importantly, the technique only relies on the standard properties of hash functions and PRNGs. Algorithm 5 is the basic simulation function used by our algorithms.

Algorithm 5: Simulate

Input:
$\mathcal{M}$: an MDP with initial state s_0
φ: a property
σ: an integer identifying a scheduler

Output:
ω: a simulation execution

```
1  Let U_prob, U_nondet be uniform PRNGs with respective samples r_pr, r_nd
2  Let H be a hash function
3  Let s denote a state, initialised s ← s_0
4  Let ω denote a execution, initialised ω ← s
5  Let s̄ be the execution vector, initially empty
6  Set seed of U_prob randomly
7  while ω ⊨ φ is not decided do
8      s̄ ← s̄ : s
9      Set seed of U_nondet to H(σ : s̄)
10     Iterate U_nondet to generate r_nd and use to resolve nondeterministic choice
11     Iterate U_prob to generate r_pr and use to resolve probabilistic choice
12     Set s to the next state
13     ω ← ω : s
14 end
```

5.3.2 An efficient iterative hash function

To implement our approach, we use an efficient hash function that constructs seeds incrementally. The function is based on modular division [Knu98, Ch. 6], such that $h = (\sigma : \overline{s}) \bmod m$, where m is a suitably large prime.

Since $\bar{s}$ is a concatenation of states, it is usually very much larger than the maximum size of integers supported as primitive data types. Hence, to generate h we use Horner's method [Hor19] [Knu98, Ch. 4]: we set $h_0 = \sigma$ and find $h \equiv h_n$ (n as in Section 5.3.1) by iterating the recurrence relation

$$h_i = (h_{i-1} 2^{b_i} + v_i) \bmod m. \tag{5.1}$$

The size of m defines the maximum number of different hash codes. The precise value of m controls how the hash codes are distributed. To avoid collisions, a simple heuristic is that m should be a large prime not close to a power of 2 [CCLRS09, Ch. 11]. Practically, it is an advantage to perform calculations using primitive data types that are native to the computational platform, so the sum in (5.1) should always be less than or equal to the maximum permissible value. To achieve this, given $x, y, m \in \mathcal{N}$, we note the following congruences:

$$(x + y) \bmod m \equiv (x \bmod m + y \bmod m) \bmod m \tag{5.2}$$

$$(xy) \bmod m \equiv ((x \bmod m)(y \bmod m)) \bmod m \tag{5.3}$$

The addition in (5.1) can thus be re-written in the form of (5.2), such that each term has a maximum value of $m - 1$:

$$h_i = ((h_{i-1} 2^{b_i}) \bmod m + (v_i) \bmod m) \bmod m \tag{5.4}$$

To prevent overflow, m must be no greater than half the maximum possible integer. Re-writing the first term of (5.4) in the form of (5.3), we see that before taking the modulus it will have a maximum value of $(m-1)^2$, which will exceed the maximum possible integer. To avoid this, we take advantage of the fact that h_{i-1} is multiplied by a power of 2 and that m has been chosen to prevent overflow with addition. We thus apply the following recurrence relation:

$$\begin{aligned} (h_{i-1} 2^j) \bmod m &= (h_{i-1} 2^{j-1}) \bmod m \\ &+ (h_{i-1} 2^{j-1}) \bmod m \end{aligned} \tag{5.5}$$

Equation (5.5) allows our hash function to be implemented using efficient native arithmetic. Moreover, we infer from (5.1) that to find the hash code corresponding to the current state in a execution, we need only know the current state and the hash code from the previous step. When considering memoryless schedulers we need only know the current state.

5.3.3 Hypothesis testing multiple schedulers

To decide whether there exists a scheduler such that $\mathrm{P}(\omega \models \varphi) \bowtie p$, we apply the SPRT to multiple (randomly chosen) schedulers. Since the probability of error with the SPRT applied to multiple hypotheses is cumulative, we consider the probability of no errors in any of M tests. Hence, in order to ensure overall error probabilities α

and β, we adopt $\alpha_M = 1 - \sqrt[M]{1-\alpha}$ and $\beta_M = 1 - \sqrt[M]{1-\beta}$ in our stopping conditions. H_1 is accepted if $ratio \geq (1-\beta_M)/\alpha_M$ and H_0 is accepted if $ratio \leq \beta_M/(1-\alpha_M)$. Algorithm 6 demonstrates the sequential hypothesis test for multiple schedulers. If the algorithm finds an example, the hypothesis is true with at least the specified confidence.

Algorithm 6: Hypothesis testing multiple schedulers

Input:
$\mathcal{M}, \varphi$: the MDP and property of interest
$H \in \{H_0, H_1\}$: the hypothesis with interval $\theta \pm \varepsilon$
α, β: the desired error probabilities of H
M: the maximum number of schedulers to test

Output: The result of the hypothesis test

Let $p^0 = \theta + \varepsilon$ and $p^1 = \theta - \varepsilon$ be the bounds of H
Let $\alpha_M = 1 - \sqrt[M]{1-\alpha}$ and $\beta_M = 1 - \sqrt[M]{1-\beta}$
Let $A = (1-\beta_M)/\alpha_M$ and $B = \beta_M/(1-\alpha_M)$
Let $\mathcal{U}_{\text{seed}}$ be a uniform PRNG and σ be its sample
for $i \in \{1, \ldots, M\}$ *while* H *is not accepted* **do**
    Iterate $\mathcal{U}_{\text{seed}}$ to generate σ_i
    Let $ratio = 1$
    while $ratio > A \wedge ratio < B$ **do**
        $\omega \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma_i)$
        $ratio \leftarrow \frac{(p^1)^{\mathbf{1}(\omega \models \varphi)}(1-p^1)^{\mathbf{1}(\omega \not\models \varphi)}}{(p^0)^{\mathbf{1}(\omega \models \varphi)}(1-p^0)^{\mathbf{1}(\omega \not\models \varphi)}} ratio$
    end
    if $ratio \leq A \wedge H = H_0 \vee ratio \geq B \wedge H = H_1$ **then**
        accept H
    end
end

5.3.4 Estimating multiple schedulers

We consider the strategy of sampling M schedulers to estimate the optimum probability. We thus generate M estimates $\{\hat{p}_1, \ldots, \hat{p}_M\}$ and take either the maximum ($\hat{p}_{\max}$) or minimum ($\hat{p}_{\min}$), as required. To overcome the cumulative probability of error with the standard Chernoff bound, we specify that *all* estimates $\hat{p}_i$ must be within ε of their respective true values p_i, ensuring that any $\hat{p}_{\min}, \hat{p}_{\max} \in \{\hat{p}_1, \ldots, \hat{p}_M\}$ are within ε of their true value. Given that all estimates $\hat{p}_i$ are statistically independent, the probability that all estimates are less than their upper bound is expressed by $\mathrm{P}(\bigwedge_{i=1}^{M} \hat{p}_i - p_i \leq \varepsilon) \geq (1-e^{-2N\varepsilon^2})^M$. Hence, $\mathrm{P}(\bigvee_{i=1}^{M} \hat{p}_i - p_i \geq \varepsilon) \leq 1-(1-e^{-2N\varepsilon^2})^M$, giving $N = \left\lceil -\ln\left(1 - \sqrt[M]{1-\delta}\right)/(2\varepsilon^2) \right\rceil$ for user-specified parameters M, ε and δ. This ensures that $\mathrm{P}(p_{\min} - \hat{p}_{\min} \geq \varepsilon) \leq \delta$ and $\mathrm{P}(\hat{p}_{\max} - p_{\max} \geq \varepsilon) \leq \delta$. To ensure the more usual stronger conditions that $\mathrm{P}(\mid p_{\max} - \hat{p}_{\max} \mid \geq \varepsilon) \leq \delta$ and

$\mathrm{P}(\mid p_{\min} - \hat{p}_{\min} \mid \geq \varepsilon) \leq \delta$, we have

$$N = \left\lceil \left(\ln 2 - \ln \left(1 - \sqrt[M]{1-\delta} \right) \right) / (2\varepsilon^2) \right\rceil . \tag{5.6}$$

N scales logarithmically with M, making it tractable to consider many schedulers. In the case of $M = 1$, (5.6) degenerates to the Chernoff bound. Algorithm 7 is the resulting extremal probability estimation algorithm for multiple schedulers.

Algorithm 7: Estimation with multiple schedulers

Input:
$\mathcal{M}, \varphi$: the MDP and property of interest
ε, δ: the required Chernoff bound
M: the number of schedulers to test
Output: Extremal estimates $\hat{p}_{\min}$ and $\hat{p}_{\max}$

Let $N = \left\lceil \ln(2/(1 - \sqrt[M]{1-\delta}))/(2\varepsilon^2) \right\rceil$ be the no. of simulations per scheduler
Let $\mathcal{U}_{\text{seed}}$ be a uniform PRNG and σ its sample
Initialise $\hat{p}_{\min} \leftarrow 1$ and $\hat{p}_{\max} \leftarrow 0$
Set seed of $\mathcal{U}_{\text{seed}}$ randomly
for $i \in \{1, \ldots, M\}$ **do**
    Iterate $\mathcal{U}_{\text{seed}}$ to generate σ_i
    Let $\mathit{truecount} = 0$ be the initial number of executions that satisfy φ
    for $j \in \{1, \ldots, N\}$ **do**
        $\omega_j \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma_i)$
        $\mathit{truecount} \leftarrow \mathit{truecount} + \mathbf{1}(\omega_j \models \varphi)$
    end
    Let $\hat{p}_i = \mathit{truecount}/N$
    if $\hat{p}_{\max} < \hat{p}_i$ **then**
        $\hat{p}_{\max} = \hat{p}_i$
    end
    if $\hat{p}_i > 0 \wedge \hat{p}_{\min} > \hat{p}_i$ **then**
        $\hat{p}_{\min} = \hat{p}_i$
    end
end
if $\hat{p}_{\max} = 0$ **then**
    No schedulers were found to satisfy φ
end

Fig. 5.4 shows the empirical cumulative distribution of schedulers generated by Algorithm 7 applied to the MDP of Fig. 5.3, using $p_1 = 0.9$, $p_2 = 0.5$, $\varphi = \mathbf{X}(\psi \wedge \mathbf{XG}^4 \neg\psi)$, $\varepsilon = 0.01$, $\delta = 0.01$ and $M = 300$. The vertical red and blue lines mark the true probabilities of φ under each of the history-dependent and memoryless schedulers, respectively. The grey rectangles show the $\pm\varepsilon$ error bounds, relative to the true probabilities. There are multiple estimates per scheduler, but all estimates are within their respective confidence bounds.

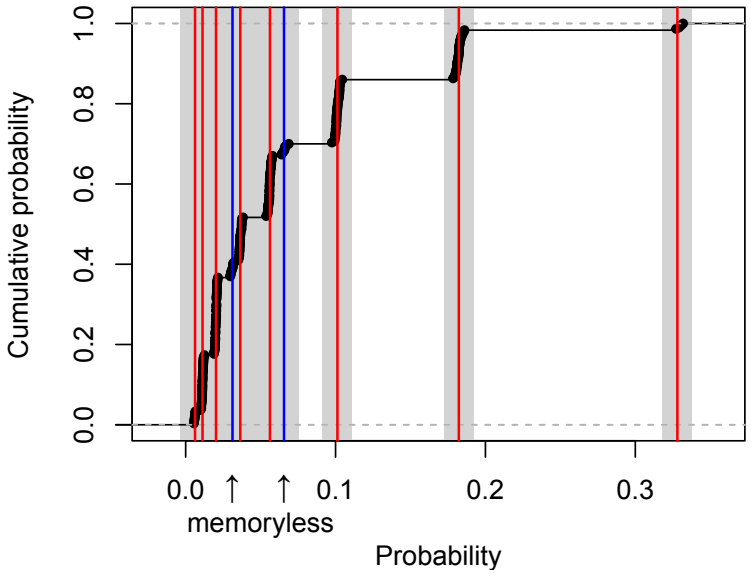


Figure 5.4: Empirical cumulative distribution of estimates from Algorithm 7 applied to MDP of Fig. 5.3.

5.4 Smart sampling

The simple sampling strategies used by Algorithms 6 and 7 have the disadvantage that they allocate equal simulation budget to all schedulers, regardless of their merit. In general, the problem we address has two independent components: the rarity of optimal schedulers and the rarity of the property under an optimal scheduler. We should allocate our simulation budget accordingly and not waste budget on schedulers that are clearly not optimal.

Motivated by the above, our smart estimation algorithm comprises three stages: (*i*) an initial undirected sampling experiment to discover the nature of the problem, (*ii*) a targeted sampling experiment to generate a candidate set of schedulers with high probability of containing an optimal scheduler and (*iii*) iterative refinement of the candidates to estimate the probability of the best scheduler with specified confidence. By excluding the schedulers with the worst estimated probabilities and re-allocating their simulation budget to the schedulers that remain, at each iterative step of stage (*iii*) the number of schedulers reduces while the confidence of their estimates increases. With a suitable choice of per-iteration budget, the algorithm is guaranteed to terminate.

In the following subsection we develop the theoretical basis of stage (*ii*).

5.4.1 Maximising the probability of seeing a good scheduler

In what follows we assume the existence of an MDP and a property φ whose probability we wish to maximise by choosing a suitable scheduler from the set $\mathbf{S}$. Let $\mathcal{P} : \mathbf{S} \to [0, 1]$ be a function mapping schedulers to their probability of satisfying φ and let $p_{\max} = \max_{s \in \mathbf{S}}(\mathcal{P}(s))$. For the sake of exposition, we consider the problem

of finding a scheduler that maximises the probability of satisfying φ and define a "good" scheduler to be one in the set $\mathbf{S}_g = \{s \in \mathbf{S} \mid \mathcal{P}(s) \geq p_{\max} - \varepsilon\}$ for some $\varepsilon \in (0, p_{\max}]$. Intuitively, a good scheduler is one whose probability of satisfying φ is within ε of $p_{\max}$, noting that we may similarly define a good scheduler to be one within ε of $p_{\min} = \min_{s \in \mathbf{S}}(\mathcal{P}(s))$, or to be in any other subset of $\mathbf{S}$. In particular, to address reward-based MDP optimisations, a good scheduler could be defined to be the subset of $\mathbf{S}$ that is near optimal with respect to a reward scheme. The notion of a "best" scheduler follows intuitively from the definition of a good scheduler.

Assuming we sample uniformly from $\mathbf{S}$, the probability of finding a good scheduler is $p_g = |\mathbf{S}_g|/|\mathbf{S}|$. The average probability of a good scheduler is $p_{\overline{g}} = \sum_{s \in \mathbf{S}_g} \mathcal{P}(s)/|\mathbf{S}_g|$. If we select M schedulers uniformly at random and verify each with N simulations, the expected number of executions that satisfy φ using a good scheduler is thus $Mp_gNp_{\overline{g}}$. The probability of seeing a execution that satisfies φ using a good scheduler is the cumulative probability

$$(1 - (1 - p_g)^M)(1 - (1 - p_{\overline{g}})^N). \tag{5.7}$$

Hence, for a given simulation budget $N_{\max} = NM$, to implement stage (ii) the idea is to choose N and M to maximise (5.7) and keep any scheduler that produces at least one execution that satisfies φ. Since, a priori, we are generally unaware of even the magnitudes of p_g and $p_{\overline{g}}$, stage (i) is necessarily uninformed and we set $N = M = \lceil \sqrt{N_{\max}} \rceil$. The results of stage (i) allow us to estimate p_g and $p_{\overline{g}}$ (see Fig. 5.9a) and thus maximise (5.7) using the heuristic $N = \lceil 1/p_{\overline{g}} \rceil$, which has near optimal performance in all but extreme cases.

5.4.2 Smart estimation

Algorithm 8 is our smart estimation algorithm to find schedulers that maximise the probability of a property. The algorithm to find minimising schedulers is similar. Lines 1 to 6 implement stage (i), lines 7 to 11 implement stage (ii) and lines 12 to 26 implement stage (iii). Note that the algorithm distinguishes $p_{\max}$ (the notional true maximum probability), $p_{\overline{\max}}$ (the true probability of the best candidate scheduler found) and $\hat{p}_{\overline{\max}}$ (the estimated probability of the best candidate scheduler).

The per-iteration simulation budget $N_{\max}$ must be greater than the number needed by the standard Chernoff bound, so ensure that there will at least be sufficient simulations to guarantee the specified confidence when the algorithm refines the candidate set to a single scheduler. Typically, the per-iteration budget will be greater than this, such that the required confidence is reached according to the Chernoff bound for multiple estimates (5.6), before refining the set of schedulers to a single element. Moreover, lines 17 to 20 allow the algorithm to quit as soon as the minimum number of simulations is reached.

5.4.3 Smart hypothesis testing

We wish to test the hypothesis that there exists a scheduler such that property φ has probability $\bowtie \theta$, where $\bowtie \in \{\geq, \leq\}$. Two advantages of sequential hypothesis

testing are that it is not necessary to estimate the actual probability to know if an hypothesis is satisfied, and the easier the hypothesis is to satisfy, the quicker it is to get a result. Algorithm 9 maintains these advantages and uses smart sampling to improve on the performance of Algorithm 6.

A sub-optimal approach would be to use Algorithm 8 to refine a set of schedulers until one is found whose estimate satisfies the hypothesis with confidence according to a Chernoff bound. Our approach is to exploit the fact that the *average* estimate at each iteration of Algorithm 8 is known with high confidence, i.e., confidence given by the total simulation budget. This follows directly from the result of [Che52], where the bound is specified for a sum of arbitrary random variables, not necessarily with identical expectations. By similar arguments based on [Wal45b], it follows that the sequential probability ratio test may also be applied to the sum of results produced during the course of an iteration of Algorithm 8. Moreover, it is possible to test each scheduler with respect to its individual results and the current number of schedulers, according to the bound given in Section 5.3.3.

Hence, if the "average scheduler" or an individual scheduler ever satisfies the hypothesis (lines 26 and 27), the algorithm immediately terminates and reports that the hypothesis is satisfied with the specified confidence. If the "best" scheduler ever individually falsifies the hypothesis (lines 29 and 30), the algorithm also terminates and reports the result. Note that this outcome does not imply that there is no scheduler that will satisfy the hypothesis, only that no scheduler was found with the given budget. Finally, if the algorithm refines the initial set of schedulers to a single instance and the hypothesis was neither satisfied nor falsified, an inconclusive result is reported (line 38).

We implement one further important optimisation. We use the threshold probability θ to directly define the simulation budget to generate the candidate set of schedulers, i.e. $N = \lceil 1/\theta \rceil$, $M = \lceil \theta N_{\max} \rceil$ (line 3). This is justified because we need only find schedulers whose probability of satisfying φ is greater than θ. By setting $N = \lceil 1/\theta \rceil$, (5.7) ensures that such schedulers, if they exist, have high probability of being observed. The initial coarse exploration used in Algorithm 8 is thus not necessary.

Algorithm 9 is our smart hypothesis testing algorithm. Note that we do not set a precise minimum per-iteration simulation budget because we expect the hypothesis to be decided with many fewer simulations than would be required to estimate the probability. In practice it is expedient to initially set a low per-iteration budget (e.g., 1000) and repeat the algorithm with an increased budget (e.g., increased by an order of magnitude) if the previous test was inconclusive.

5.5 Experiments

To demonstrate the performance of smart sampling, we implemented Algorithms 8 and 9 in Plasma Lab. We performed a number of experiments on standard models taken from the numerical model checking literature, most of which can be found illustrated on the PRISM website [KNP11]. We found that all of our estimation

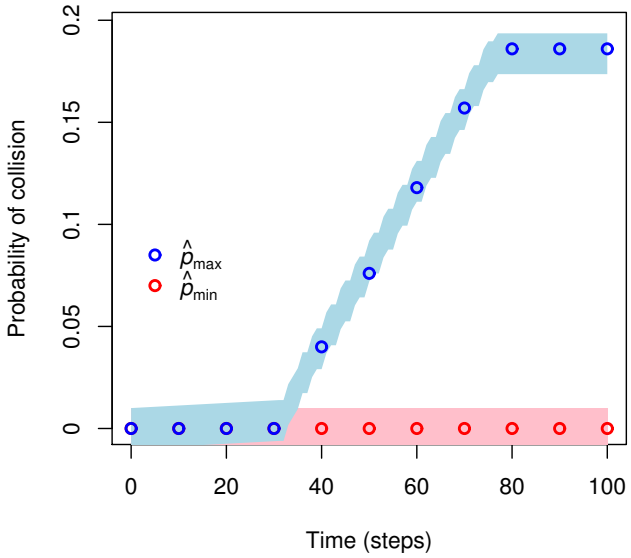


Figure 5.5: Estimated maximum and minimum probabilities of second collision in WLAN protocol. Shaded regions denote true values ±0.01.

experiments achieved their specified Chernoff bounds ($\varepsilon = \delta = 0.01$ in all cases) with a relatively modest per-iteration simulation budget of 10^5 simulations. The actual number of simulation cores used for the estimation results was subject to availability and varied between experiments. To facilitate comparisons we therefore normalise all timings to be with with respect to 64 cores. Typically, each data point was produced in a few tens of seconds. Our hypothesis tests were performed on a single machine, without distribution. Despite this, most experiments completed in just a few seconds (some in fractions of a second), demonstrating that our smart hypothesis testing algorithm is able to take advantage of easy hypotheses.

5.5.1 IEEE 802.11 Wireless LAN protocol

We consider a reachability property of the IEEE 802.11 Wireless LAN (WLAN) protocol model of [KNS02]. The protocol aims to avoid "collisions" between devices sharing a communication channel, by means of an exponential backoff procedure when a collision is detected. We therefore estimate the probability of the second collision at various time steps, using Algorithm 7 with per-iteration budget of 10^5 simulations. Fig. 5.5 illustrates the estimated maximum probabilities ($\hat{p}_{\max}$) and minimum probabilities ($\hat{p}_{\min}$) for time steps $k \in \{0, 10, \ldots, 100\}$. The property is expressed as $\mathbf{F}^k col = 2$. The shaded areas indicate the true probabilities ±0.01, the specified absolute error bound using Chernoff bound $\varepsilon = \delta = 0.01$. Our results are clearly very close to the true values. Table 5.1 gives the results of hypothesis tests based on the same model using property $\mathbf{F}^{100} col = 2$. See Section 5.5.2 for a description.

5.5.2 IEEE 802.3 CSMA/CD protocol

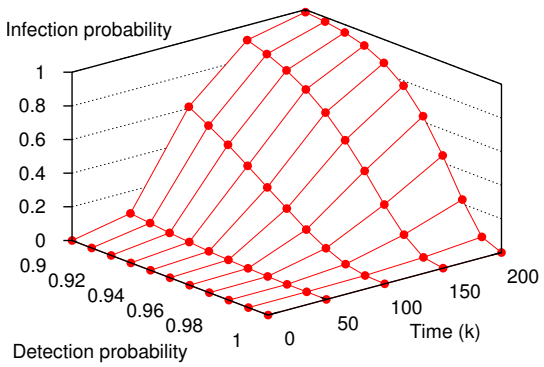
The IEEE 802.3 CSMA/CD protocol is a wired network protocol that is similar in operation to that of IEEE 802.11, but using collision detection instead of collision avoidance. In Table 5.1 we give the results of applying Algorithm 9 to the IEEE 802.3 CSMA/CD protocol model of [KNPS06]. The models and parameters are chosen to compare with results given in Table III in [HMZ+12], hence we also give results for hypothesis tests performed on the WLAN model used in Section 5.5.1. In contrast to the results of [HMZ+12], our results are produced on a single machine, with no parallelisation. There are insufficient details given about the experimental conditions in [HMZ+12] to make a formal comparison (e.g., error probabilities of the hypothesis tests and number of simulation cores), but it seems that the performance of our algorithm is generally much better. We set $\alpha = \beta = \delta = 0.01$, which constitute a fairly tight bound, and note that, as expected, the simulation times tend to increase as the threshold θ approaches the true probability.

5.5.3 Choice coordination

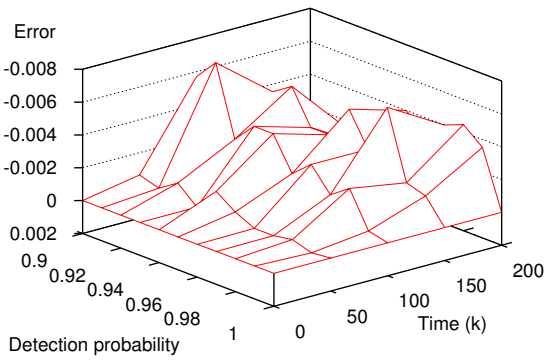
To demonstrate the scalability of our approach, we consider the choice coordination model of [NM10] and estimate the minimum probability that a group of six tourists will meet within T steps. The model has a parameter ($BOUND$) that limits the state space. We set $BOUND = 100$, making the state space of $\approx 5 \times 10^{16}$ states intractable to numerical model checking. Fortunately, it is possible to infer the correct probabilities from tractable parametrisations. For $T = 20$ and $T = 25$ the true minimum probabilities are respectively 0.5 and 0.75. Using smart sampling and a Chernoff bound of $\varepsilon = \delta = 0.01$, we correctly estimate the probabilities to be 0.496 and 0.745 in a few tens of seconds on 64 simulation cores.

5.5.4 Network virus infection

Network virus infection is a subject of increasing relevance. Hence, using a per-iteration budget of 10^5 simulations, we demonstrate the performance of Algorithm 8 on the virus infection model of [KNP11], based on [KNPV09]. The network is illustrated in Fig. 5.1 and comprises three sets of linked nodes: a set of nodes containing one infected by a virus, a set of nodes with no infected nodes and a set of barrier nodes which divides the first two sets. A virus chooses which node to infect nondeterministically. A node detects a virus probabilistically and we vary this probability as a parameter for barrier nodes. We consider time as a second parameter. Figs. 5.6 and 5.7 illustrate the estimated probabilities that the target node in the uninfected set will be infected. We observe in Figs. 5.6b and 5.7b that the estimated minimums are within $[-0.0070, +0.00012]$ and the estimated maximums are within $[-0.00012, +0.0083]$ of their true values. The respective negative and positive biases to these error ranges reflects the fact that Algorithm 8 converges from respectively below and above (as illustrated in Fig. 5.9b). The average time to generate a point in Fig. 5.6 was approximately 100 seconds, using 64 simulation

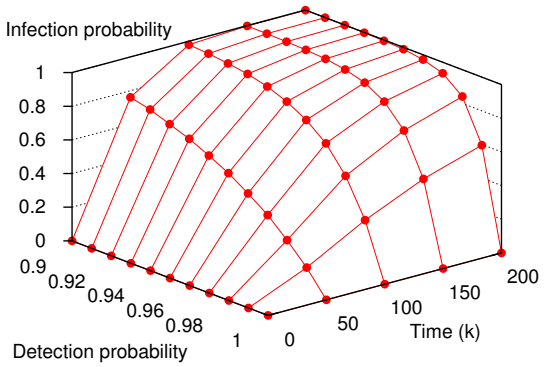


(a) Estimates.

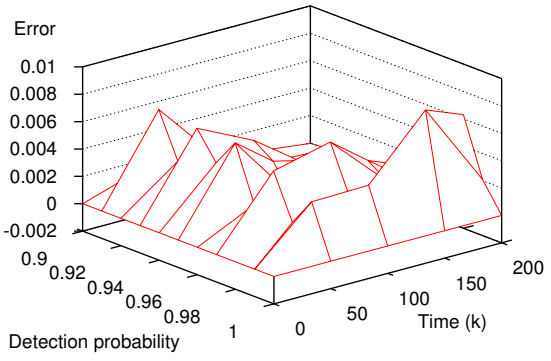


(b) Errors.

Figure 5.6: Minimum probability of network infection.



(a) Estimates.



(b) Errors.

Figure 5.7: Maximum probability of network infection.

cores. Points in Fig. 5.7 took on average approximately 70 seconds.

5.5.5 Gossip protocol

Gossip protocols are an important class of network algorithms that rely on local connectivity to propagate information globally. Using the gossip protocol model of [KNP08], we used Algorithm 8 with per-simulation budget of 10^5 simulations to estimate the maximum ($\hat{p}_{\max}$) and minimum ($\hat{p}_{\min}$) probabilities that the maximum path length between any two nodes is less than 4 after T time steps. This is expressed by the property $\mathbf{F}^T max\_path\_length < 4$. The results are illustrated in Fig. 5.8. Estimates of maximum probabilities are within $[-0, +0.0095]$ of the true values. Estimates of minimum probabilities are within $[-0.007, +0]$ of the true values. Each point in the figure took on average approximately 60 seconds to generate using 64 simulation cores.

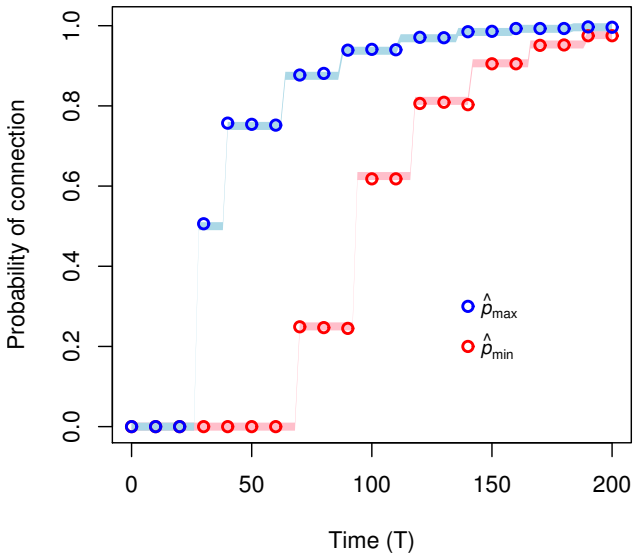


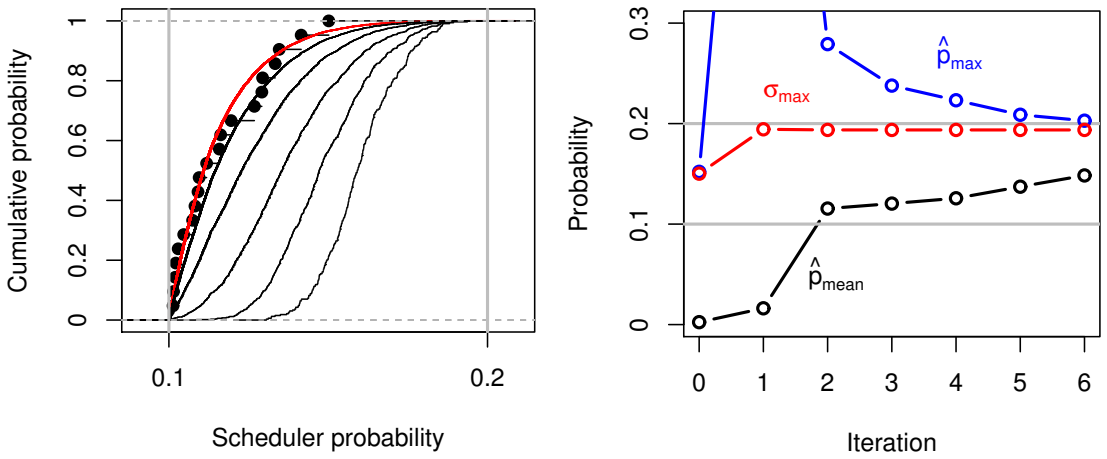
Figure 5.8: Estimated probabilities that maximum path length is < 4 in gossip protocol model. Shaded regions denote ± 0.01 of true values.

5.6 Convergence and counterexamples

The techniques described in the preceding sections open up the possibility of efficient lightweight verification of MDPs, with the consequent possibility to take full advantage of parallel computational architectures, such as multi-core processors, clusters, grids, clouds and general purpose computing on graphics processors (GPGPU). These architectures may potentially divide the problem by the number of available computational devices (i.e., linearly), however this must be considered in the context of scheduler space increasing exponentially with path length. Although Monte Carlo techniques are essentially impervious to the size of the state space (they also work with non-denumerable space), it is easy to construct verification problems for which there is a unique optimal scheduler. Such examples do not necessarily invalidate the approach, however, because it may not be necessary to find the possibly unique optimal scheduler to return a result with a level of statistical confidence. The nature of the distribution of schedulers nevertheless affects efficiency, so in this section we explore the convergence properties of smart sampling, give a counterexample from the literature and motivate our ongoing development of lightweight learning techniques.

Essentially, the problem is that of exponentially distributed schedulers, i.e., having a very low mass of schedulers close to the optimum. Fig. 5.10 illustrates the difference between exponentially decreasing and linearly decreasing distributions with the same overall mass. In both cases $p_{\max} \approx 0.2$ (the density at 0.2 is zero), but the figure shows that there is more probability mass near 0.2 in the case of the linear distribution.

Fig.5.9 illustrates the convergence of Algorithm 8, using a per-iteration budget of 10^6 applied to schedulers whose probability of success (i.e., of satisfying a hy-



(a) Scheduler distributions. Dots denote the results of initial sampling. The red line is the set of schedulers to refine. Black lines show the result of subsequent refinements.

(b) Estimates and schedulers. At each iterative step: $\hat{p}_{\max}$ is the maximum estimate, $\hat{p}_{\text{mean}}$ is the mean estimate and $\sigma_{\max}$ is the true maximum probability of the available schedulers.

Figure 5.9: Convergence of Algorithm 8 with exponentially distributed scheduler probabilities (Fig. 5.10) and per-iteration budget of 10^6 simulations.

pothetical property) is distributed according to the exponential distribution of Fig. 5.10. Fig. 5.9a shows how the initial undirected sampling (black dots) crudely approximates $p_{\max}$. This approximation is then used to generate the candidate set of schedulers (red distribution). The black lines illustrate five iterations of refinement, resulting in a shift of the distribution towards $p_{\max}$. Fig. 5.9b illustrates the same shift in terms of the convergence of probabilities. Iteration 0 corresponds to the undirected sampling. Iteration 1 corresponds to the generation of the candidate set of schedulers. For these first two iterations, $\hat{p}_{\text{mean}}$ includes the schedulers which have zero probability of success. In subsequent iterations the candidates all have non-zero probability of success. Importantly, the figure demonstrates that there is a significant increase in the maximum probability of scheduler success ($\sigma_{\max}$) between iteration 0 and iteration 1, and that this maximum is maintained throughout the subsequent refinements. Despite the apparently very low density of schedulers near $p_{\max}$, Algorithm 8 is able to make a good approximation.

The theoretical performance demonstrated in Fig. 5.9 explains why we are able to achieve good results in Section 4.4. It is nevertheless possible to find examples for which accurate results are difficult to achieve. Fig. 5.11 illustrates the results of applying Algorithm 8 to instances of the self-stabilising algorithm of [IJ90], using a per-iteration budget of 10^5. We see that estimates (black dots) do not well approximate the true values (red shaded areas).

To improve the performance of our algorithms it is possible to better allocate simulation budget. For example, if good schedulers are very rare it may be beneficial to increase the per-iteration budget (thus increasing the possibility of seeing a good scheduler in the initial candidate set) but increase the proportion of schedulers rejected after each iteration (thus reducing the overall number of iterations and

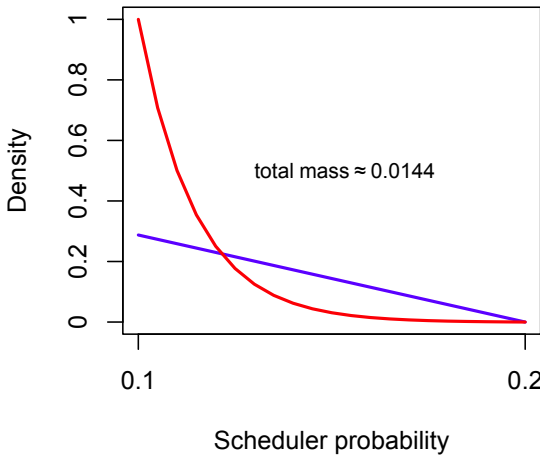


Figure 5.10: Theoretical linear (blue) and exponential (red) scheduler densities. Both have probability mass ≈ 0.0144 and zero density at scheduler probability 0.2.

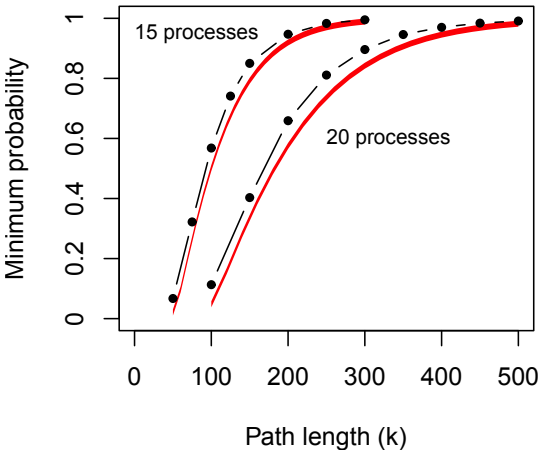


Figure 5.11: Performance of smart sampling (black dots) applied to self-stabilising models of [IJ90]. Red shaded areas denote true values ±0.01.

maintaining a fixed total number of simulations). To avoid rejecting good schedulers under such a regime, it may be necessary to reject fewer schedulers in the early iterations when confidence is low.

Such "secondary optimisations" are unlikely to overcome problems that tend to scale exponentially. Hence, our main focus of future development will be lightweight learning algorithms that facilitate directed sampling and piecewise construction of schedulers.

5.7 Conclusion

We believe our techniques are immediately extensible to continuous time MDPs and other models that use nondeterminism. As noted in Section 5.4.1, it is also possible to apply smart sampling to reward-based MDP optimisation problems. In this case a good scheduler would be one that produces an optimal reward and the initial candidate set of schedulers would be refined according to rewards, rather than probability.

By hashing the set of executions to a smaller set of hash codes, our algorithms sample from only a subset of all possible schedulers. This is not necessarily a problem, since any sampling approach will test far fewer schedulers than the maximum number of hash codes. In line with other statistical techniques, we are investigating means to quantify the confidence of chosen schedulers with respect to optimality.

Despite the foregoing, it is easy to construct examples where good schedulers are vanishingly rare. In such cases, merely quantifying the low confidence may not be useful. Our ongoing focus is therefore the development of lightweight learning tech-

niques that construct schedulers piece-wise, to improve convergence and to consider a much larger set of schedulers.

Algorithm 8: Smart Estimating

Input:
$\mathcal{M}$: an MDP
φ: a property
ϵ, δ: the required Chernoff bound
$N_{\max} > \ln(2/\delta)/(2\epsilon^2)$: the per-iteration budget

Output: $\hat{p}_{\overline{\max}} \approx p_{\max}$, where $p_{\overline{\max}} \approx p_{\max}$ and $\mathrm{P}(|p_{\overline{\max}} - \hat{p}_{\overline{\max}}| \geq \epsilon) \leq \delta$

$N \leftarrow \lceil\sqrt{N_{\max}}\rceil$; $M \leftarrow \lceil\sqrt{N_{\max}}\rceil$
$S \leftarrow \{M \text{ seeds chosen uniformly at random}\}$
$\forall \sigma \in S, \forall i \in \{1, \ldots, N\} : \omega_i^\sigma \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma)$
$R : S \to \mathcal{N}$ maps scheduler seeds to number of executions satisfying φ:
  $R \leftarrow \{(\sigma, n) \mid \sigma \in S \wedge \mathcal{N} \ni n = \sum_{i=1}^{N} \mathbf{1}(\omega_i^\sigma \models \varphi)\}$
$\hat{p}_{\overline{\max}} \leftarrow \max_{\sigma \in S}(R(\sigma)/N)$
$N \leftarrow \lceil 1/\hat{p}_{\overline{\max}}\rceil$, $M \leftarrow \lceil N_{\max}\, \hat{p}_{\overline{\max}}\rceil$
$S \leftarrow \{M \text{ seeds chosen uniformly at random}\}$
$\forall \sigma \in S, \forall i \in \{1, \ldots, N\} : \omega_i^\sigma \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma)$
$R \leftarrow \{(\sigma, n) \mid \sigma \in S \wedge \mathbb{N} \ni n = \sum_{i=1}^{N} \mathbf{1}(\omega_i^\sigma \models \varphi)\}$
$S \leftarrow \{\sigma \in S \mid R(\sigma) > 0\}$
$\forall \sigma \in S,\ R(\sigma) \leftarrow 0$; $i \leftarrow 0$; $\mathit{conf} \leftarrow 1$
while $\mathit{conf} > \delta \wedge S \neq \emptyset$ **do**
  $i \leftarrow i + 1$
  $M_i \leftarrow |S|$
  $N_i \leftarrow 0$
  while $\mathit{conf} > \delta \wedge N_i < \lceil N_{\max}/M_i\rceil$ **do**
    $N_i \leftarrow N_i + 1$
    $\mathit{conf} \leftarrow 1 - (1 - e^{-2\epsilon^2 N_i})^{M_i}$
    $\forall \sigma \in S : \omega_{N_i}^\sigma \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma)$
  end
  $R \leftarrow \{(\sigma, n) \mid \sigma \in S \wedge \mathbb{N} \ni n = \sum_{j=1}^{N_i} \mathbf{1}(\omega_j^\sigma \models \varphi)\}$
  $\hat{p}_{\overline{\max}} \leftarrow \max_{\sigma \in S}(R(\sigma)/N_i)$
  $R' : \{1, \ldots, |S|\} \to S$ is an injective function s.t.
    $\forall (n, \sigma), (n', \sigma') \in R',\ n > n' \Rightarrow R(\sigma) \geq R(\sigma')$
  $S \leftarrow \{\sigma \in S \mid \sigma = R'(n) \wedge n \in \{\lfloor |S|/2\rfloor, \ldots, |S|\}\}$
end

Algorithm 9: Smart Hypothesis Testing

Input:
$\mathcal{M}$: an MDP
φ: a property
$H : \mathrm{P}(\omega \models \varphi) \geq \theta \pm \varepsilon$ is the hypothesis
α, β: the desired error probabilities of H
$N_{\max}$: the per-iteration simulation budget
Output: The result of the hypothesis test

Let $p^0 = \theta + \varepsilon$, $p^1 = \theta - \varepsilon$
Let $A = (1-\beta)/\alpha$, $B = \beta/(1-\alpha)$
$N \leftarrow \lceil 1/\theta \rceil$; $M \leftarrow \lceil \theta N_{\max} \rceil$
$S \leftarrow \{M \text{ seeds chosen uniformly at random}\}$
$\forall \sigma \in S, \forall i \in \{1, \ldots, N\} : \omega_i^\sigma \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma)$
$R \leftarrow \{(\sigma, n) \mid \sigma \in S \wedge \mathbb{N} \ni n = \sum_{i=1}^{N} \mathbf{1}(\omega_i^\sigma \models \varphi)\}$
if $\frac{(p^1)^{\sum R(\sigma)}(1-p^1)^{N_{\max}-\sum R(\sigma)}}{(p^0)^{\sum R(\sigma)}(1-p^0)^{N_{\max}-\sum R(\sigma)}} \leq A$ **then**
  Accept H and quit
end
$S \leftarrow \{\sigma \in S \mid R(\sigma) > 0\}$, $M \leftarrow |S| + 1$
while $M > 1$ **do**
  $M \leftarrow |S|$
  Let $\alpha_M = 1 - \sqrt[M]{1-\alpha}$, $\beta_M = 1 - \sqrt[M]{1-\beta}$
  Let $A_M = (1-\beta_M)/\alpha_M$, $B_M = \beta_M/(1-\alpha_M)$
  Let $ratio = 1$
  for $\sigma_i \in S, i \in \{1, \ldots, M\}$ **do**
    Let $ratio_i = 1$
    for $j \in \{1, \ldots, N\}$ **do**
      $\omega \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma_i)$
      if $\omega \models \varphi$ **then**
        $ratio \leftarrow \frac{p_1}{p_0} ratio$; $ratio_i \leftarrow \frac{p_1}{p_0} ratio_i$
      end
      else
        $ratio \leftarrow \frac{1-p_1}{1-p_0} ratio$; $ratio_i \leftarrow \frac{1-p_1}{1-p_0} ratio_i$
      end
      if $ratio \leq A \vee ratio_i \leq A_M$ **then**
        Accept H and quit
      end
      if $ratio_i \geq B_M$ **then**
        Reject H (given budget) and quit
      end
    end
  end
  $R' : \{1, \ldots, |S|\} \to S$ is an injective function s.t.
  $\forall (n, \sigma), (n', \sigma') \in R',\ n > n' \Rightarrow R(\sigma) \geq R(\sigma')$
  $S \leftarrow \{\sigma \in S \mid \sigma = R'(n) \wedge n \in \{\lfloor |S|/2 \rfloor, \ldots, |S|\}\}$
end
Inconclusive result (given budget)

CSMA 3 4	θ	0.5	0.8	0.85	0.86	0.9	0.95
	Time (s)	0.5	3.5	737	*	2.9	2.5
CSMA 3 6	θ	0.3	0.4	0.45	0.48	0.5	0.8
	Time (s)	1.3	5.2	79	*	39	2.6
CSMA 4 4	θ	0.5	0.7	0.8	0.9	0.93	0.95
	Time (s)	0.2	0.3	4.0	8.6	*	3.8
WLAN 5	θ	0.1	0.15	0.18	0.2	0.25	0.5
	Time(s)	0.8	2.6	*	2.9	2.9	1.3
WLAN 6	Time(s)	1.3	2.2	*	6.5	1.3	1.3

Table 5.1: Hypothesis test results for CSMA/CD and WLAN protocols. Time is simulation time to achieve the correct result on a single machine. Asterisks denote true probabilities.

Chapter 6

SMC for Stochastic Adaptive Systems

6.1 Context

Critical systems increasingly rely on dynamically adaptive programs to respond to changes in their physical environments. Reasoning about such systems require to design new verification techniques and formalisms that take this model of reactivity into account [Che09].

This chapter proposes a complete formalism for the rigorous design of stochastic adaptive systems (SAS), whose components' behaviors and environment changes are represented via stochastic information. Adding some stochastic feature to components' models is more realistic, especially regarding the environment aspect, e.g. the probability of hardware failure, the fire frequency in a forest or a growing city population...

We view the evolution of our system as a sequence of views, each of them representing a topology of the system at a given moment of time. In our setting, views are represented by a combination of Markov chains, and stochastic adaptive transitions that describe the environment evolution as transitions between different views of the SAS (e.g. adding or removing components). Each view thus associates the new environment behaviour and a new system configuration (a new topology, addition or suppression of system components...). The incremental design is naturally offered to the system architect who can extend easily an existing model by creating new views.

As in previous chapters, properties of views can be specified with BLTL. To reason about sequences of view, we propose Adaptive BLTL (A-BLTL) that is an extension of BLTL with an adaptive operator to reason about the dynamic change of views. We also show that the formalism extend to contracts [SPE08, Mey92, DCL10] that permit to reason about both assumptions and guarantees of the system.

Consider the system described in Fig. 6.1. This system is composed by three different views linked by adaptive transitions (represented by dashed arrows). Each view contains some system components in a particular topology denoting a config-

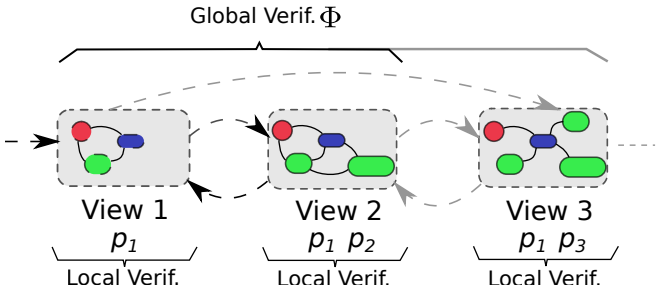


Figure 6.1: Illustration of SAS Methodology

uration of the SAS. Each local property p_1, p_2, p_3 is attached to one or more views. There is also global property Φ. The SAS is initially designed by View 1 and View 2 and the black dashed arrows. Properties p_1, p_2 are validated for the corresponding views and Φ is validated against this complete initial version of the system. To fit with the upcoming settings of the system, the system architect updates the model by adding View 3 with new adaptive transitions (in grey). This requires to only validate p_1 and p_3 against View 3 and to validate again the global property Φ against the system including all the three views.

We propose a new Statistical Model Checking (SMC) [YCGS05, SVA05] algorithm to compute the probability for a SAS to satisfy an A-BLTL property. One of the key points to implement an SMC algorithm is the ability to bound a priori the size of the simulation, which is an important issue. Indeed, although the SAS can only spend a finite amount of time in a given view, the time bound is usually unknown and simulation cannot be bounded. To overcome the problem, we expand on the work of Clarke [HCZ11] and consider a combination of SMC and model checking algorithm for untimed systems.

As a second contribution, we propose high-level formalisms to represent both SAS and A-BLTL/contracts. The formalism used to specify SAS relies on an extension of the Reactive Module Language (RML) used by the popular Prism toolset [KNP11]. Properties are represented with an extension of the Goal and Contract Specification Language (GCSL) [ABL13] defined in the DANSE IP project [DAN13]. This language offers English-based pattern to reason about timed properties without having to learn complex mathematics inherent to any logic.

Finally, as a last contribution, we have implemented our work in Plasma Lab. The implementation has been tested on a realistic case study defined with industry partners of DANSE.

6.2 Modeling Stochastic Adaptive Systems

In this section, we present the formal model used to encode behaviors of adaptive systems. In our setting, a view of a system is represented by the combination of several views, each of them being a DTMC (resp. CTMCs). We can compute the parallel composition $C_1|C_2$ of two DTMCs (resp. CTMCs) defined over the

same alphabet Σ. Let $(S_1, \overline{q}_1, \Sigma, \rightarrow_1, AP_1, L_1)$ and $(S_2, \overline{q}_2, \Sigma, \rightarrow_2, AP_2, L_2)$ be the two underlying transition systems. We first compute their parallel composition, which is a labelled transition system $(S, \overline{q}, \Sigma, \rightarrow, AP, L)$, where $S = S_1 \times S_2$, $\overline{s} = (\overline{s}_1, \overline{s}_2)$, $AP = AP_1 \cup AP_2$, $L(s) = L_1(s_1) \cup L_2(s_2)$ and the transition relation $\rightarrow$ is defined according to the following rule:

$$\frac{s_1 \xrightarrow{a}_1 s_1' \quad s_2 \xrightarrow{a}_2 s_2'}{(s_1, s_2) \xrightarrow{a} (s_1', s_2')} \tag{6.1}$$

In case of DTMCs, the new transition probability matrix is such that $\mathbf{P}((s_1, s_2), (s_1', s_2')) = \mathbf{P}(s_1, s_1') * \mathbf{P}(s_2, s_2')$, and in case of CTMCs the new transition rate matrix is such that $\mathbf{R}((s_1, s_2), (s_1', s_2')) = \mathbf{R}(s_1, s_1') * \mathbf{R}(s_2, s_2')$. DTMCs with different alphabets can also be composed and they synchronize on common actions. However, if both DTMCs can perform a non synchronized action, a uniform distribution is applied to resolve the non determinism. In case of CTMCs, the two actions are in concurrence, such that if $s_1 \xrightarrow{a}_1 s_1'$ with $a \notin \Sigma_2$, then $(s_1, s_2) \xrightarrow{a} (s_1', s_2)$ and $\mathbf{R}((s_1, s_2), (s_1', s_2)) = \mathbf{R}(s_1, s_1')$. In what follows, we denote by $Sys = C_1|C_2|\ldots|C_n$ the DTMC (resp. CTMC) that results from the composition of the components $C_1, C_2, \ldots, C_n$.

We are now ready to define our representation. An adaptive system consists in several successive views. It starts in an initial view that evolves until it reaches a state in which an adaptation is possible. This adaptation consists in a view change that depends on a probability distribution that represents uncertainty of the environment.

Definition 6.1. A SAS is a tuple $(\Delta, \Gamma, S, \overline{sys}, \rightsquigarrow)$ where:

- $\Delta = \{C_1, C_2, \ldots, C_n\}$ is a set of DTMCs (resp. CTMCs) that are the components of the SAS.
- Γ is the set of views of the SAS, such that each view is a stochastic system obtained from the parallel composition some components from Δ.
- $\overline{sys} \in \Gamma$ is the initial view.
- S is the set of states of the SAS. S is the union of the states of each view in Γ, *i.e.* for each state $s \in S$ there exists $\{C_1, C_2, \ldots, C_k\} \subseteq \Delta$ such that $s \in S_1 \times S_2 \times \cdots \times S_k$ (where $\forall i, 1 \leq i \leq k$, S_i is the set of states of C_i).
- $\rightsquigarrow \subseteq S \times [0,1]^S$ is a set of adaptive transitions.

Observe that the number of components per state may vary. This is due to the fact that different views may have different components. Observe also that it is easy to add new views to an existing adaptive system without having to re-specify the entire set of views. An element $(s, \mathbf{p}) \in \rightsquigarrow$ consists in a state s from a view $sys \in \Gamma$ and a probability distribution $\mathbf{p}$ over the states in S. When $s \neq s'$, we denote $s \rightsquigarrow s'$ if

there exists $\mathbf{p}$ such that $(s, \mathbf{p}) \in \rightsquigarrow$ and $\mathbf{p}(s') > 0$, which means that state s can be adapted into state s' with probability $\mathbf{p}(s')$.

An execution ω in a SAS is either a finite combination of n executions $\omega = \omega_0\omega_1 \ldots \omega_n$, such that for all $0 \leq i \leq n-1$, $\omega_i = (s_{0i}, t_{0i}), (s_{1i}, t_{1i}), \ldots (s_{li}, t_{li})$ is a finite execution of C_i, and $s_{li} \rightsquigarrow s_{0i+1}$, and $t_{0i+1} = 0$, and ω_n is a finite or infinite execution. Otherwise ω may be an infinite combination of finite executions $\omega = \omega_0\omega_1 \ldots$ that satisfy for all i the same constraints.

6.3 A logic for SAS properties

6.3.1 Probabilistic Adaptive Bounded Linear Temporal Logic

We consider quantitative verification of dynamic properties that are expressed via a quantitative extension of the Adaptive Linear Temporal Logic (A-LTL) proposed in [ZC06a]. Our logic, which we call Adaptive Bounded Linear Temporal Logic (A-BLTL), relies on an extension of BLTL combined with an adaptive operator. Although the logic is not strictly more expressive than BLTL, it is more suitable to describe properties of individual views, as well as global properties of the adaptive system, and it allows to develop specific algorithms for these properties. In the last part of the section, we also show how one can define contracts on such logic, where a contract [Mey92] is a pair of assumptions/guarantees that must be satisfied by the system.

We now introduce an adaptive operator in the spirit of [ZC06b]. The new logic A-BLTL is an extension of BLTL with an adaptive operator $\Phi \Rightarrow \Omega_{\leq k}\Psi$, where Φ is a BLTL formula, Ψ is an A-BLTL formula, Ω is a predicate over the states of different views of the SAS, and k is a time bound that limits the execution time of the adaptive transition. We will also consider unbounded versions of the adaptive operator.

Definition 6.2.[A-BLTL semantics] Let $\Phi \Rightarrow \Omega_{\leq k}\Psi$ be an A-BLTL formula and $\omega = (s_0, t_0), (s_1, t_1), \ldots$ be an execution of the SAS:

$$\omega \models \Phi \Rightarrow \Omega_{\leq k}\Psi \equiv \exists i,\ i = min\{j \mid t_0 \leq t_{j-1} \leq t_0 + k \wedge \omega_{|j} \models \Phi \wedge s_{j-1} \rightsquigarrow s_j \wedge \Omega(s_{j-1}, s_j)\} \wedge \omega^i \models \Psi \quad (6.2)$$

The property is unbounded if $k = \infty$, and in that case we write $\Phi \Rightarrow \Omega\Psi$.

According to Definition 6.2, an execution $\omega = (s_0, t_0), (s_1, t_1), \ldots$ satisfies an adaptive formula $\Phi \Rightarrow \Omega_{\leq k}\Psi$ if and only if there exists a minimal prefix of ω that satisfies Φ and reaches a state s_{j-1}, such that $\Omega(s_{j-1}, s_j)$ is satisfied, and such that the suffix of ω from state s_j satisfies Ψ. Therefore to satisfy this formula it is necessary to observe an adaptation compatible with Ω. We relax this constraint by introducing a new operator $\Phi \rightarrow \Omega_{\leq k}\Psi$, for which an adaptation is not necessary but triggers a check of Ψ when it happens. It is equivalent to the following formula: $\Phi \rightarrow \Omega_{\leq k}\Psi \equiv (\Phi \Rightarrow \Omega_{\leq k} true) \Rightarrow (\Phi \Rightarrow \Omega_{\leq k}\Psi)$.

We finally introduce stochastic contracts, that are used to reason about both the adaptive system and its environment via assumptions and guarantees.

Definition 6.3.[Contracts for SAS] A *contract* is defined as a pair (A, G), where A and G are respectively called the Assumption and the Guarantee. A SAS $\mathcal{M}$ satisfies the contract (A, G) iff $\forall \omega,\ \omega \models A \Rightarrow \omega \models G$, where ω is a execution of $\mathcal{M}$ and $\omega \models A$ (resp. G) means the execution ω satisfies the assumption A (resp. the guarantee G). In that case we write $\mathcal{M} \models (A, G)$.

6.3.2 Verifying SAS Properties using SMC

In this work, we will use the Monte Carlo approach to estimate probabilities. Hoqwever, other techniques such as SPRT or rare events approaches can also be used. The model checking of A-BLTL is rather evident. Given an A-BLTL property $\Phi_1 \Rightarrow \Omega_{\leq k}\Phi_2$, the monitor will first check whether the run satisfies Φ_1 using classical runtime verification techniques. If no, then the property is not satisfied. If yes, then one checks whether there is a pair of two successive states between t_0 and $t_0 + k$ that satisfies Ω. The latter is done by parsing the run. If this pair does not exist, then the property is not satisfied. Else we start a new monitor from the suffix of the run starting in the second state of the pair in order to verify Φ_2.

6.3.3 Verifying unbounded SAS Properties using SMC

We propose a method inspired by [HCZ11] to check unbounded A-BLTL properties. The principle is to combine a reachability analysis by model checking the underlying finite-state machine, with a statistical analysis of the stochastic model using the algorithms introduced previously.

We consider an A-BLTL property $\Phi \Rightarrow \Omega\Psi$, where Φ and Ψ are BLTL formulas. We first consider the reachability problem with objective $G = \{s \mid \exists s'.s \rightsquigarrow s' \wedge \Omega(s, s')\}$. The preliminary to the statistical analysis is to compute the set $Sat(Reach(G))$, that is all the states of the SAS that may eventually reach a state in G. This can be computed using classical model checking algorithms for finite-state machines. Only the underlying automata of the DTMCs or CTMCs of the SAS is used for this analysis.

Once this preliminary computation is performed, the CHECK algorithm from Fig. 14 is used to monitor the runs of the SAS. The algorithm takes as input the A-BLTL property and a run ω. The run should be in general infinite as there is no bound on the length of the runs that satisfied an unbounded A-BLTL property. In that case the states would be generated on-the-fly. The algorithm returns true or false, whether the run satisfied the property. We also denote CHECK(Φ, ω) as the monitoring of the BLTL property Φ. Then the first step on line 2 is to monitor Φ on the run ω. If the result is false then the property $\Phi \Rightarrow \Omega\Psi$ is not satisfied. Otherwise the algorithm searches through ω for two states *prec* and *curr* such that $\Omega(prec, curr)$ is true. This is possible if *curr* belongs to the precomputed set

$Sat(Reach(G))$. If Ω is satisfied the last step on line 10 is to monitor Ψ from the current position in ω.

For homogeneous Markov chains (with constant probability matrices, as it is the assumption in this paper), the algorithm almost surely (with probability 1) terminates, since it either reaches a state where Ω is unreachable, or the probability to reach two states that satisfy Ω is not null. It can be iterated to check sequences of adaptive operators, that is to say properties where Ψ is also an unbounded A-BLTL.

Algorithm 10: Check($\Phi \Rightarrow \Omega\Psi, \omega$) : Algorithm to monitor unbounded A-BLTL properties

```
1  if ¬CHECK(Φ, ω) then
2  |  return false
3  end
4  i ← 0, prec ← null, curr ← null
5  while true do
6  |  prec ← curr, curr ← ω[i]
7  |  if curr ∉ Sat(Reach(G)) then
8  |  |  return false
9  |  end
10 |  if Ω(prec, curr) then
11 |  |  return CHECK(Ψ, ω^i)
12 |  end
13 |  i ← i + 1
14 end
```

6.4 A software engineering point of view

In this section, we propose high level formalisms to specify both adaptive systems and their properties. Then, we define semantics of those formalisms by exploiting the definitions introduced in the previous sections. This gives us a free verification technology for them. The situation is illustrated in Figure 6.2.

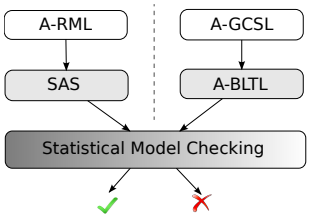


Figure 6.2: SAS verification flow

6.4.1 Adaptive RML systems as a high level formalism for SAS

We represent adaptive systems with Adaptive Reactive Module Language (A-RML), an extension of the Reactive Module Languages (RML) used by the PRISM toolset [KNP11]. Due to space limit, the syntax common to RML and A-RML is only briefly described here [1].

The RML language is based on the synchronisation of a set of modules defined by the user. A module is declared as a DTMC or CTMC, *i.e.* some local variables with a set of guarded commands. Each command has a set of actions of the form λ_i:$\mathtt{a}_i$ where λ_i is the probability (or the rate) to execute $\mathtt{a}_i$. A-RML extends RML such that each module can have some parameters in order to define its initial state.

```
module MOD_NAME(<Parameters>)
   <local_vars>
      ...
   [chan] g_k -> (λ_0:a_0) + ... + (λ_n:a_n);
      ...
endmodule
```

The optional channel identifiers prefixing commands are used to strongly synchronise the different modules of a RML system. A module is synchronised over the channel `chan` if it has some commands prefixed by `chan`. We say a command is independent if it has no channel identifier.

In A-RML a system is a set of modules and global variables. The modules synchronise on common channels such that the system commands are the independent commands of each module and the synchronised commands. A command synchronised on `chan` forces all the modules that synchronised over `chan` to simultaneously execute one of their enabled commands prefixed by `chan`. If one module is not ready, *i.e.* it has no enabled commands for `chan`, the system has no enabled command over `chan`. Similarly to a module, if the system reaches a state with a non-deterministic choice, it is solved by a stochastic behaviour based on a uniform distribution. This solution allows to execute the A-RML system in accordance with the DTMC/CTMC models.

```
system SYS_NAME(<Parameters>)
   <global_variables>
   ...
   <module_declarations>
   ...
endsystem
```

An adaptive system consists in a set of different views, each represented by an A-RML system, and a list of adaptations represented by adaptive commands. The `adaptive` environment is used to specify which one is the initial view and what adaptations are possible.

```
adaptive
   init at SYS_NAME(<Initial values>)
```

[1] The full syntax can be found at http://project.inria.fr/plasma-lab/

```
    ...
    { SYS_NAME | g_k } -> λ_0:{a_0} + ... + λ_n:{a_n};
    ...
endadaptive
```

An adaptive command is similar to module command. It has a guard $\mathtt{g}_k$ that applies to the current view `SYS_NAME`, and a set of actions $\lambda_i{:}\mathtt{a}_i$ where λ_i is the probability (or the rate) to execute action $\mathtt{a}_i = \mathtt{SYS\_NAME'}(\mathtt{e}_0, \ldots, \mathtt{e}_m)$. This action defines the next view `SYS_NAME'` after performing the adaptation. This view is determined according to the states of the previous view by setting the parameters of `SYS_NAME'` with the expressions $\mathtt{e}_0, \ldots, \mathtt{e}_m$ evaluated over `SYS_NAME`. The execution of the adaptive command is done in accordance with the SAS semantics.

Theorem 6.4. The semantics of A-RML can be defined in terms of SAS.

The proof of the above theorem is a direct consequence of the fact that semantics of RML is definable in terms of composition of MC, and that the definition of an adaptive command can also be represented as a MC.

6.4.2 A contract language for SAS specification

The Goal and Contract Specification Language (GCSL) was first proposed in [ABL13] to formalise properties of adaptive systems in the scope of the DANSE project. It has a strong semantics based on BLTL but it has a syntax close to the hand written English requirements. Dealing with formal temporal logic is often an issue to formalise correctly the initial English requirements. Most of the time the formalisation frequently contains some mistakes, which is due to the nesting of the temporal operators. The difficulty for correctly specifying properties is enough to make the overall methodology useless.

The GCSL syntax combines a subset of the Object Constraint Language (OCL) [OMG10] (used to define state properties, *i.e.* Boolean relations between the system components) and English behavioural patterns used to express the evolution of these state properties during the execution of the system. The usage of OCL is illustrated in Example 6.5.

Example 6.5. We consider a SAS describing the implementation of an emergency system in a city. The city area is divided as a set of districts where each district may have some equipment to fight against the fire, e.g. some fire stations with fire brigades and fire fighting cars. Each district is also characterised by a risk of fire and the considered damages are mainly related to the population size of each district. The requirement *"Any district cannot have more than 1 fire station, except if all districts have at least 1"* ensures the minimal condition for the equipment distribution in the city. We use syntactic coloration to make the difference between the parts of the language used in the property: the words in red are identifiers from the model, the blue part is from OCL, like collection handling, and the black words are variables:

City.itsDdistricts→exists(district | district.**ownedFireStations** > 1**) implies**
City.itsDistricts→forAll(district | district.**ownedFireStations** ≥ 1**)**

GCSL patterns are used to specify temporal properties. In this section we only present a subset of such patterns that is considered to be general enough to specify properties of a large set of industry-examples from the DANSE project. After having read this section, the user shall understand that the set can be easily increased. Each pattern can nest one or more state properties, denoted in the grammar by the non-terminals `<OCL-prop>` and `<arith-rel>`, that respectively denote a state property written in OCL or an arithmetic relation between the identifiers used in the model. The non-terminal `<int>` denotes a finite time interval over which the temporal pattern is applied, and `<N>` is a natural number. The patterns can be used directly or combined with OCL: applying a pattern to a collection of system components defines a behavioural property that is applied to each element of the collection. We present below an excerpt of the complete GCSL grammar available in [ABL13]:

```
<GCSL> ::= <OCL-coll>->forAll(<variable>| <pattern>)
| <OCL-coll>->exists(<variable>| <pattern>)
| <OCL-prop>
| <pattern>

<pattern> ::= whenever [<prop>] occurs [<prop>] holds during following [<int>]
| whenever [<prop>] occurs [<prop>] implies [<prop>] during following [<int>]
| whenever [<prop>] occurs [<prop>] does not occur during following [<int>]
| whenever [<prop>] occurs [<prop>] occurs within [<int>]
| [<prop>] during [<int>] raises [<prop>]
| [<prop>] occurs [<N>] times during [<int>] raises [<prop>]
| [<prop>] occurs at most [<N>] times during [<int>]
| [<prop>] during [<int>] implies [<prop>] during [<int>] then [<prop>] during [<int>]

<prop> ::= <OCL-prop> | <arith-rel>
```

Example 6.6. Consider the following requirement about the model described in Example 6.5: *"The fire fighting cars hosted by a fire station shall be used all simultaneously at least once in 6 months"*. This requirement uses both GCSL and OCL patterns:
City.itsFireStations→forAll(fStation | **Whenever** [fStation.**hostedFireFightingCars** → **exists(**ffCar | ffCar.**isAtFireStation)**] **occurs,** [fStation.**hostedFireFightingCars→forall(**ffCar | ffCar.**isAtFireStation = false)**] **occurs within** [**6 months**]**)**

We now propose A-GCSL, a syntax extension for GCSL that can be used to describe adaptive requirements of SAS. A-GCSL extends the GCSL grammar by adding a new pattern that allows to express adaptive relations as done with the two adaptive operators defined in Section 6.3. The first pattern of `<dyna-spec>` is equivalent to the operator $\Rightarrow \Omega$ and the second one denotes $\rightarrow \Omega$. Any adaptive requirement has three elements (A, Ω, G) that are called assumption, trigger and guarantee, respectively. The assumption and guarantee are specified in GCSL, whereas the trigger is in OCL. The syntax allows to compose the patterns by specifying the guarantee with an adaptive pattern. For instance, a composed requirement of the form `if` Φ_1 `holds and for all rule that satisfies` Ω `then (if` Φ_2 `holds and for all rule that satisfies` Ω' `then` Φ_3 `holds) holds` is equivalent to the property $\Phi_1 \Rightarrow \Omega\Phi_2 \Rightarrow \Omega'\Phi_3$. The A-GCSL grammar is the following:

```
<dyna-spec> ::= if [<GCSL>] holds and for all rule that satisfies [<prop>]
                then ( <GCSL> | <dyna-spec> ) holds
| if [<GCSL>] holds then there exists a rule satisfying [<prop>]
                and ( <GCSL> | <dyna-spec> ) holds
```

Example 6.7. Consider again the system in Example 6.5 and the following A-GCSL requirement:
if [**City.underFire = 0**] **holds and for all rule such that rule satisfies** [**City.underFire** $\geq$ **3**] **then** [**City.itsDistricts**$\rightarrow$**forall(**district | district.**decl = false** $\Rightarrow$ **whenever** [district.**decl = true**] **occurs,** [district.**fire = 0**] **occurs within** [**50 hours**] **)**] **holds**

The attribute `underFire` denotes the number of districts in which a fire has been declared. If there are more than three fires in the city, then the fire stations change their usual emergency management into a crisis one. When such management is activated, the firemen have 50 hours to fix the problem. The requirement can be translated in A-GCSL using the following formula:

$$\Phi_6 = \big(underFire = 0\big) \rightarrow underFire \geq 3_{\leq 10000} \bigwedge_{d_i:district} \Big(\neg d_i.decl \Rightarrow G_{\leq 10000}\ (d_i.decl \Rightarrow F_{\leq 50}\ d_i.fire = 0)\Big)$$

In [ABL13], we have showed that any GCSL pattern can be translated into a BLTL formula. The result extends as follows.

Theorem 6.8. Any A-GCSL pattern can be translated into an A-BLTL property.

This result is an immediate consequence of the definition of the adaptive pattern.

6.5 Experiments with SAS

Our work has been implemented in Plasma Lab. Here, we describe the experiments on a large-size case study.

6.5.1 CAE model

Together with our industrial partners in the DANSE project, we have developed the Concept Alignment Example (CAE). The CAE is a fictive adaptive system example inspired by real-world Emergency Response data to a city fire. It has been built as a playground to demonstrate new methods and models for the analysis and visualization of adaptive systems designs.

The CAE describes the organization of the firefighting forces. We consider in our study that the city is initially divided into 4 districts, and that the population might increase by adding 2 more districts. Different and even more complex examples can be built using the components of this design.

A fire station is assigned to the districts, but as the fire might spread within the districts, the system can adapt itself by hiring more firemen. We can therefore design a SAS with three views as described in Fig. 6.3.

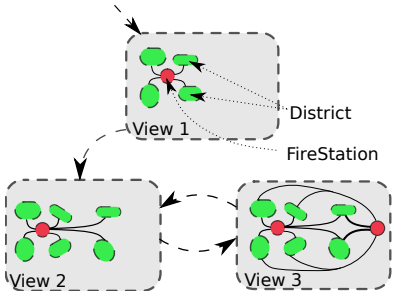


Figure 6.3: Components and Views in the CAE model

Adaptive transitions exist between these views to reflect changes in the environment and adaptations of the system. Initially in View 1, the system can switch to View 2 when the population of the city increase. This change models an uncertainty of the environment, and for the purpose of this study we fix its probability to 0.01 Then, if the number of fires becomes greater than 2, the system adapts itself by switching to View 3. If the number of fires eventually reduces and becomes lower than 2, the system might return to View 2. Again, as this change is uncertain, we fix its probability to 0.8.

We design several A-RML models of the system that consist in two types of modules: `District` and `FireStation`, both based on a CTMC semantics. First, we study a model `AbstractCAE` that is an abstract view of the SAS. In this model, the `District` module, presented below, is characterized by a constant parameter `p`, that determines the probability of fire, and by two Boolean variables `decl` and `men`, that respectively defines if a fire has been declared and if the firemen are allocated to the district. The module `fireStation` has one constant parameter $\texttt{distance}_\texttt{i}$ for each module of the system. This parameter determines the probability to react at a fire, such that the greater the distance, the lower the probability. However a fire station can only treat one fire at a time, which is encoded with a Boolean variable `allocated`. The fire stations and the districts synchronize on channels `allocate` and `recover`, that respectively allocate firefighters to the district and bring them back when the fire is treated. The different views are constructed by instantiating and renaming the modules presented above.

```
module District( const int p )
  decl : bool init false;
  men: bool init false;
  [] !decl -> p/1000: (decl'=true);
  [allocate] decl & !men -> (men'=true);
  [recover] decl & men -> 1/p: (decl'=false) & (men'=false);
endmodule
```

We refine this model to better encode the behaviour of the SAS. In this new model `ConcreteCAE` a new variable `fire` of module `District` ranges from 0 to 10 and grades the intensity of the fire. The fire stations can now assign several cars (from 0 to 5) to each districts. Therefore the variables `men` and `allocated` becomes integers.

```
module District( const int p )
  fire : [0..10] init 0;
  decl : bool init false;
  men: [0..5] init 0;
  [] fire=0 -> p/1000: (fire'=1);
  [] fire>0 & fire<10 -> p/((1+men)*100): (fire'=fire+1);
  [] fire>0 & !decl -> (fire*fire)/10: (decl'=true);
  [allocateSt1] decl & fire>0 -> (fire*fire)/10: (men'=men+1);
  [allocateSt2] decl & fire>0 -> (fire*fire)/10: (men'=men+1);
  [] men>0 & fire>0 -> men/10: (fire'=fire-1);
  [recover] decl>0 & fire=0 -> 1000: (men'=0)&(decl'=false);
endmodule
```

From the two models we can consider several subparts composed by one or several views of the SAS. Adaptive commands are used to model the transitions between the different views.

- `AbstractCAE_1` consists in View 1 and 2 from model `AbstractCAE`.
- `AbstractCAE_2` consists in View 2 and 3.
- `AbstractCAE_3` has the same views as `AbstractCAE_2` but is initiated in View 3 instead of View 2.
- `ConcreteCAE_1` only consists in View 1 from model `ConcreteCAE`.
- `ConcreteCAE_2` only consists in View 2.
- `ConcreteCAE_3` only consists in View 3.
- `ConcreteCAE_Full` is the full model of `ConcreteCAE`, with the 3 views and all the adaptive transitions between them.

6.5.2 Checking requirements

The requirements are expressed in A-GCSL and translated to A-BLTL. We first check the model `AbstractCAE` against A-BLTL properties with adaptive operators. Our goal is to verify that the transitions between the different views of the system occurs and satisfy some properties.

The first property, *if* [`true`] *holds then there exists a rule satisfying* [`underfire` $\leq$ `1`] *and Always* [!`maxfire`], checks that when the system is in View 1, it eventually switches to View 2 when the number of districts that have declared a fire (`underfire`) is still lower than 1, and that as a result the system remains safe for a limited time period, *i.e.* the number of districts that have declared a fire is not maximum (`maxfire` is false). To check this property we limit the analysis to the model `AbstractCAE_1` with only View 1 and View 2. The A-GCSL property is translated in an A-BLTL formula: $\Phi_1 =$ `true` $\Rightarrow$ `underfire` $\leq$ `1`$G_{\leq 1000}$!`maxfire`, and the results in Table 6.1 show that the probability to satisfy the property is only 50%. This justify the need to add a second fire station, as in View 3.

The second property, *if* [`true`] *holds then there exists a rule satisfying* [`true`] *and Always* [!`maxfire`], checks that from View 2 a second fire station is quickly added, which switches the system to View 3, and that then the system is safe. The property is checked on the model `AbstractCAE_2` using the A-BLTL formula : $\Phi_2 = \texttt{true} \Rightarrow \texttt{true}_{\leq 100} G_{\leq 10000}\ \texttt{!maxfire}$.

Finally, with the property *if* [`true`] *holds then there exists a rule satisfying* [`true`] *and* [`true`], we check that from View 3 the system eventually returns to View 2. Therefore we use the model `AbstractCAE_3` that starts in View 3 and we check the A-BLTL formula $\Phi_3 = \texttt{true} \Rightarrow \texttt{true}_{\leq 100}\texttt{true}$.

The `AbstractCAE` models are simple enough to be able to perform reachability analyses and check the unbounded A-BLTL properties presented above using Algorithm 14. In a second step we consider the models `ConcreteCAE` to better evaluate the safety of the system. The state spaces of these models contain several millions of states, and therefore, they can only be analyzed by purely SMC algorithms. We verify the two following properties:

- *Always* !`maxfire`, to check that the maximum of fire intensity of 10 is never reached in any district. This corresponds to $\Phi_4 = G_{\leq 10000}\ \texttt{!maxfire}$.
- *Whenever* [`fire` $>$ `0`] *occurs* [`fire` $=$ `0`] *within* [50 *hours*], to check that a fire in a district is totally extinct within 50 hours. This corresponds to $\Phi_5 = G_{\leq 10000}\big(\texttt{d6.fire} > 0 \Rightarrow F_{\leq 50}\ \texttt{d6.fire} = 0\big)$.

These two properties are first checked for each view of the system. The results in Table 6.1 show that while View 1 and View 3 are surely safe, View 2 is frequently unsafe. But when we check these properties on the complete adaptive model `ConcreteCAE_Full`, with the three views, we can show that the system remains sufficiently safe. It proves that after a change of the environment (the increase of population) the system is able to adapt itself to guaranty its safety.

In the last experiment of Table 6.1 we check the A-GCSL property presented in Example 6.7. This bounded adaptive A-GCSL property is checked using the full `ConcreteCAE` model.

We have performed each experiment in Plasma Lab with a confidence $\delta = 0.01$ and an error bound $\varepsilon = 0.02$. The results in Table 6.1 give the probabilities estimation and the time needed to perform the computation.

6.6 Conclusion

This paper presents a new methodology for the rigorous design of stochastic adaptive systems. Our model is general, but the verification procedure can only reason on a finite and known set of views. Our formalism is inspired from [ZC06b], where both the stochastic extension and high level formalisms are not considered. In future work, we will extend this approach to purely dynamic systems. Another objective is to extend the work to reason about more complex properties such as energy consumption.

Property	CAE Model	Estimation interval	Consumed Time
Φ_1	`AbstractCAE_1` View 1, View 2	$[0.53, 0.56]$	1351s
Φ_2	`AbstractCAE_2` View 2, View 3	$[0.84, 0.86]$	11s
Φ_3	`AbstractCAE_3` AbstractCAE_2 starting from View 3	$[0.98, 1]$	1363s
Φ_4	`ConcreteCAE_6` 4 dist. 1 sta.	$[0.95, 0.99]$	11s 9s
Φ_4 Φ_5	`ConcreteCAE_2` 6 dist. 1 sta.	$[0.46, 0.5]$ $[0.21, 0.25]$	15s 13s
Φ_4 Φ_5	`ConcreteCAE_3` 6 dist. 2 sta.	$[0.98, 1]$ $[0.98, 1]$	30s 31s
Φ_4 Φ_5	`ConcreteCAE_Full` 4-6 dist. 1-2 sta.	$[0.89, 0.93]$ $[0.82, 0.86]$	25s 42s
Φ_6	`ConcreteCAE_Full` 4-6 dist. 1-2 sta.	$[0.47, 0.51]$	109s

Table 6.1: Experiments on CAE models

Chapter 7

Statistical model Checking for Priced timed stochastic automata

7.1 Introduction

As stated in the introduction of this thesis, there are several variants and extensions of MC aiming at handling real-time and hybrid systems with quantitative constraints on time, energy or more general continuous aspects [AD94, ACH+95, BFH+01, ATP01]. Within the field of embedded systems these formalisms and their supporting tools [SPI, SMV, UPP, Fre08] are now successfully applied to time- and energy-optimal scheduling, WCET analysis and schedulability analysis.

Compared with traditional approaches, a strong point of real-time model checking is that it (in principle) only requires a model to be applicable, thus extensions to multi-processor setting is easy. A weak point of model checking is the state-space explosion, i.e. the exponential growth in the analysis effort measured in the number of model-components. Another limitation of real-time model checking is that it merely provides – admittedly most important – hard quantitative guarantees, e.g. the worst case response time of a recurrent task under a certain scheduling principle, the worst case execution time of a piece of code running on a particular execution platform, or the worst case time before consensus is reached by a real-time network protocol. In addition to these hard guarantees, it would be desirable in several situations to obtain refined performance information concerning likely or expected behaviors in terms of timing and resource consumption. In particular, this would allow to distinguish and select between systems that perform identically from a worst-case perspective.

As a first contribution we propose a stochastic semantics for Priced Timed Automata (PTA), whose clocks can evolve with different rates, while[1] being used with no restrictions in guards and invariants. Networks of PTAs (NPTA) are created by composing PTAs via input and output actions. More precisely, we define a natural stochastic semantics for networks of NPTAs based on races between components being composed. We shall observe that such race can generate arbitrarily complex

[1] in contrast to the usual restriction of priced timed automata [BFH+01, ATP01]

stochastic behaviors from simple assumptions on individual components. We shall see that our semantics cannot be emulated by applying the existing stochastic semantic of [BBB+07, BBBM08] to the product of components. Other related work includes the very rich framework of stochastic timed systems of MoDeST [BDHK04]. Here, however, general hybrid variables are not considered and parallel composition does not yield fully stochastic models. For the notion of probabilistic hybrid systems considered in [TEF11] the choice of time is resolved non-deterministically rather than stochastically as in our case. Moreover, based on the stochastic semantics, we are able to express refined performance properties, e.g. in terms of probabilistic guarantees of time- and cost-bounded properties[2].

To allow for the efficient analysis of probabilistic performance properties we propose to work with Statistical Model Checking. One of the main contribution of the work is to provide an efficient implementation of several existing SMC algorithms that we use for checking the correctness of NPTAs with respect to a stochastic extension of cost-constrained temporal logic – this extension being conservative with respect to the classical (non-stochastic) interpretation of the logic. We shall observe that two timed bisimilar NPTAs may be distinguisable by PWCTL. The series of algorithms we implemented includes SPRT and Monte Carlo. Our implementation relies on a new efficient algorithm for generating runs of NPTAs in a random manner. In addition, we also propose another SMC algorithm to compare the probabilities of two properties without computing them individually – which is useful to compare the performances of a program with one of its evolutions at cheap cost. This probability comparison problem, which is far beyond the scope of existing time model checking approaches, can be approximated with an extension of the sequential hypothesis testing and has the advantage of unifying the confidence in the comparison. In addition to be the first to apply such extension in the context of formal verification, we also propose a new variant that allows to reuse existing results in parallel when comparing the properties on different timed bounds.

Finally, one of the most interesting contribution of our work takes the form of a series of new case studies that are analyzed with a new stochastic extension of Uppaal [DLL+11]. Particularly, we show how our approach can be used to resolve scheduling problems. Such problems are defined using Duration Probabilistic Automata (DPA) [MLK10], a new and natural model for specifying list of tasks and shared resources. We observe that our approach is not only more general, but also an order of magnitude faster than the hypothesis testing engine recently implemented in the Prism toolset. Our work thus presents significant advances in both the modeling and the efficient verification of network of complex systems.

Organisation of the chapter In Section 7.2, we introduce our model of timed stochastic automata together with an illustrative example. Section 7.3 proposes a new probabilistic semantics to the model as well as an algorithm to simulate executions stochastically. Sections 7.4 and 7.5 present our new statistical model checking algorithms, while Section 7.6 reports on experiments. Finally, Section 7.7

[2]Clocks with different rates can be used to model costs.

concludes the paper.

Related work. Some works on probabilistic semantics of timed automata have already been discussed above. Simulation-based approaches such as Monte Carlo have been in use since decades, however the use of simulation and hypothesis testing to reason on formal models is a more recent advance. First attempts to apply hypothesis testing on stochastic extension of Hennessy-Milner logic can be found in [LS89]. In [YS02b,You05a], Younes was the first to apply hypothesis testing to stochastic systems whose properties are specified with (bounded) temporal logic. His approach is implemented in the Ymer toolset [You05c] and can be applied on time-homogeneous generalized semi-Markov processes, while our semantics addresses the composition of stochastic systems allowing to compose a global system from components and reason about communication between independent processes. In addition to Younes work we explore continuous-time features, formalize and implement Wald's ideas where the probability comparison can be evaluated on NPTA processes. In a recent work [ZPC10], Zuliani et al. extended the SMC approach to hybrid systems. Their work is a combination of [JCL+09] and [CDL08] based on Simulink models (non-linear hybrid systems), whereas our method is specialised to networks of priced timed automata where model-checking techniques can be directly applicable using the same tool suite. In addition we provide means of comparing performances without considering individual probabilities. Finally, a very recent work [BFHH11b] proposes partial order reduction techniques to resolve non-determinism between components rather than defining a unique stochastic distribution on their product behaviors. While this work is of clear interest, we point out that the application of partial order may considerably increase the computation time and for some models partial orders cannot resolve non-determinism, especially when considering continuous time [Min99]. Finally, we mention [KSB10] that proposes a stochastic semantics to UPPAAL's models through simulation. This work does not consider race between components and offers no tool implementation.

7.2 Network of Priced Timed Automata

We consider the notion of *Networks of Priced Timed Automata (NPTA)*, generalizing that of regular timed automata (TA) in that clocks may have different rates in different locations. In fact, the expressive power (up to timed bisimilarity) of NPTA equals that of general linear hybrid automata (LHA) [ACH+95], rendering most problems – including that of reachability – undecidable.

Let X be a finite set of variables, called *clocks*[3]. A *clock valuation* over X is a mapping $\nu : X \to \mathbb{R}_{\geq 0}$, where $\mathbb{R}_{\geq 0}$ is the set of nonnegative reals. We write $\mathbb{R}^X_{\geq 0}$ for the set of clock valuations over X. Let $r : X \to \mathbb{N}$ be a *rate vector*, assigning to each clock of X a rate. Then, for $\nu \in \mathbb{R}^X_{\geq 0}$ and $d \in \mathbb{R}_{\geq 0}$ a delay, we write $\nu + r \cdot d$ for the clock valuation defined by $(\nu + r \cdot d)(x) = \nu(x) + r(x) \cdot d$ for any clock $x \in X$.

[3]We will (mis)use the term "clock" from timed automata, though in the setting of NPTAs the variables in X are really general real-valued variables.

We denote by $\mathbb{N}^X$ the set of all rate vectors. If $Y \subseteq X$, the valuation $\nu[Y]$ is the valuation assigning 0 when $x \in Y$ and $\nu(x)$ when $x \notin Y$. An *upper bounded (lower bound) guard* over X is a finite conjunction of simple clock bounds of the form $x \sim n$ where $x \in X$, $n \in \mathbb{N}$, and $\sim \in \{<, \leq\}$ ($\sim \in \{>, \geq\}$) We denote by $\mathcal{U}(X)$ ($\mathcal{L}(X)$ the set of upper (lower) bound guards over X, and write $\nu \models g$ whenever ν is a clock valuation satisfying the guard g. Let $\Sigma = \Sigma_i \uplus \Sigma_o$ be a disjoint sets of input and output actions.

Definition 7.1. A *Priced Timed Automaton* (PTA) is a tuple $\mathcal{A} = (L, \ell_0, X, \Sigma, E, R, I)$ where: (i) L is a finite set of locations, (ii) $\ell_0 \in L$ is the initial location, (iii) X is a finite set of clocks, (iv) $\Sigma = \Sigma_i \uplus \Sigma_o$ is a finite set of actions partitioned into inputs (Σ_i) and outputs (Σ_o), (v) $E \subseteq L \times \mathcal{L}(X) \times \Sigma \times 2^X \times L$ is a finite set of edges, (vi) $R : L \to \mathbb{N}^X$ assigns a rate vector to each location, and (viii) $I : L \to \mathcal{U}(X)$ assigns an invariant to each location.

The semantics of NPTAs is a timed labelled transition system whose states are pairs $(\ell, \nu) \in L \times \mathbb{R}^X_{\geq 0}$ with $\nu \models I(\ell)$, and whose transitions are either delay $(\ell, \nu) \xrightarrow{d} (\ell, \nu')$ with $d \in \mathbb{R}_{\geq 0}$ and $\nu' = \nu + R(\ell) \cdot d$, or discrete $(\ell, \nu) \xrightarrow{a} (\ell', \nu')$ if there is an edge (ℓ, g, a, Y, ℓ') such that $\nu \models g$ and $\nu' = \nu[Y]$. We write $(\ell, \nu) \rightsquigarrow (\ell', \nu')$ if there is a finite sequence of delay and discrete transitions from (ℓ, ν) to (ℓ', ν').

Networks of Priced Timed Automata Following the compositional specification theory for timed systems in [DLL+10a], we shall assume that NPTAs are: (1)[*Input-enabled*:] for all states (ℓ, ν) and input actions $\iota \in \Sigma_i$, for all TAs j, there is an edge $(\ell^j, g, \iota, Y, \ell^{j\prime})$ such that $\nu \models g$, (2) [*Deterministic*:] for all states (ℓ, ν) and actions $a \in \Sigma$, whenever $(\ell, \nu) \xrightarrow{a} (\ell', \nu')$ and $(\ell, \nu) \xrightarrow{a} (\ell'', \nu'')$ then $\ell' = \ell''$ and $\nu' = \nu''$, and (3) [*Non-zenos*:] time always diverge. Moreover, different automata synchronize on matching inputs and outputs as a standard broadcast synchronization [Góm09].

Whenever $\mathcal{A}^j = (L^j, X^j, \Sigma^j, E^j, R^j, I^j)$ ($j = 1 \dots n$) are NPTA, they are *composable* into a *closed network* iff their clock sets are disjoint ($X^j \cap X^k = \emptyset$ when $j \neq k$), they have the same action set ($\Sigma = \Sigma^j = \Sigma^k$ for all j, k), and their output action-sets provide a partition of Σ ($\Sigma^j_o \cap \Sigma^k_o = \emptyset$ for $j \neq k$, and $\Sigma = \cup_j \Sigma^j_o$). For $a \in \Sigma$ we denote by $c(a)$ the unique j with $a \in \Sigma^j$.

Definition 7.2. Let $\mathcal{A}^j = (L^j, X^j, \Sigma, E^j, R^j, I^j)$ (with $j = 1 \dots n$) be composable NPTAs. Their *composition* $(\mathcal{A}_1 \,|\, \dots \,|\, \mathcal{A}_n)$ is the NPTA $\boldsymbol{\mathcal{A}} = (L, X, \Sigma, E, R, L)$ where (i) $L = \times_j L^j$, (ii) $X = \cup_j X^j$, (iii) $R(\boldsymbol{\ell})(x) = R^j(\ell^j)(x)$ when $x \in X^j$, (iv) $I(\boldsymbol{\ell}) = \cap_j I(\ell^j)$, and (v) $(\boldsymbol{\ell}, \cap_j g_j, a, \cup_j r_j, \boldsymbol{\ell'}) \in E$ whenever $(\ell_j, g_j, a, r_j, \ell'_j) \in E^j$ for $j = 1 \dots n$.

Example 1. Let A, B, T and AB be the priced timed automata depicted in Fig. 7.1 [4] Then A, B and T are composable as well as AB and T. In fact the composite

[4]The broadcast synchronization we use allows us to ignore missing input transitions that may otherwise be added as looping transitions.

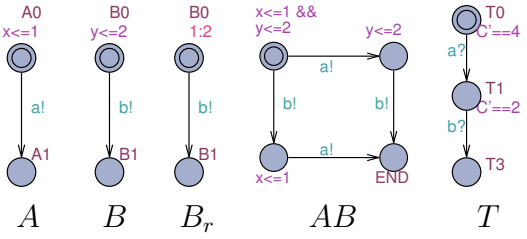


Figure 7.1: Three composable NPTAs: A, B and T; A, B_r and T; and AB and T.

systems $(A|B|T)$ and $(AB|T)$ are timed (and priced) bisimilar, both having the transition sequence:
$\big((A_0, B_o, T_0), [x = 0, y = 0, C = 0]\big) \xrightarrow{1}\xrightarrow{a!} \big((A_1, B_0, T_1), [x = 1, y = 1, C = 4]\big) \xrightarrow{1}\xrightarrow{b!} \big((A_1, B_1, T_2), [x = 2, y = 2, C = 6]\big)$,
demonstrating that the final location T_3 of T is reachable with cost 6.

7.3 Probabilistic Semantics of NPTA

Continuing Example 1 we may realise that location T_3 of the component T is reachable within cost 0 to 6 and within total time 0 and 2 in both $(A|B|T)$ and $(AB|T)$ depending on when (and in which order) A and B (AB) chooses to perform the output actions $a!$ and $b!$. Assuming that the choice of these time-delays is governed by probability distributions, we will in this section define a probability measure over sets of infinite runs of networks of NPTAs.

In contrast to the probabilistic semantics of timed automata in [BBBM08,BBB+07] our semantics deals with networks and thus with races between components. Let $\mathcal{A}^j = (L^j, X^j, \Sigma, E^j, R^j, I^j)$ $(j = 1 \ldots n)$ be a collection of composable NPTAs. Under the assumption of input-enabledness, disjointness of clock sets and output actions, states of the the composite NPTA $\boldsymbol{\mathcal{A}} = (\mathcal{A}_1 \,|\, \ldots \,|\, \mathcal{A}_n)$ may be seen as tuples $\mathbf{s} = (s_1, \ldots, s_n)$ where s_j is a state of $\mathcal{A}^j$, i.e. of the form (ℓ, ν) where $\ell \in L^j$ and $\nu \in \mathbb{R}^{X^j}_{\geq 0}$. Our probabilistic semantics is based on the principle of independency 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 minimum delay: the output is broadcast and all other components may consequently change state.

Probabilistic Semantics of NPTA Components Let us first consider a component $\mathcal{A}^j$ and let St^j denote the corresponding set of states. For each state $s = (\ell, \nu)$ of $\mathcal{A}^j$ we shall provide probability distributions for both delays and outputs. In this presentation, we restrict to uniform and universal distributions, but arbitrary distributions can be considered.

The *delay density function* μ_s over delays in $\mathbb{R}_{\geq 0}$ will be either a uniform or

an exponential distribution depending on the invariant of ℓ. Denote by E_ℓ the disjunction of guards g such that $(\ell, g, o, -, -) \in E^j$ for some output o. Denote by $d(\ell, \nu)$ the infimum delay before enabling an output, i.e. $d(\ell, \nu) = \inf\{d \in \mathbb{R}_{\geq 0} : \nu + R^j \cdot d \models E_\ell\}$, and denote by $D(\ell, \nu)$ the supremum delay, i.e. $D(\ell, \nu) = \sup\{d \in \mathbb{R}_{\geq 0} : \nu + R^j \cdot d \models I^j(\ell)\}$. If $D(\ell, \nu) < \infty$ then the delay density function μ_s is a uniform distribution on $[d(\ell, \nu), D(\ell, \nu)]$. Otherwise – that is $I^j(\ell)$ does not put an upper bound on the possible delays out of s – the delay density function μ_s is an exponential distribution with a rate $P(\ell)$, where $P : L^j \to \mathbb{R}_{\geq 0}$ is an *additional* distribution rate component added to the NPTA $\mathcal{A}^j$. For every state $s = (\ell, \nu)$, the *output probability function* γ_s over Σ_o^j is the uniform distribution over the set $\{o : (\ell, g, o, -, -) \in E^j \wedge \nu \models g\}$ whenever this set is non-empty [5]. We denote by s^o the state after the output of o. Similarly, for every state s and any input action ι, we denote by s^ι the state after having received the input ι.

Probabilistic Semantics of Networks of NPTA We shall now see that while the stochastic semantics of each PTA is rather simple (but quite realistic), arbitrarily complex stochastic behavior can be obtained by their composition.

Reconsider the closed network $\boldsymbol{\mathcal{A}} = (\mathcal{A}_1 \,|\, \ldots \,|\, \mathcal{A}_n)$ with a state space $\mathsf{St} = \mathsf{St}_1 \times \cdots \times \mathsf{St}_n$. For $\mathbf{s} = (s_1, \ldots, s_n) \in \mathsf{St}$ and $a_1 a_2 \ldots a_k \in \Sigma^*$ we denote by $\pi(\mathbf{s}, a_1 a_2 \ldots a_k)$ 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 $t_1, \ldots, t_n \in \mathbb{R}_{\geq 0}$, that is runs where the i'th action a_i has been outputted by the component $A_{c(a_i)}$. We now inductively define the following measure for such sets of runs:

$$\mathbb{P}_{\boldsymbol{\mathcal{A}}}\big(\pi(\mathbf{s}, a_1 \ldots a_n)\big) = \int_{t \geq 0} \mu_{s_c}(t) \cdot \Big(\prod_{j \neq c} \int_{\tau > t} \mu_{s_j}(\tau) d\tau\Big) \cdot \gamma_{s_c{}^t}(a_1) \cdot \mathbb{P}_{\boldsymbol{\mathcal{A}}}\big(\pi(\mathbf{s}^t)^{a_1}, a_2 \ldots a_n)\big) dt$$

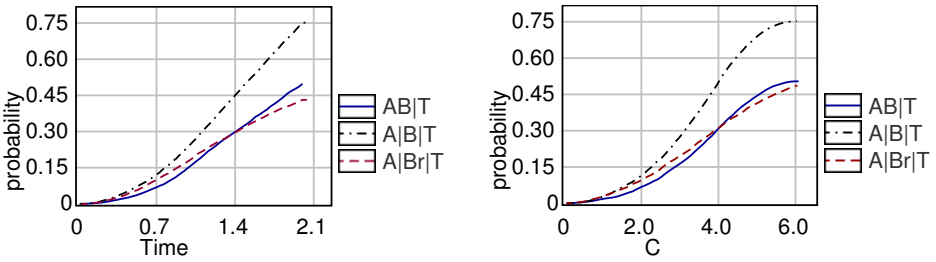
where $c = c(a_1)$, and as base case we take $P_{\boldsymbol{\mathcal{A}}}(\pi(\mathbf{s}), \varepsilon) = 1$.

This definition requires a few words of explanation: at the outermost level we integrate over all possible initial delays t. For a given delay t, the outputting component $c = c(a_1)$ will choose to make the broadcast at time t with the stated density. Independently, the other components will choose to a delay amount, which – in order for c to be the winner – must be larger than t; hence the product 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, in the global state resulting from all components having delayed t time-units and changed state according to the broadcasted action a_1 the probability of runs according to the remaining actions $a_2 \ldots a_n$ is taken into account.

Logical Properties Following [Pan10], the measure $\mathbb{P}_{\boldsymbol{\mathcal{A}}}$ may be extended in a standard and unique way to the σ-algebra generated by the sets of runs (so-called cylinders) $\pi(\mathbf{s}, a_1 a_2 \ldots a_n)$. As we shall see this will allow us to give proper semantics to a range of probabilistic time- and cost-constrained temporal properties. Let $\mathcal{A}$ be a NPTA. Then we consider the following non-nested PWCTL properties:

$$\psi \;::=\; \mathbb{P}\big(\Diamond_{C \leq c} \varphi\big) \sim p \;\mid\; \mathbb{P}\big(\Box_{C \leq c} \varphi\big) \sim p$$

[5] otherwise a specific weight distribution can be specified and used instead.



(a) (b)

Figure 7.2: Cumulative probabilities for time and cost-bounded reachability of T_3.

where C is an observer clock (of $\mathcal{A}$), φ a state-property (wrt. $\mathcal{A}$) , $\sim \in \{<, \leq, =, \geq, >\}$, and $p \in [0, 1]$. This logic is a stochastic extension of the classical WCTL logic for non-stochastic systems, where the existential quantifier is replaced by a probability operator. For the semantics let $\mathcal{A}^*$ be the modification of $\mathcal{A}$, where the guard $C \leq c$ has been conjoined to the invariant of all locations and an edge $(\ell, \varphi, o_\varphi, \emptyset, \ell)$ has been added to all locations ℓ, where o_φ is a new output action. Then:

$$\mathcal{A} \models \mathbb{P}\big(\Diamond_{C \leq c}\varphi\big) \sim p \;\text{ iff }\; \mathbb{P}_{\mathcal{A}^*}\Big(\bigcup_{\sigma \in \Sigma^*} \pi(s_0, \sigma o_\varphi)\Big) \sim p$$

which is well-defined since the σ-algebra on which $\mathbb{P}_{\mathcal{A}^*}$ is defined is closed under countable unions and finite intersections. To complete the semantics, we note that $\mathbb{P}(\Box_{C \leq c}\varphi) \sim p$ is equivalent to $(1 - p) \sim \mathbb{P}(\Diamond_{C \leq c}\neg\varphi)$. [6]

Compared with previous stochastic semantics of timed automata (see e.g., [BBB+07,BBBM08]), we emphasize the novelty of the semantics of NPTA in terms of RACES between components, truthfully reflecting their independencies. In particular our stochastic semantics of a network $(A_1|..|A_n)$ is significantly different from that obtained by applying the stochastic semantics of [BBB+07, BBBM08] to a product construction $A_1 A_2 \ldots A_n$, as information about independencies are lost. So though $(A_1|..|A_n)$ and $A_1 A_2 \ldots A_n$ are timed bisimilar they are in general not probabistic timed bisimilar, and hence distinguishable by PWCTL. The situation is illustrated with the following example.

Example 7.3. Reconsider the Example of Fig. 7.1. Then it can be shown that $(A|B|T) \models \mathbb{P}\big(\Diamond_{t \leq 2} T_3\big) = 0.75$ and $(A|B|T) \models \mathbb{P}\big(\Diamond_{C \leq 6} T_3\big) = 0.75$, whereas $(AB|T) \models \mathbb{P}\big(\Diamond_{t \leq 2} T_3\big) = 0.50$ and $(AB|T) \models \mathbb{P}\big(\Diamond_{C \leq 6} T_3\big) = 0.50$. Fig. 7.2 gives a time- and cost-bounded reachability probabilities for $(A|B|T)$ and $(AB|T)$ for a range of bounds. Thus, though the two NPTAs satisfy the same WCTL properties, they are obviously quite different with respect to PWCTL. The NPTA B_r of Fig. 7.1 is a variant of B, with the uniform delay distribution enforced by the invariant $y \leq 2$ being replaced by an exponential distribution with rate $\frac{1}{2}$. Here $(A|B_r|T)$ satisfies $\mathbb{P}\big(\Diamond_{t \leq 2} T_3\big) \approx 0.41$ and $\mathbb{P}\big(\Diamond_{C \leq 6} T_3\big) \approx 0.49$.

[6] We also note that the above (stochastic) interpretation of PWCTL is a conservative extension of the classical (non-stochastic) interpretation of WCTL, in the sense that $\mathcal{A} \models \mathbb{P}\big(\Diamond_{C \leq c}\varphi\big) > 0$ implies $\mathcal{A}_n \models \mathsf{E}\Diamond_{C \leq c}\varphi$, where $\mathcal{A}_n$ refers to the standard non-stochastic semantics of $\mathcal{A}$.

7.4 Statistical Model Checking for NPTA

As we pointed out, most of model checking problems for NPTAs and PWCTL (including reachability) are undecidable. Our solution is to use a technique that approximates the answer. We rely on SMC. At the heart of any SMC approach, there is an algorithm used to generate runs of the system following a stochastic semantics. We propose such an algorithm for NPTAs corresponding to the stochastic semantics proposed in Section 7.3. Then, we recap existing statistic algorithms, providing the basis for a first SMC algorithm for NPTAs.

Generating Runs of NPTA SMC is used for properties that can be monitored on finite runs. Here, we propose an algorithm that given an NPTA generates a random run up to a cost bound c (with time bounds being a simple case) of an observer clock C. A run of a NPTA is a sequence of alternations of states $\boldsymbol{s_0} \xrightarrow{d_0} \boldsymbol{s'_0} \xrightarrow{o_0} \boldsymbol{s_1} \xrightarrow{d_1} \dots \boldsymbol{s}_n$ obtained by performing delays d_i and emitting outputs o_i. Here we consider a network of NPTAs with states being of the form $(\boldsymbol{\ell}, \nu)$. We construct random runs according to Algorithm 11. We start from an initial state $(\boldsymbol{\ell_0}, \nu_0)$ and repeatedly concatenate random successor states until we reach the bound c for the given observer clock C. Recall that $\nu(C)$ is the value of C in state $(\boldsymbol{\ell}, \nu)$, and the rate of C in location $\boldsymbol{\ell}$ is $R(C)(\boldsymbol{\ell})$. We use the notation $\oplus$ to concatenate runs and $tail(run)$ to access the last state of a run and $delay(\mu_s)$ returns a random delay according to the delay density function μ_s as described in Section 7.3. The statement "pick" means choose uniformly among the possible choices. Lines 5-6 stop the delay when the runs reach their time bounds with the values of the clocks depending on their rates. The Algorithm 11 may be seen to be correct with respect to the stochastic semantics of NPTAs given in Section 7.3 in the sense that the probability of the (random) run $RR_{\boldsymbol{\mathcal{A}}}\big((\boldsymbol{\ell_0}, \nu_0), C, c\big)$ satisfying $\Diamond_{C \leq c}\varphi$ is $\mathbb{P}_{\boldsymbol{\mathcal{A}}}\Big(\Diamond_{C \leq c}\, \varphi\Big)$.

7.5 Statistical Model-Checking for comparisons

Here, we want to compare $p_1 = \mathbb{P}_{\boldsymbol{\mathcal{A}}}(\Diamond_{C_1 \leq c_1}\varphi_1)$ and $p_2 = \mathbb{P}_{\boldsymbol{\mathcal{A}}}(\Diamond_{C_2 \leq c_2}\varphi_2)$ without computing them. This comparison has clear practical applications e.g. it can be used to compare the performances of an original program with one of its newly designed extensions. This comparison cannot be performed with the algorithm presented in the previous section. Moreover, using Monte Carlo to estimate the probabilities (which is costly) would not help as both such probabilities would be estimated with different confidences that could hardly be related[7]. Wald has shown that this problem can be reduced to a sequential hypothesis testing one. Our contributions here are (1) to apply this algorithm in the formal verification area, (2) to extend the original algorithm to handle cases where we observe the same outcomes for both experiments, and (3) to implement a parametric extension of the algorithm

[7]Interleaving intervals for the estimate (even with same confidence) may give non-deterministic results, not to mention that computing estimates is more expensive than hypothesis testing in terms of runs.

Algorithm 11: Random run for a NPTA-network $\mathcal{A}$

```
1  function RR_A((ℓ_0, ν_0), C, c)
3  run := (ℓ, ν) := tail(run) := (ℓ_0, ν_0)
5  while ν(C) < c do
7      for i = 1 to |ℓ| do d_i := delay(μ_(ℓ_i,ν_i))
9      d := min_{1≤i≤|ℓ|}(d_i)
11     if d = +∞ ∨ ν(C) + d * R(ℓ)(C) ≥ c then
13         d := (ν(C) − c)/R(ℓ)(C)
15         return run⊕ --d--> (ℓ, ν + d * R(ℓ))
16     end
18     else
20         pick k such that d_k = d; ν_d := ν + d * R(ℓ)
22         pick ℓ_k --g,o,r--> ℓ'_k with g(ν_d)
24         run := run⊕ --d--> (ℓ, ν_d) --g,o,r--> (ℓ[l'_k/l_k], [r ↦ 0](ν_d))
25     end
27     (ℓ, ν) := tail(run)
28 end
29 return run
```

that allows to reuse results on several timed bounds. More precisely, instead of comparing two probabilities with one common cost bound $C \leq c$, the new extension does it for all the N bounds $i * c/N$ with $i = 1 \dots N$ by reusing existing runs.

Comparison Algorithm. Let the efficiency of satisfying $\Diamond_{C_1 \leq c_1} \varphi_1$ over runs be given by $k_1 = p_1/(1-p_1)$ and similarly for $\Diamond_{C_2 \leq c_2} \varphi_2$. The relative superiority of "φ_2 over φ_1" is measured by the ratio $u = \frac{k_2}{k_1} = \frac{p_2(1-p_1)}{p_1(1-p_2)}$. If $u = 1$ both properties are equally good, if $u > 1$, φ_2 is better, otherwise φ_1 is better. Due to indifference region, we have two parameters u_0 and u_1 such that $u_0 < u_1$ to make the decision. If $u \leq u_0$ we favor φ_1 and if $u \geq u_1$ we favor φ_2. The parameter α is the probability of rejecting φ_1 when $u \leq u_0$ and the parameter β is the probability of rejecting φ_2 when $u \geq u_1$. An outcome for the comparison algorithm is a pair $(x_1, x_2) = (r_1 \models \Diamond_{C_1 \leq c_1} \varphi_1, r_2 \models \Diamond_{C_2 \leq c_2} \varphi_2)$ for two independent runs r_1 and r_2. In Wald's version (lines 10–14 of Algorithm 12), the outcomes $(0,0)$ and $(1,1)$ are ignored. The algorithm works if it is guaranteed to eventually generate different outcomes. We extend the algorithm with a qualitative test (lines 5–9 of Algorithm 12) to handle the case when the outcomes are always the same. The hypothesis we test is $\mathbb{P}_{\mathcal{A}}((r_1 \models \Diamond_{C_1 \leq c_1} \varphi_1) = (r_2 \models \Diamond_{C_2 \leq c_2} \varphi_2)) \geq \theta$ for two independent runs r_1 and r_2. We note that this does not affect the correctness of the original algorithm for accepting or rejecting process 2. The modified algorithm now returns *indifferent* in addition, which corresponds to our added hypothesis to cut down the number of necessary runs[8]. Typically we want the parameters $p'_0 = \theta + \delta_0$ (for the corresponding

[8]This also frees us from the assumption that the processes have some different outputs.

hypothesis H_0) and $p'_1 = \theta - \delta_1$ (for H_1) to be close to 1. Our version of the comparison algorithm is shown in algorithm 12 with the following initializations:

$$a = \frac{\log(\frac{\beta}{1-\alpha})}{\log(u_1)-\log(u_0)}, r = \frac{\log(\frac{1-\beta}{\alpha})}{\log(u_1)-\log(u_o)}, c = \frac{\log(\frac{1+u_1}{1+u_0})}{\log(u_1)-\log(u_o)}$$

Algorithm 12: Comparison of probabilities

```
1  function comprise(S:model , ψ1, ψ2: properties)
3  check := 1, q := 0, t := 0
5  while true do
7      Observe the random variable x1 corresponding to ψ1 for a run.
9      Observe the random variable x2 corresponding to ψ2 for a run.
11     if check = 1 then
13         x := (x1 == x2)
15         q := q + x * log(p'1/p'0) + (1 − x) * log((1 − p'1)/(1 − p'0))
17         if q ≤ log(β/(1 − α)) then return indifferent
19         if r ≥ log((1 − β)/α) then check := 0
20     end
22     if x1 ≠ x2 then
24         a := a + c, r := r + c
26         if x1 = 0 and x2 = 1 then t := t + 1
28         if t ≤ a then accept process 2.
30         if t ≥ r then reject process 2.
31     end
32 end
```

Parametrised Comparisons We now generalise the comparison algorithm to give answers not only for one cost bound c but N cost bounds $i * c/N$ (with $i = 1 \ldots N$). This algorithm is of particular interest to generate distribution over timed bounds value of the property. The idea is to reuse the runs of smaller bounds. When $\Diamond_{C\leq c}\varphi_1$ or $\Diamond_{C\leq c}\varphi_2$ holds on some run we keep track of the corresponding point in cost (otherwise the cost value is irrelevant). Every pair or runs gives a pair of outcomes (x_1, x_2) at cost points (c_1, c_2). For every $i = 1 \ldots N$ we define the new pair of outcomes $(y_{i_1}, y_{i_2}) = \big(x_1 \wedge (i \cdot c/N \geq t_1 \cdot rate_C), x_2 \wedge (i \cdot c/N \geq t_2 \cdot rate_C)\big)$ for which we use our comparison algorithm. We terminate the algorithm when a result for every i^{th} bound is known.

7.6 Case Studies

We have extended UPPAAL with the algorithms described in this paper. The implementation provides access to all the powerful features of the tool, including user defined functions and types, and use of expressions in guards, invariants, clock-rates as well as delay-rates. Also the implementation supports branching edges with discrete probabilities (using weights), thus supporting probabilistic timed automata (a feature for which our stochastic semantics of NPTA may be easily extended).

Besides these additional features, the case-studies reported below (as well as the plots in the previous part of the paper) illustrate the nice features of the new plot composing GUI of the tool[9]. Our objective here is not to study the evolutions of performances with the increase of condidence level, but rather to give a sample of case studies on which our approach can be applied.

Train-Gate Example We consider a train-gate example, where N trains want to cross a one-track bridge. We extend the original model by specifying an arrival rate for Train i ($(i+1)/N$). Trains are then approaching, but they can be stopped before some time threshold. When a train is stopped, it can start again. Eventually trains cross the bridge and go back to their safe state. The template of these trains is given in Fig. 7.3(a). Our model captures the natural behavior of arrivals with some exponential rate and random delays chosen with uniform distributions in states labelled with invariants. The tool is used to estimate the probability that Train 0 and Train 5 will cross the bridge in less than 100 units of time. Given a confidence level of 0.05 the confidence intervals returned are $[0.541, 0.641]$ and $[0.944, 1]$. The tool computes for each time bound T the frequency count of runs of length T for which the property holds. Fig. 7.3(b) shows a superposition of both distributions obtained directly with our tool that provides a plot composer for this purpose.

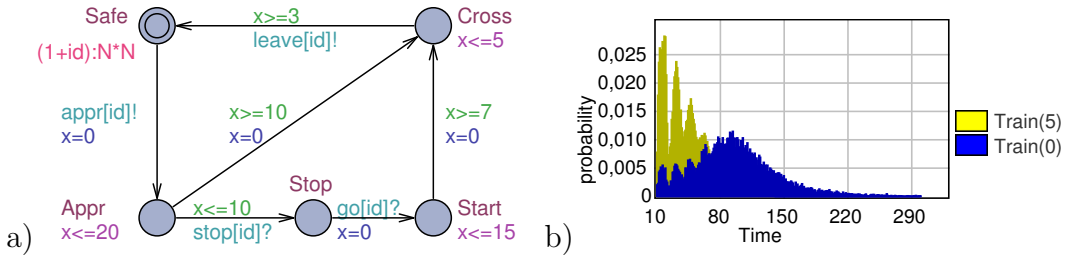


Figure 7.3: Template of a train (a) and probability density distributions for $\Diamond_{T \leq t}$`Train(0).Cross` and $\Diamond_{T \leq t}$`Train(5).Cross`.

The distribution for Train 5 is the one with higher probability at the beginning, which confirms that this train is indeed the faster one. An interesting point is to note the valleys in the probability densities that correspond to other trains conflicting for crossing the bridge. They are particularly visible for Train 0. The number of valleys corresponds to the number of trains. This is clearly not a trivial distribution (not even uni-modal) that we could not have guessed manually even from such a simple model. In addition, we use the qualitative check to cheaply refine the bounds to $[0.541, 0.59]$ and $[0.97, 1]$.

We then compare the probability for Train 0 to cross when all other trains are stopped with the same probability for Train 5. In the first plot (Fig. 7.4 top), we check the same property with 100 different time bounds from 10 to 1000 in steps of 10 and we plot the number of runs for each check. These experiments only check for the specified bound, they are not parametrised. In the second plot, we use the

[9] *http://www.cs.aau.dk/~adavid/smc/* for details.

parametric extension presented in Section 7.5 with a granularity of 10 time units. We configured the thresholds u_0 and u_1 to differentiate the comparisons at $u_0 = 1-\epsilon$ and $u_1 = 1+\epsilon$ with $\epsilon = 0.1, 0.05, 0.01$ as shown on the figure. In addition, we use a larger time bound to visualise the behaviors after time 600 that are interesting for our checker. In the first plot of Fig. 7.4, we show for each time bound the average of runs needed by the comparison algorithm repeated 30 times for different values of ϵ. In the bottom plot, we first superpose the cumulative probability for both trains (curves Train 0 and Train 5) that we obtain by applying the quantitative algorithm of Section 7.4 for each time bound in the sampling. Interestingly, before that point, train 5 is better and later train 0 is better. Second, we compare these probabilities by using the comparison algorithm (curves 0.1 0.05 0.01). This algorithm can retrieve 3 values: 0 if Train 0 wins, 1 if Train 5 wins and 0.5 otherwise. We report for each time bound and each value of ϵ the average of these values for 30 executions of the algorithm.

In addition, to evaluate the efficiency of computing all results at once to obtain these curves, we measure the accumulated time to check all the 100 properties for the first plot (sequential check), which takes $92s, 182s, 924s$ for $\epsilon = 0.1, 0.05, 0.01$, and the time to obtain all the results at once (parallel check), which takes $5s, 12s, 92s$. The experiments are done on a Pentium D at 2.4GHz and consume very little memory. The parallel check is about 10 times faster[10]. In fact it is limited by the highest number of runs required as shown by the second peak in Fig. 7.4. The expensive part is to generate the runs so reusing them is important. Note that at the beginning and at the end, our algorithm aborts the comparison of the curves, which is visible as the number of runs is sharply cut.

Lightweight Media Access Control Protocol (LMAC). This protocol is used in sensor networks to schedule communication between nodes. It is targeted for distributed self-configuration, collision avoidance and energy efficiency. In this study we reproduce the improved UPPAAL model from [FvHM07] without verification optimisations, parametrise with network topology (ring and chain), add probabilistic weights (exponential and uniform) over discrete delay decisions and examine statistical properties which were not possible to check before. Based on [FvH07], our node model consumes 21, 22, 2 and 1 power units when a node is sending, receiving, listening for messages or being idle respectively.

Fig. 7.5a shows that collisions may happen in all cases and the probability of collision is higher with exponential decision weights than uniform decision weights, but seems independent of topology (ring or chain). The probability of collision stays stable after 50 time units, despite longer simulations, meaning that the network may stay collision free if the first collisions are avoided. We also applied the method for parametrised probability comparison for the collision probability. The results show that up to 14 time units the probabilities are the same and later exponential weights have higher collision probability than uniform, but the results were inconclusive when comparing different topologies.

[10]The implementation checks simulations sequentially using a single thread.

r

Table 7.1: Performance of SMC (sec)

Param.			Estim.				Hyp. Testing			
n	k	m	PRISM	Up_p	Up_d	Up_c	PRISM	Up_p	Up_d	Up_c
4	4	3	2.7	0.3	0.2	0.2	2.0	0.1	0.1	0.1
6	6	3	7.7	0.6	0.5	0.4	3.9	0.2	0.2	0.3
8	8	3	26.5	1.2	0.9	0.7	16.4	0.5	0.4	0.3
20	40	20	>300				>300	35.5	26.2	20.7
30	40	20	>300				>300	61.2	41.8	33.2
40	40	20	>300				>300	92.2	56.9	59.5
40	20	20	>300				>300	41.1	31.2	26.5
40	30	20	>300				>300	68.8	46.7	46.1
40	55	40	>300				>300			219.5

The probable collision counts in the chain topology are shown in Fig. 7.5b, where the case with 0 collisions has a probability of 87.06% and 89.21% when using exponential and uniform weights respectively. The maximum number of probable collisions is 7 for both weight distributions despite very long runs, meaning that the network eventually recovers from collisions.

Fig. 7.6 shows energy consumption probability density: using uniform and exponential weights in a chain and a ring topologies. The probability `Pr[energy <= 50000](<> time>=1000)` as estimated. Ring topology uses more power (possibly due to collisions), and uniform weights use slightly less energy than exponential weights in these particular topologies.

Duration Probabilistic Automata (DPA) [KSB]. Those automata are used for modelling job-shop problems. A DPA consists of several Simple DPAs (SDPA). An SDPA is a processing unit, a clock and a list of tasks to process sequentially. Each task has an associated duration interval, from which its duration is chosen (uniformly). Resources are used to model task races – we allow different resource types and different quantities of each type. A fixed priority scheduler is used to resolve conflicts. An example is shown in Fig. 7.7. DPA can be encoded in our tool (continuous or discrete time semantics) or in PRISM (discrete semantics), see the technical report [PvV11]. In PRISM, integer and boolean variables are used to encode the current tasks and resources. PRISM only supports the discrete time model. In UPPAAL, a chain of waiting and task locations is created for each SDPA. Guards and invariants encode the duration of the task, and an array of integers contain the available resources. The scheduler is encoded as a separate template.

For UPPAAL, we have modelled a discrete version as close as possible to the PRISM model (Up_p), an improved discrete version that "jumps" to interesting points in time (Up_d), and a continuous time version that making full use of our formalism (Up_c).

The performance of the translations is shown in Tab. 7.1, based on DPAs with n SDPAs, k tasks per SDPA and m resource types. The resource usage and duration interval are randomised. In the hypothesis testing column, UPPAAL uses the sequential hypothesis testing introduced in Section 7.4, whereas PRISM uses its own new implementation of the hypothesis testing algorithm. In the estimation column,

both UPPAAL and PRISM use the quantitative check of Section 7.4, but UPPAAL is faster thanks to its more suitable formalism. For both tools, the error bounds used are $\alpha = \beta = 0.05$. In the hypothesis test, the indifference region size is 0.01, while we have $\epsilon = 0.05$ for the quantitative approach. The query for the approximation test is: "What is the probability of all SDPAs ending within t time units?", and for hypothesis testing it is: "Do all SDPAs end within t time units with probability greater than 40%?". The value of t varies for each model as it was computed by simulating the system 369 times and represent the value for which at least 60% of the runs reached the final state. Each number in the table is the average of 10 SMC analyses on the given model. The results show that UPPAAL is an order of magnitude faster than PRISM even with the discrete encoding, which puts UPPAAL at a disadvantage given that it is designed for continuous time[11].

7.7 Conclusion and Extensions of the results

The contribution of this chapter have been obtained together with researchers from Aalborg University. The work, which was implemented in UPPAAL has been continued in various directions. One of those directions has been to extend to the formalism to non-linear dynamics [BDL+12b]. Another contribution was in considering dynamic logics [DLLP13] and efficient monitoring techniques [BDL+12a] for timed extension of LTL. Our main objective is to export the formalism developed in this chapter within the Plasma Lab framework. Our second objective is to extend the results from the other chapters to the new timed stochastic semantic.

[11] We note that the number of steps generated for the runs of the PRISM model and the discrete UPPAAL model upp_p are comparable.

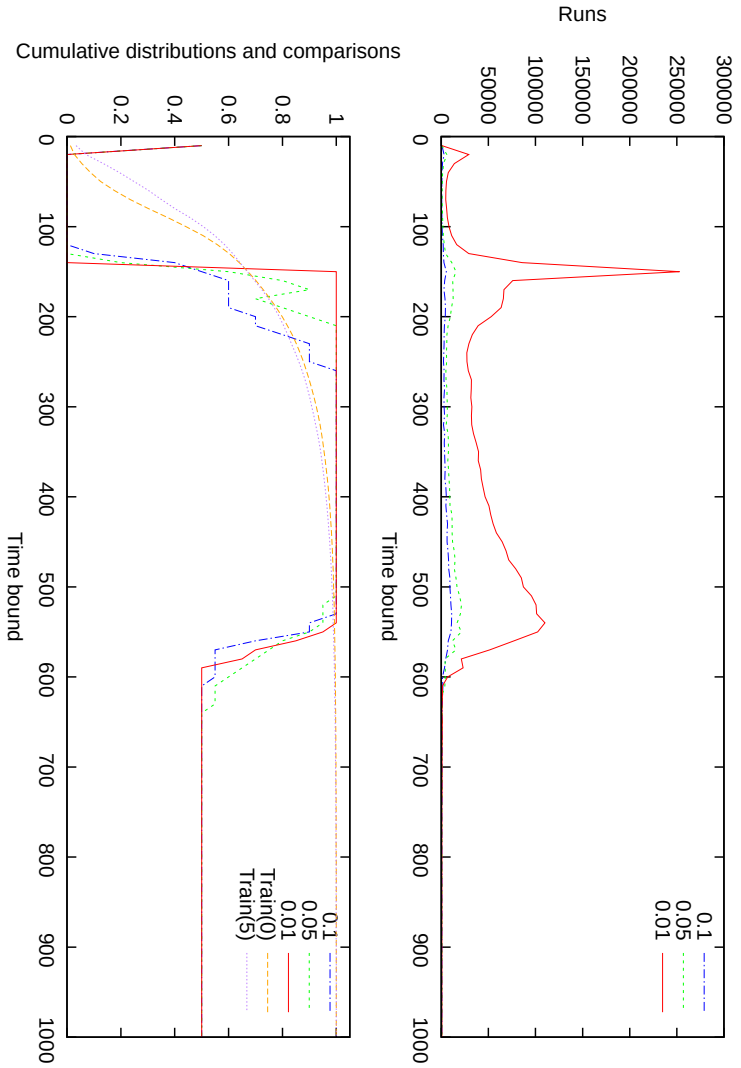
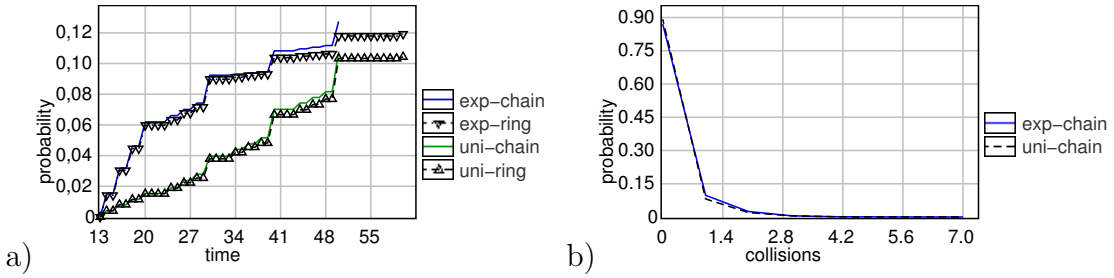
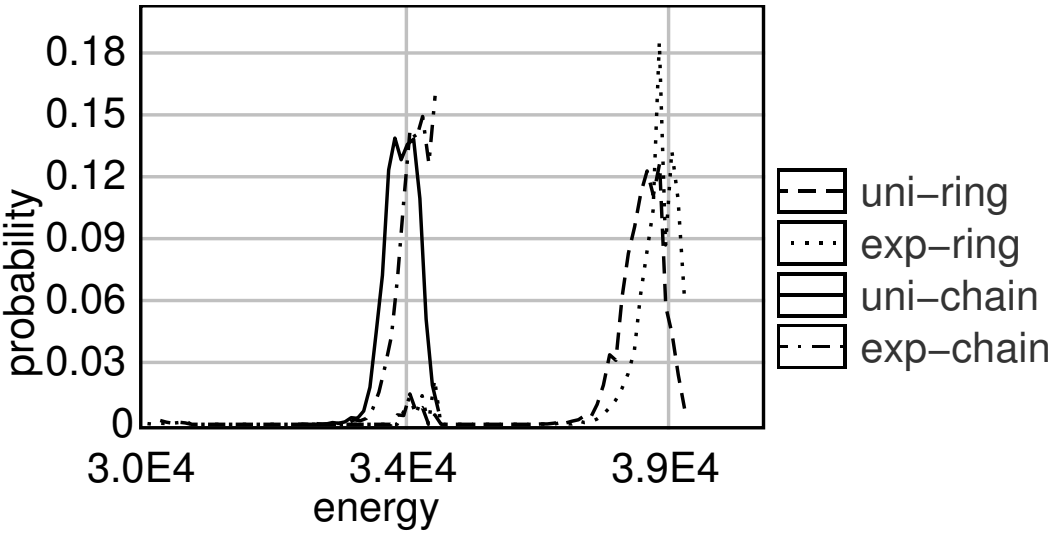


Figure 7.4: Comparing trains 0 and 5.



a) b)

Figure 7.5: Collision probabilities when using exponential and uniform weights in chain and ring topologies, a) cumulative probability of collision over time and b) probability of having various numbers of collisions.



r

Figure 7.6: Total energy consumption.

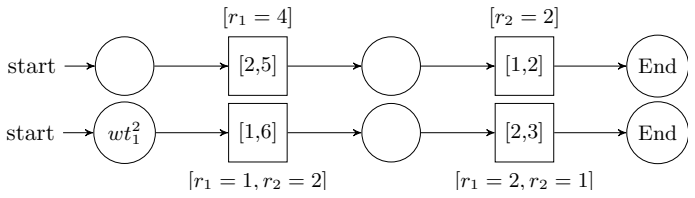


Figure 7.7: Rectangles are busy states and circles are for waiting when resources are not available. There are $r_1 = 5$ and $r_2 = 3$ resources available.

Chapter 8

Conclusion and Perspectives

In this section, we present some directions for future work in SMC; they complete those proposed at the end of each chapter. We then briefly describe our research project for the next five years. There we put the work we have done on SMC in perspective with work we have done on interface theories, contracts, and product lines.

8.1 Perspectives for Statistical Model Checking

1In this thesis, we have presented several contributions to Statistical Model Checking. In this section, we will briefly survey directions for future research.

Chapter 2 presents a technique to handle rare events for stochastic systems. It would be of interest to extend this technique to a larger class of systems. This includes, e.g., the real-time stochastic model introduced in Chapter 7. In [ZBC12], the authors already considered rare events for real-time systems. However, they have been adopting an importance sampling approach and they do not exploit the structure of the system in the definition of their Cross-entropy. One of the main challenge in extending our work will be to adapt the score function to the real-time setting.

The work on non-deterministic systems presented in Chapter 5 can also be extended to real-timed systems, in the spirit of [FHH+11]. It would also be of interest to combine extend the work of Chapter 2 to non-deterministic systems. Crucial here is to define a notion of rare event that depends on probabilistic and/or non-deterministic informations. Another objective will be to extend the approach to a richer class of systems such as those with rewards.

Our quest to minimize the number of simulations also passes by the introduction of new SMC algorithms. In [ZPC10, JCL+09], the authors considered Bayesian Statistical Model Checking that minimize the number of simulations required to converge by exploiting a prior knowledge on the probability distribution. One of our objective is to combine this work with those of Chapters 2, 3, 7, and 5. Preliminary work exists in [GHJ+14], but for importance sampling only.

As outlined in the thesis, techniques to verify unbounded properties and nested

properties are still not efficient. We plan to conduct a study to improve existing algorithm. The latter shall be done with the objective of extending SMC to the full PCTL* spectrum [BK08].

The efficiency of SMC largely depends on the number of executions that need to be generated and monitored. Studying new efficient monitoring techniques that can minimize the time needed to produce a simulation are of crucial importance.

Composite systems present a major challenge for formal verification. In [dAdSF+05], we have proposed a series of interface theories that allows us to reduce the complexity of the design by exploiting abstraction and incremental-design mechanism. In [BBB+12], we have experimentally showed that this approach can be used to compute a stochastic abstraction of a large-size heterogeneous system on which SMC can efficiently be applied. Our objective would be to generalize this experiment into a full theory.

Aside from those technical works, our main objective for the near future is to pursue the development of Plasma Lab. Particularly, we would like to integrate the results obtained in Chapter 7 directly into the tool. We shall also continue the integration of Plasma Lab with other professional toolsets such as DESYRE or MODELICA. One of our main objectives will also be to exploit new hardware technologies such as beagleboard (pursuing the work from the DALI project) and GPGPU to increase the efficiency of the tool. Finally, we will also consider new application domains for the tool. This includes, among others, systems biology [JCL+09].

To conclude, we briefly mention other works of interest. This includes progress measure, or the verification of models that encode both hardware and software constraints. Another perspective is to apply SMC to the verification of Software product lines. Those are a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way. One of the major challenges with product lines is to exploit commonalities to avoid redoing model checking for each products. In [CCH+14, CHL+14, CCS+13, ACL+15], we have proposed a series of work that go into that direction. It would be of interest to combine those works with SMC. Finally, it is undoubted that SMC has a spot in emerging research topics such as privacy quantification [BLMW13].

8.2 Systems of Systems: a Vision

In this section we briefly present our vision for the rigorous design of Systems of Systems. This project takes the name of ESTASYS, which states for "Efficient STAtistical Methods in SYstems of Systems".

The advent of service-oriented and cloud architectures is leading to generations of computer systems that exhibit a new type of complexity: such systems are no longer statically configured, but comprise components that are systems in their own right, able to discover, select and bind on-the-fly to other components that can deliver services that they require. These complex systems, referred to as *Systems of*

Systems (SoS), can change over time as each component creates and modifies the network over which it needs to operate: as they execute, the components create a network of their own and use it to fulfil their goals.

The Internet, made up of an unsupervised and rapidly growing, dynamically configured set of computers and physical connections, is an obvious illustration of the potential complexity of dynamic networks of interactions. Another example is the so-called "Flash Crash" in the U.S. equity market: on May 6, 2010, a block sale of 4.1 billion dollars of futures contracts executed on behalf of a fund-management company triggered a complex pattern of interactions between the high-frequency algorithmic trading systems (algos) that buy and sell blocks of financial instruments and made the Dow Jones Industrial Average drop more than 600 points, representing the disappearance of 800 billion dollars of market value. This example is an illustration of the faulty divergence of SoS behaviour, where the system starts to misbehave and dynamically creates new components that follow the same pattern and make the problem worse. Examples of this include when a SoS detects high energy use and invokes a new component to reduce the energy, thus consuming *more* energy. Until now, such divergence has been mostly handled by humans that eventually observe the faulty behaviour and manually intervene to stop it. This human-based solution is not always successful and clearly unsatisfactory, since it acts retrospectively, when the system has already failed.

8.2.1 Our Grand Challenge

SoS are an efficient means of achieving high performance and are thus becoming ubiquitous. Society's increasing reliance on SoS demands that they are reliable, but tools to guarantee this at the design stage do not exist. Most conventional formal analysis techniques, even those dedicated to adaptive systems [Che09], fail when applied to SoS because they are designed to reason on systems whose state space can be predicted in advance. The grand challenge addressed by ESTASYS is the fundamental overhaul of formal methods techniques in the design of SoS life cycle.

Our objective is to propose a revolutionary new formal methodology to support an evolutionary adaptive and iterative SoS life-cycle. *We foresee the following breakthroughs:*

1. In existing verification approach, the high level language used to describe a system and its architecture have often been abstracted. This approach will not work with SoS. Indeed, there are complex aspects of SoS such as dynamicity (e.g., components with independent decision joining or leaving the SoS) that must be handled at the architectural level. In addition, reasoning on the behavioral part of the system also impact (is reused/interleave with) the evolution of the architecture (detection of emergent behaviors, privacy by change of architecture and communication, etc). Consequently, the analysis of the behavioral part of the SoS should be reused to understand the evolution of its architecture. This means that the approach that will be proposed in the rest of this section has to be put in perspective with a global vision of how to

design and understand SoS at an engineering level.

2. One of our main focuses will be to develop a theory that addresses the Behavioral view point of the SoS. Two of the major difficulties will be 1. to address the complex heterogeneous nature of SoS, and 2. the complexity of the system. To cope with the complexity, we will propose to develop a new formal model that relies on abstracting the behaviors of components via an extension of the interface theory framework we proposed in [dAdSF+05]. Interface act as an abstraction for the internal behaviour of each component, and they only reveal their public informations. To cope with the heterogeneous aspects, we plan to extend and generalize the work done on FMI/FMU [FMU]. This theory proposes a unified framework for heterogeneous timed components, but yet it does not take into account other quantitative aspects such as probabilities, energy, etc. Another difficulty here will be to generate those interfaces directly from the behavioral views of the architecture defined above.

3. We will exploit our new interface model in order to make predication on the evolution of the system, or to detect emergeant behaviors. This analysis only makes sens if we have some knowledge of the environment where the system has been deployed. To obtain this knowledge, cutting edge algorithms coming from the area of statistics and learning will be exploited to make predictions about autonomous systems making local decisions. Specifically, **statistical abstractions** of the observed runtime behaviour of components will be used to quantify, e.g., the probability that a number of new components satisfying some constraints will be started at a given execution point. Runtime verification will monitor the executions of the deployed system to create distributions embedded in the interfaces framework proposed above. When a deployed system is available, ESTASYS will interleave simulation, analysis and runtime monitoring, using real behaviour to update the statistical abstractions, and eventually replace some of those abstractions by concrete SoS decicated interfaceq models. Our methodology will adopt a Bayesian approach: (i) an initial, plausible distribution is 'guessed', based on whatever is known; (ii) the system is simulated using the current approximated distribution; (iii) the behaviour of the simulated system becomes the new approximation; (iv) the process is iterated as necessary. While learning-based simulation approaches, such as model fitting, can be used to learn the abstraction by conducting simulations from a finite set of initial components, we will have to provide clear evidence that a global property holds on the system if it holds on its corresponding statistical abstraction.

4. Another objective will be to develop new statistical algorithms for SMC that scale efficiently and handle undecidability will impact the formal analysis of complex systems. Those algorithms will largely extend those presented in this thesis and respond to questions such as "can we predict an emergeant behavior", "what are the parameters to fullfill this property whather the environment does", etc. In addition, those algorithms should exploit the composite nature

of SoS, and handle new problematic such as computer security (which is crucial for a robust interconnected framework) and privacy. For the latter issue, we plan to extend our work from [BLMW13, BLN+14] from an numerical to a simulation approach. Another challenge is the one of parameter optimization in order to optimize the architectural design of the SoS

5. Our results will be implemented as an extension of Plasma Lab that will be constructed in close collaboration with our industrial partners. This will ensure relevance to industry and potentially high impact in the marketplace. Observe that the architecture of Plasma Lab has been made in such a way that adding those new plugins should be easy.

8.2.2 Potential Impact

Our vision will lead to the creation of a top class research team at INRIA as well as to an interdisciplinary community of researchers and practitioners at the world level. .

ESTASYS will set the foundations for an engineering domain dedicated to SoS that will benefit the European software industry. This will be achieved by creating mathematical models that capture the computational power, autonomous decisions and complex stochastic and real-time dynamics of SoS. ESTASYS will produce new decidability and complexity results, simulation-based techniques, and algorithms with correctness arguments. All aim at efficient reasoning about SoS and will be traced back to case studies.

Chapter 9

Appendix

9.1 Curriculum Vitae

9.1.1 Brief presentation of Axel Legay

Name: Axel Legay
Date of Birth: 23/03/1980
Marital status: Single
Email address: axel.legay@inria.fr
Personal web page: `http://people.irisa.fr/Axel.Legay/`

Professional Experience

- 2014 – : Member of Inria's evaluation committee.
- 2013 – 2015: Part time associate Professor, Royal Holloway University of London.
- 2011 – 2013: Part time associate Professor, Aalborg University, Denmark.
- 2009 – Present: Full-time researcher, INRIA Rennes, France.
- 2008 – 2009: Post-doctoral researcher at Carnegie Mellon University, USA.
- Member of model checking group of Prof Ed. Clarke.
- Fellow of the Belgian American Educational Foundation.

Education

- 2003 – 2007: PhD in Computer Science, *University of Liege*, Belgium.
- Thesis: *Generic Techniques for the Verification of Infinite-State Systems*
- Advisor: Pierre Wolper.

- Defense date: 7th of December 2007.
- 2002 – 2003: Researcher, *University of Liege*, Belgium.
- 1998 – 2002: Master in Computer Science, *University of Liege*, Belgium.

Research Area

Since 2002, I have developed a research activity in the field of formal verification. I have been active in the area of symbolic verification of infinite-state systems (PhD thesis) and component-based design (started during my PhD and continued as a main topic). In 2008, I started to work in the area of quantitative model checking. I proposed new simulation-based approaches known under the names of statistical model checking and stochastic abstractions. I develop a balanced activity between theoretical contributions, empirical analysis and validation and applied research in collaboration with industrial partners.

Since 2013, I am coordinating an exploratory action on the topic of Systems of Ssystems at Inria Rennes. This team is constituted of eleven researchers.

International Recognition (A Selection)

- Since 2009, I have been leading a collaboration with Aalborg University in Denmark. This collaboration has supported several montly stays in both directions, as well as several long-term visits by students. We have co-published more than 20 conference papers and 5 journal papers.
- Since 2010, I have been collaborating with the team of Prof. Heymans. I have been visiting him regularly and we have published five papers, among which three have been presented at the international Conference on Software Engineering.
- I have a longstanding collaboration with Profs. Bensalem and Sifakis. I have visited them regularly at Verimag Grenoblem and Lausanne.
- During my PhD thesis, I worked three months as a visiting researcher at Oxford University and during six months as a visiting researcher at University of California. I am still punctually collaborating with researchers from those two institutes.

I have several other collaborations with Moshe Vardi, Edmund Clarke, Joost-Pieter Katoen, and my colleagues from the European and national projects.

Funding ID as principal INRIA investigator

I am (or have been) the Inria PI for the following projects.

- EU Project ACANTO (2015 – 2018). This project aims at designing an autonomous robot working in a distributed environment.
- ANR *ALgebraic Methods for Real-Time and HYbrid Model Checking* (Malthy) (2014 – 2018): The objective of this project is to study new models and techniques to reason on quantitative systems.
- PLASMA (2012 – 2014): Inria research action fot the implementation of PLASMA; 100k.
- EU Project *Self Energy-Supporting Autonomous Computation* (SENSATION) (2012 – 2015). The focus of the project is in developing new formal techniques to reason on energy problems. 300k for INRIA.
- EU Project *Devices for assisted living* (DALI) (2011 – 2014). The focus of the project is in proposing new devices for assisted living.
- EU Project *Designing for Adaptability and evolutioN in System of systems Engineering* (DANSE) (2011 – 2014). The focus of the danse project is in studying SoS, which is also the main focus of my application.
- CREATE Project ESTASE (2011 – 2013). This project, which is under the creative thematic of the Britany region, focuses on stochastic abstractions for a fixed number of applications.

I also participate(d) to the following projects:

- EU Project *Embedded Multi-Core systems for Mixed Criticality applications in dynamic and changeable real-time environments* (EMC2) (2014 – 2016). This project aims at deploying new tools to verifying Cyber Physical Systems.
- EU Project *Component-based Embedded Systems design* (COMBEST) (2009 – 2011). The focus of COMBEST was the development of new component-based design approach for software design. 360 k for INRIA.
- Network of Excellence ARTIST Design (2002 – 2012). The focus of ARTIST was in the study of emergent approaches for the rigorous design of embedded applications.

9.1.2 Supervision

I have been supervising around ten postdocs and five experimented engineers on the topics of statistical model checking, interface theories, and information quantifying flow.

I offically co-supervise(d) 2 PhD students: Cyrille Jegourel (100 percents) and Mounir Chadli (50 percents). Due to my collaborations, I also co-supervised Benoît Delahaye (50 pecents), Maxime Corddy (university of Namur, 50 percents), Andreas Classens (University of Namur, 50 percents), Mikkel Peddersen (Aalborg University, 50 percents), and Dany poulsen (Aalborg University, 50 percents)

9.1.3 Services to Community

I have been invited in the following events.

1. 1st International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems.
2. 4th Artist Design Workshop on Foundations of Component-based Design, 2009.
3. 5th Artist Design Workshop on Foundations of Component-based Design, 2010.
4. 2nd Quantitative Model Checking School, 2012.
5. 4th International Workshop on Foundations of Interface Technologies.
6. Cyber phyiscal systems school Algeria 2013.
7. 10th International Symposium on Formal Aspects of Component Software.
8. Cyber physical systems School Grenoble 2014.
9. 8th International Symposium on Theoretical Aspects of Software Engineering.

I also acted as invited speaker for more than 30 seminars.

I have been chairing/organizing the following events.

1. 4th International Conference on Runtime Verification: PC Chair, General Chair.
2. 11th International Workshop on Verification of Infinite-State Systems.
3. 3rd International Workshop on Foundations of Interface Theories.
4. 1st International Rigorous Embedded Design workshop 2011.
5. 2nd Quantities in Formal Methods workshop 2012.
6. 4th Runtime Verification Conference 2013.
7. 12th International Conference on Formal Modeling and Analysis of Timed Systems.

8. 2nd Workshop MEALS for exchanges between Europe and Argentina.
9. Statistical Model Checking track chair at the 6th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation
10. 1st international workshop on Software Product Line Analysis Tools.

I also organized two Dagstuhl events and several project meetings.

I have been guest editor for the following journals:

- Guest editor for the International Journal on Software Tools for Technology Transfer on "Cyber Physical Systems".
- Guest editor for the Formal Methods in System Design journal on "runtime verification"
- Guest editor for the International Journal on Software Tools for Technology Transfer on "statistical model checking".

9.1.4 Program committees

I have been serving in more than 35 program committees among which TACAS, FASE, MEMOCODE, and TCS.

9.1.5 Publications

I have published more than 180 articles in top class journals and conferences. Here are some interesting statistics:

- 3 regular papers and 4 tool papers in *International Conference on Computer Aided Verification*.
- 3 regular papers and 1 tool papers in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*.
- 3 regular papers, 2 tool papers, and 1 tutorial in *International Conference on Quantitative Evaluation of SysTems*.
- 3 papers in *International Conference on Foundations of Software Technology and Theoretical Computer Science*.
- 4 regular papers and a tutorial in *International Conference on Software Engineering*.
- 4 papers in *Theoretical Computer Science*.
- 2 papers in *Information and Computation*.

- 1 paper in *Performances Evaluation*.
- 3 papers in *Formal Methods and System Design*.
- 1 paper in *Transaction and Software Engineering*.

9.2 Brief Description of other works by the author

In this section, we briefly outline three other research topics that we have pursued over the past six years.

9.2.1 Interface and Contract theories

Nowadays, systems are tremendously big and complex, resulting from the assembling of several components. These many components are in general designed by teams, working independently but with a common agreement on what the interface of each component should be. As a consequence, mathematical foundations that allow to reason at the abstract level of interfaces are needed. Any good interface theory should propose a satisfaction relation (to decide whether a system is an implementation of an interface), a consistency check (to decide whether the specification admits an implementation), a refinement (to compare specifications in terms of inclusion of sets of implementations), logical composition (to compute the intersection of sets of implementations), and structural composition (to combine interfaces).

Most of existing interface theories [daH05, Nym08] do not allow to specify timed (scheduling, ...) and/or stochastic (failures, ...) constraints. However, handling at least one of these aspects is often needed to model complex systems such as embedded and heterogeneous systems. In [DLL+10a], we have proposed the first complete interface theory for timed systems. Timed I/O interfaces are timed automata whose transitions are equipped with Input (environment) and Output (system) modalities. We defined satisfaction, refinement, composition, and conjunction. We also proposed an optimistic game-based approach to decide whether a specification admits at least one implementation. The theory comes together with an algorithm to synthesize an interface automaton from two specifications. Our approach has been implemented in the tool set Ecdar [DLL+10b], that is an extension of the well-known Uppaal tool set. Our Ecdar tool set has been able to handle large case studies provided by the Danish company Danfoss. Such case studies are beyond the scope of existing verification tool sets. One of the major problems of Ecdar is that the code largely depends on the one of Uppaal which is not open source. To fix this issue, we decided to make PyEcdar, a new tool on interface theories independent from Ecdar. PyEcdar implements all the technologies of timed interfaces and several game-based features that are not under Ecdar. The tool is flexible. In another work [CDL+11], we have proposed Constraint Markov Chains (CMC), a new specification theory for Markov Chains (MC). This new model permits rich constraints on probability

distributions and thus generalizes prior abstractions such as Interval MCs. This is the first specification theory for MCs with such closure properties. This work has then been generalized to also handle nondeterminism. The new model, which we call Abstract Probabilistic Automata (APA) [DKL+13] is a specification theory for Markov Decision Processes. Alternative models of timed and stochastic interfaces have also been considered. Finally in [FL14], we have been adapting interface theories to take quantities into account. This includes energy consumption or costs.
Link with the thesis and with our vision: As outlined in our vision, interface theories play a crucial role as a tool to abstract possibly unknown environment and should be part of any SoS language. In the near future, we will thus combine our work on interface theories with the one on Statistical Model Checking. Particularly, we will make sure that SMC exploits the composite nature of interface theories.

9.2.2 Software Product Lines

Software Product Lines (SPL) engineering is a software engineering paradigm that exploits the commonality between similar software products to reduce life cycle costs and time-to-market. Many SPLs are critical and would benefit from efficient verification through model checking. Model checking SPL is more difficult than for single systems, since the number of different products is potentially huge. In a series of recent work [CCH+14, CHL+14, CCS+13], we have introduced Feature Transition Systems (FTS), a formal, compact representation of SPL behavior, and provided efficient algorithms to verify FTS. Yet we still face the state space explosion problem, like any model checking based verification. Here, we tackle state space explosion through simulation-based model checking. We define a new simulation relation for FTS and provide means of computation. We extend well-known simulation preservation properties to FTS. We evaluate our approach by assessing state space size reduction and compare costs of FTS-based simulation with respect to single product verification. Results demonstrate that FTS are a solid foundation for formal validation of SPL. Our approach is the first to handle real-life size product lines. Wee have been developing extensions for real-time [CSHL12], and stochastic systems [ACL+15]. One promising direction is to combine the stochastic work done on software product lines with statistical model checking.
Link with the thesis and with our vision: Variability plays a crucial work in SoS who automatically reconfigure to react to environment changes. We will thus need to combine our work on variability with the one on interface theories to extract a formal model for SoS.

9.2.3 Information Quantification flow

Information theory provides a powerful quantitative approach to measuring security and privacy properties of systems. By measuring the *information leakage* of a system security properties can be quantified, validated, or falsified. When security concerns are non-binary, information theoretic measures can quantify exactly how

much information is leaked. The knowledge of such informations is strategic in the developments of component-based systems.

The quantitative information-theoretical approach to security models the correlation between the secret information of the system and the output that the system produces. Such output can be observed by the attacker, and the attacker tries to infer the value of the secret by combining this information with its knowledge of the system.

Armed with the produced output and the source code of the system, the attacker tries to infer the value of the secret. The quantitative analysis we implement computes with arbitrary precision the number of bits of the secret that the attacker will expectedly infer. This expected number of bits is the information leakage of the system.

The quantitative approach generalizes the qualitative approach and thus provides superior analysis. In particular, a system respects non-interference if and only if its leakage is equal to zero. In practice very few systems respect non-interference, and for those who dont it is imperative to be able to distinguish between the ones leaking a very small amount of bits and the ones leaking a significant amount of bits, since only the latter are considered to pose a security vulnerability to the system.

Since black box security analyzes are immediately invalidated whenever an attacker gains information about the source code of the system, we assume that the attacker has a white box view of the system, meaning that it has access to the systems source code. This approach is also consistent with the fact that many security protocol implementations are in fact open source.

The scope of modern software projects is too large to be analyzed manually. For this reason we provide tools that can support the analyst and locate security vulnerabilities in large codebases and projects. In a series of work [BLTW13, BLN+14, BLMW13], we have proposed Quail, a tool that can be used to quantify privacy of components. Quail is the only tool able to perform an arbitrary-precision quantitative analysis of the security of a system depending on private information. Thanks to its Markovian semantics model, Quail computes the correlation between the systems observable output and the private information, obtaining the amount of bits of the secret that the attacker will infer by observing the output. Quail is open source and can be downloaded at `https://project.inria.fr/quail/`.

Quail is able to evaluate the safety of randomized protocols depending on secret data, allowing to verify a security protocols effectiveness. Quail can also be used to find previously unknown security vulnerabilities in software systems and security protocols. The tool can verify whether a protocol is protecting its secret in a perfect way, and quantify how much the secret is exposed to being revealed otherwise.

Quail has been used to quantify whether voting protocols respect the anonymity of the voters, proving that preference ranking voting schemes are more secure than single preference ones. It has also been applied to the security of smart grids and a number of classic examples like dining cryptographers, authentication protocols and grades protocol.

Link with the thesis and with our vision: As said in our vision, checking

the security of a set of interconnected objects is crucial. Unfortunately, techniques developed so far suffers from the same problem than classical model checking approaches. We thus have the ambition to amplify the applicability of our approach by combining it with SMC.

Bibliography

[A.99] Rajeev A. Timed automata. In *CAV*, volume 1633 of *LNCS*. Springer, 1999. 5

[ABL13] A. Arnold, B. Boyer, and A. Legay. Contracts and behavioral patterns for systems of systems: The EU IP DANSE approach. In *AiSoS*. EPTCS, 2013. 86, 92, 93, 94

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *TCS*, 138(1):3–34, 1995. 99, 101

[ACL+15] V. Alves, M. Cordy, A. Lanna, A. Legay, V. Nunes, P-Y. Schobbens, G. Rodrigues, and A. Shariffloo. Modeling and verification for dependability in software product lines. In *HASE*. IEEE, 2015. 116, 127

[AD94] R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994. 99

[AdA95] A. and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *FSTTCS*, LNCS. Springer, 1995. 63, 64

[AM11] K. Ahnert and M. Mulansky. Odeint – Solving Ordinary Differential Equations in C++ . *AIP Conference Proceedings*, 1389(1):1586–1589, 2011. 56

[ATP01] R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In *HSCC*, LNCS. Springer, 2001. 99

[BBB+07] C. Baier, N. Bertrand, P. Bouyer, T. Brihaye, and M. Größer. Probabilistic and topological semantics for timed automata. In *FSTTCS*, volume 4855 of *LNCS*. Springer, 2007. 100, 103, 105

[BBB+12] A. Basu, S. Bensalem, M. Bozga, B. Delahay, and A Legay. Statistical abstraction and model-checking of large heterogeneous systems. *Int. J. Softw. Tools Technol. Transf.*, 14(1):53–72, February 2012. 116

[BBBM08] N. Bertrand, P. Bouyer, T. Brihaye, and N. Markey. Quantitative model-checking of one-clock timed automata under probabilistic semantics. In *QEST*. IEEE Computer Society, 2008. 100, 103, 105

[BCCZ99] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without bdds. In *TACAS*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999. 5

[BCLS13] B. Boyer, K. Corre, A. Legay, and S. Sedwards. Plasma-lab: A flexible, distributable statistical model checking library. In *QEST*, volume 8054 of *LNCS*, pages 160–164, 2013. 6, 12

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. and Comp.*, 98(2):142–170, 1992. 4

[BDHK04] H. Bohnenkamp, P.R. D'Argenio, H. Hermanns, and J.-P. Katoen. Modest: A compositional modeling formalism for real-time and stochastic systems. Technical Report CTIT 04-46, University of Twente, 2004. 100

[BDL+12a] P. E. Bulychev, A. David, K. G. Larsen, A. Legay, G. Li, D. Bøgsted Poulsen, and A. Stainer. Monitor-based statistical model checking for weighted metric temporal logic. In *LPAR*, volume 7180 of *LNCS*. Springer, 2012. 112

[BDL+12b] P. E. Bulychev, A. David, K. G. Larsen, M. Mikucionis, D. Poulsen, A. Legay, and Z. Wang. UPPAAL-SMC: statistical model checking for priced timed automata. In *QAPL*, volume 85 of *EPTCS*, 2012. 112

[Bel57] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957. 64, 65

[BFH+01] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. W. Vaandrager. Minimum-cost reachability for priced timed automata. In *HSCC*. ACM, 2001. 99

[BFHH11a] J. Bogdoll, L. Fioriti, A. Hartmans, and H. Hermanns. Partial order methods for statistical model checking and simulation. In *Formal Techniques for Distributed Systems*. Springer, 2011. 66

[BFHH11b] J. Bogdoll, L-M Fiorti, A. Hartmanns, and H. Hermanns. Partial order methods for statistical model checking and simulation. In *FORTE*, LNCS. Springer, 2011. to appear. 101

[BHHK03] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003. 5

[BHP12] B. Barbot, S. Haddad, and C. Picaronny. Coupling and importance sampling for statistical model checking. In Cormac Flanagan and Barbara König, editors, *TACAS*, volume 7214 of *LNCS*. Springer, 2012. 19

[BJK+05] M. Broy, B. Jonsson, J-P. Katoen, M. Leucker, and A. Pretschner, editors. *Model-Based Testing of Reactive Systems, Advanced Lectures The volume is the outcome of a research seminar that was held in Schloss Dagstuhl in January 2004*, volume 3472 of *Lecture Notes in Computer Science*. Springer, 2005. 3

[BK08] C. Baier and J-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008. 3, 63, 64, 116

[BLMW13] F. Biondi, A. Legay, P. Malacaria, and A. Wasowski. Quantifying information leakage of randomized protocols. In *VMCAI*, volume 7737 of *LNCS*. Springer, 2013. 116, 119, 128

[BLN+14] F. Biondi, A. Legay, B. F. Nielsen, P. Malacaria, and A. Wasowski. Information leakage of non-terminating processes. In *FSTTCS*, volume 29 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. 119, 128

[BLTW13] F. Biondi, A. Legay, L-M. Traonouez, and A. Wasowski. QUAIL: A quantitative security analyzer for imperative code. In *CAV*, volume 8044 of *LNCS*. Springer, 2013. 128

[BMR05] T. Ball, T. D. Millstein, and Sriram K. Rajamani. Polymorphic predicate abstraction. *ACM Trans. Program. Lang. Syst.*, 27(2), 2005. 4

[BN93] M. Basseville and I. V. Nikiforov. *Detection of Abrupt Changes: Theory and Application*. Prentice-Hall, Inc., 1993. 37, 40, 41

[Bry92] R. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Survey*, 24(3):293–318, 1992. 4

[CB06] F. Ciesinski and C. Baier. Liquor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *Proc. of 3rd Int. Conference on the Quantitative Evaluation of Systems (QEST)*, pages 131–132. IEEE, 2006. 5

[CCH+14] A. Classen, M. Cordy, P. Heymans, A. Legay, and P-Y. Schobbens. Formal semantics, modular specification, and symbolic verification of product-line behaviour. *Sci. Comput. Program.*, 80:416–439, 2014. 116, 127

[CCLRS09] T. Cormen, Charles E. C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009. 69

[CCS+13] A. Classen, M. Cordy, P-Y. Schobbens, P. Heymans, A. Legay, and J-F. Raskin. Featured transition systems: Foundations for verifying variability-intensive systems and their application to LTL model checking. *IEEE Trans. Software Eng.*, 39(8), 2013. 116, 127

[CDL08] E. M. Clarke, A. Donzé, and A. Legay. Statistical model checking of mixed-analog circuits with an application to a third order delta-sigma modulator. In *HVC*, volume 5394 of *LNCS*, pages 149–163. Springer, 2008. 101

[CDL+11] B. Caillaud, B. Delahaye, K. G. Larsen, A. Legay, M. L. Pedersen, and A. Wasowski. Constraint markov chains. *TCS*, 412(34), 2011. 126

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981. 4

[CES09] E.M. Clarke, E. A. Emerson, and J. Sifakis. Model checking: algorithmic verification and debugging. *Commun. ACM*, 52(11):74–84, November 2009. 64, 65

[CFG+13] A. Colombo, D. Fontanelli, D. Gandhi, A. De Angeli, L. Palopoli, S. Sedwards, and A. Legay. Behavioural Templates Improve Robot Motion Planning with Social Force Model in Human Environments. In *ETFA*. IEEE, 2013. 53, 54, 55, 58, 60

[CFGN09] H. Chockler, E. Farchi, B. Godlin, and S. Novikov. Cross-entropy-based replay of concurrent programs. In *Fundamental Approaches to Software Engineering, 12th International Conference, FASE 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5503 of *LNCS*, pages 201–215. Springer, 2009. 35

[CG04] F. Ciesinski and M. Größer. On probabilistic computation tree logic. In *Validation of Stochastic Systems*, LNCS, 2925, pages 147–188. Springer, 2004. 5

[CG07] F. Cérou and A. Guyader. Adaptive multilevel splitting for rare event analysis. *Stochastic analysis and applications*, 25, 2007. 27

[CGP99] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999. 4

[Che52] H. Chernoff. A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the sum of Observations. *Ann. Math. Statist.*, 23(4):493–507, 1952. 19, 20, 56, 74

[Che09] Cheng et al. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*, 2009. 85, 117

[CHL+14] M. Cordy, P. Heymans, A. Legay, P-Y. Schobbens, B. Dawagne, and M. Leucker. Counterexample guided abstraction refinement of product-line behavioural models. In *FSE*. ACM, 2014. 116, 127

[CMFA12] F. Cérou, P. Del Moral, T. Furon, and A.Guyader. Sequential Monte Carlo for rare event estimation. *Statistics and Computing*, 22, 2012. 26, 27

[CSHL12] M. Cordy, P-Y; Schobbens, P. Heymans, and A. Legay. Behavioural modelling and verification of real-time software product lines. In *SPLC*. ACM, 2012. 127

[CV03] E. M. Clarke and H. Veith. Counterexamples revisited: Principles, algorithms, applications. In *Verification: Theory and Practice, Essays Dedicated to Zohar Manna on the Occasion of His 64th Birthday*, volume 2772 of *LNCS*, pages 208–224. Springer, 2003. 4

[CY95] Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995. 5

[dAdSF+05] L. de Alfaro, L. Dias da Silva, M. Faella, A. Legay, P. Roy, and M. Sorea. Sociable interfaces. In *FROCOS*, volume 3717 of *LNCS*. Springer, 2005. 116, 118

[daH05] L. de alfaro and T. A. Henzinger. Interface-based design. In *Engineering Theories of Software Intensive Systems*, volume 195 of *NATO Science Series II: Mathematics, Physics and Chemistry*, pages 83–104. 2005. 126

[DAL] DALi project web site. http://www.ict-dali.eu/dali. 49, 60

[DAN13] DANSE. Designing for adaptability and evolution in sos engineering, dec 2013. 86

[DCL10] B. Delahaye, B. Caillaud, and A. Legay. Probabilistic contracts : A compositional reasoning methodology for the design of stochastic systems. In *ACSD*. IEEE, 2010. 85

[DDL+12] A. David, D. Du, K. G. Larsen, A. Legay, M. Mikucionis, D. Bøgsted Poulsen, and S. Sedwards. Smc for stochastic hybrid systems. In *HSB*, pages 122–136, 2012. 42

[DDL+13] Alexandre David, Dehui Du, Kim Guldstrand Larsen, Axel Legay, and Marius Mikucionis. Optimizing control strategy using statistical model checking. In *NASA Formal Methods*, volume 7871 of *LNCS*. Springer, 2013. 43

[DG07] Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theor. Comput. Sci.*, 380(1-2):69–86, 2007. 5

[DKL+13] B. Delahaye, J-P. Katoen, K. G. Larsen, A. Legay, M. L. Pedersen, F. Sher, and A. Wasowski. Abstract probabilistic automata. *Inf. Comput.*, 232, 2013. 127

[DLL+10a] A. David, K. .G. Larsen, A. Legay, U. Nyman, and A. Wasowski. Timed I/O automata: a complete specification theory for real-time systems. In *HSCC*. ACM, 2010. 102, 126

[DLL+10b] Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. ECDAR: An environment for compositional design and analysis of real time systems. In *ATVA*, volume 6252, pages 365–370, 2010. 126

[DLL+11] A. David, K.G. Larsen, A. Legay, Z.Wang, and M. Mikucionis. Time for real statistical model-checking: Statistical model-checking for real-time systems. In *CAV*, LNCS. Springer, 2011. 6, 100

[DLLP13] A. David, K. G. Larsen, A. Legay, and D. Poulsen. Statistical model checking of dynamic networks of stochastic hybrid automata. *ECEASST*, 66, 2013. 112

[DM04] P. Del Moral. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. Probability and Its Applications. Springer, 2004. 22, 26

[DPP04] A. Dovier, C. Piazza, and A. Policriti. An efficient algorithm for computing bisimulation equivalence. *TCS*, 311(1-3), 2004. 4

[FG05] C. Flanagan and P. Godefroid. Dynamic partial-order reduction for model checking software. In *POPL*, pages 110–121. ACM, 2005. 4

[FHH+11] M. Franzle, E. Moritz Hahn, H. Hermanns, N. Wolovick, and L. Zhang. Measurability and safety verification for stochastic hybrid systems. In *HSCC*. ACM, 2011. 115

[FL14] U. Fahrenberg and A. Legay. General quantitative specification theories with modal transition systems. *Acta Inf.*, 51(5), 2014. 127

[FMU] Functional mock-up interface. available at https://www.fmi-standard.org/. 118

[Fre08] G. Frehse. Phaver: algorithmic verification of hybrid systems past hytech. *STTT*, 10(3):263–279, 2008. 99

[FvH07] L. Frans and Willem van Hoesel. *Sensors on speaking terms: schedule-based medium access control protocols for wireless sensor networks*. PhD thesis, University of Twente, June 2007. 110

[FvHM07] A. Fehnker, L. van Hoesel, and A. Mader. Modelling and verification of the lmac protocol for wireless sensor networks. In *Integrated Formal Methods*, volume 4591 of *LNCS*. Springer, 2007. 110

[GHJ+14] A. Kumar Ghosh, F. Hussain, S. Jha, C. James Langmead, and S. Kumar Jha. Discovering rare behaviours in stochastic differential equations using decision procedures: applications to a minimal cell cycle model. *IJBRA*, 10(4/5), 2014. 115

[GHSZ99] P. Glasserman, P. Heidelberger, P. Shahabuddin, and T. Zajic. Multilevel splitting for estimating rare event probabilities. *Oper. Res.*, 47(4):585–600, April 1999. 19

[Góm09] R. Gómez. A compositional translation of timed automata with deadlines to uppaal timed automata. In *Formal Modeling and Analysis of Timed Systems*, volume 5813, pages 179–194, 2009. 102

[GPS02] G.Cabodi, P.Camurati, and S.Quer. Can bdds compete with sat solvers on bounded model checking? In *Proc. of 39th Design Automation Conference (DAC)*, pages 117–122. ACM, 2002. 5

[GS05] R. Grosu and S. A. Smolka. Monte carlo model checking. In *TACAS*, volume 3440 of *LNCS*. Springer, 2005. 12

[GTJ+10] B. Grabiec, L-M. Traonouez, C. Jard, D. Lime, and O. H. Roux. Diagnosis using unfoldings of parametric time petri nets. In *FORMATS*, volume 6246 of *LNCS*. Springer, 2010. 43

[HCZ11] Younes H.., E. M. Clarke, and P. Zuliani. Statistical verification of probabilistic properties with unbounded until. In *Formal Methods: Foundations and Applications*, volume 6527 of *LNCS*, pages 144–160. Springer, 2011. 9, 86, 89

[HFMV02] D. Helbing, I. Farkas, P. Molnár, and T. Vicsek. Simulation of Pedestrian Crowds in Normal and Evacuation Situations. In Michael Schreckenberg and Som Deo Sharma, editors, *Pedestrian and Evacuation Dynamics*. Springer, 2002. 52, 53, 60

[HFV00a] D. Helbing, I. Farkas, and T. Vicsek. Freezing by heating in a driven mesoscopic system. *Phys. Rev. Lett.*, 84, 2000. 52, 53

[HFV00b] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 407, 2000. 49, 52, 53, 54, 56

[HJ94] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994. 64

[HM95] D. Helbing and P. Molnár. Social force model for pedestrian dynamics. *Phys. Rev. E*, 51, 1995. 52, 53

[HMZ+12] David Henriques, Joao G. Martins, Paolo Zuliani, André Platzer, and Edmund M. Clarke. Statistical model checking for Markov decision processes. In *Quantitative Evaluation of Systems, 2012 Ninth International Conference on*, pages 84–93. IEEE, 2012. 66, 76

[Hor19] W. G. Horner. A new method of solving numerical equations of all orders, by continuous approximation. *Philosophical Transactions of the Royal Society of London*, 109, 1819. 69

[HR02] K. Havelund and G. Rosu. Synthesizing monitors for safety properties. In *TACAS*, volume 2280 of *LNCS*, pages 342–356. 2002. 9

[HT13] H. Hartmanns and M. Timmer. On-the-fly confluence detection for statistical model checking. In *NASA Formal Methods*. Springer, 2013. 66

[IJ90] A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizating mutual exclusion. In *PODC*. ACM, 1990. xii, 79, 80

[INR12] INRIA. Plasma-lab: a statistical model checker, December 2012. 56

[JCL+09] S. Kumar Jha, E. M. Clarke, C. J. Langmead, A. Legay, A. Platzer, and P. Zuliani. A bayesian approach to mc biological systems. In *CMSB*, pages 218–234, 2009. 101, 115, 116

[JKO+07] D. N. Jansen, J-P. Katoen, M. Oldenkamp, M. Stoelinga, and I. S. Zapreev. How fast and fat is your probabilistic model checker? an experimental performance comparison. In *HVC*, volume 4899 of *LNCS*. Springer, 2007. 9

[JLS12a] C. Jegourel, A. Legay, and S. Sedwards. A Platform for High Performance Statistical Model Checking – PLASMA. In Cormac Flanagan and Barbara König, editors, *TACAS*, volume 7214 of *LNCS*. Springer, 2012. 14, 19

[JLS12b] C. Jegourel, A. Legay, and S. Sedwards. Cross-Entropy Optimisation of Importance Sampling Parameters for Statistical Model Checking. In *CAV*, volume 7358 of *LNCS*, pages 327–342. Springer, 2012. 19, 21

[JRLD07] J. G. Jessen, J. I. Rasmussen, G. G. Larsen, and A. David. Guided controller synthesis for climate controller using uppaal tiga. In *FORMATS*, volume 4763 of *LNCS*. Springer, 2007. 43

[Kah50] Herman Kahn. Random sampling (Monte Carlo) techniques in neutron attenuation problems. *Nucleonics*, 6(5):27, 1950. 19

[KH51] H. Kahn and T. E. Harris. Estimation of Particle Transmission by Random Sampling. In *Applied Mathematics*, volume 5 of *series 12*. National Bureau of Standards, 1951. 19, 21

[KM53] H. Kahn and A. W. Marshall. Methods of Reducing Sample Size in Monte Carlo Computations. *Operations Research*, 1(5), November 1953. 19

[KMN02] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002. 65, 66

[KNP04] M. Z. Kwiatkowska, G. Norman, and D. Parker. Prism 2.0: A tool for probabilistic model checking. In *QEST*, pages 322–323. IEEE, 2004. 5

[KNP08] M. Kwiatkowska, G. Norman, and D. Parker. Analysis of a gossip protocol in PRISM. *SIGMETRICS Perform. Eval. Rev.*, 36(3), 2008. 77

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV'11*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011. 6, 74, 76, 86, 91

[KNPS06] M. Kwiatkowska, G. Norman, D. ParkerDavid, and G. Sproston. Performance analysis of probabilistic timed automata using digital clocks. *FMSD*, 29, 2006. 76

[KNPV09] M. Kwiatkowska, G. Norman, D. Parker, and M. G. Vigliotti. Probabilistic mobile ambients. *TCS*, 410(12-13), 2009. 76

[KNS02] M. Kwiatowska, G. Norman, and G. Sproston. Probabilistic Model Checking of the IEEE 802.11 Wireless Local Area Network Protocol. In *Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*. Springer-Verlag, 2002. 75

[Knu98] D. E. Knuth. *The Art of Computer Programming*. Addison-Wesley, 3rd edition, 1998. 68, 69

[KSB] V. Kaczmarczyk, M. Sir, and Z. Bradac. Stochastic timed automata simulator. In *4th European Computing Conference*. WSEAS. 111

[KSB10] V. Kaczmarczyk, M. Sir, and Z. Bradac. Stochastic timed automata simulator. 2010. In Proceedings of the 4th European Computing Conference. 101

[LP12] R. Lassaigne and S. Peyronnet. Approximate planning and verification for large Markov decision processes. In *Proc. 27th Annual ACM Symposium on Applied Computing*. ACM, 2012. 66

[LR81] D. Lehmann and M. Rabin. On the Advantage of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem (Extended Abstract). In *Proc. 8th Ann. Symposium on Principles of Programming Languages*, pages 133–138, 1981. 30

[LS89] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. In *POPL*, pages 344–352, 1989. 101

[LST14] A. Legay, S. Sedwards, and L.-M. Traonouez. Scalable verification of Markov decision processes. In *4th Workshop on Formal Methods in the Development of Software (FMDS 2014)*, LNCS. Springer, 2014. 64, 66, 67

[Mey92] B. Meyer. Applying "design by contract". *Computer*, 25(10):40–51, 1992. 85, 88

[Min99] M. Minea. *Partial Order Reduction for Verification of Timed Systems*. PhD thesis, Carnegie Mellon, 1999. 101

[MLK10] O. Maler, K. G. Larsen, and B. H. Krogh. On zone-based analysis of duration probabilistic automata. In *INFINITY*, volume 39 of *EPTCS*, pages 33–46, 2010. 100

[MU49] N. Metropolis and S. Ulam. The Monte Carlo Method. *Journal of the American Statistical Association*, 44(247):335–341, September 1949. 19

[NM10] U. Ndukwu and A. McIver. An expectation transformer approach to predicate abstraction and data independence for probabilistic programs. In *QAPL*, 2010. 76

[Nym08] U. Nyman. *Modal Transition Systems as the Basis for Interface Theories and Product Lines*. PhD thesis, Aalborg University, 2008. 126

[Oka59] Masashi Okamoto. Some inequalities relating to the partial sum of binomial probabilities. *Annals of the Institute of Statistical Mathematics*, 10:29–35, 1959. 11

[OMG10] OMG. Ocl v2.2, feb 2010. 92

[Pan10] P. Panangaden. *Labelled Markov Processes*. Imperial College Press, 2010. 104

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 46–57, 1977. 4

[Put94] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994. 64

[PvV11] D. Poulsen and J. van Vliet. Duration probabilistic automata. Technical report, Aalborg University, 2011. 111

[RdBSH12] D. Reijsbergen, P. de Boer, W. Scheinhardt, and B. Haverkort. Rare event simulation for highly dependable systems with fast repairs. *Performance Evaluation*, 69(7–8), 2012. 19

[RR55] M. Rosenbluth and A. W. Rosenbluth. Monte Carlo Calculation of the Average Extension of Molecular Chains. *Journal of Chemical Physics*, 23(2), February 1955. 19

[SMV] The smv model checker (SMV). Available at `http://www.kenmcmil.com/smv.html` . 99

[SPE08] SPEEDS. D 2.5.4: Contract specification language, apr 2008. 85

[SPI] The spin tool (spin). Available at `http://spinroot.com/spin/whatispin.html`. 99

[SVA04] K. Sen, M. Viswanathan, and G. Agha. Statistical model checking of black-box probabilistic systems. In *CAV*, LNCS 3114. Springer, 2004. 10

[SVA05] K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of stochastic systems. In *CAV*, pages 266–280, 2005. 9, 12, 86

[TEF11] T. Teige, A. Eggers, and M. Fränzle. Constraint-based analysis of concurrent probabilistic hybrid systems: An application to networked automation systems. *Nonlinear Analysis: Hybrid Systems*, 2011. 100

[UPP] The uppaal tool. Available at `http://www.uppaal.com/`. 99

[Var85] M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. 26th. Annual IEEE Symposium on Foundations of Computer Science*, 1985. 63

[VAVA91] M. Villén-Altamirano and J. Villén-Altamirano. RESTART: A Method for Accelerating Rare Event Simulations. In *Queueing, Performance and Control in ATM*. Elsevier, 1991. 19

[VHV08] G. Verdier, N. Hilgert, and J-P. Vila. Adaptive threshold computation for cusum-type procedures in change detection and isolation problems. *CS & DA*, 52(9), 2008. 40, 41

[Wal45a] A. Wald. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, 1945. 10, 11

[Wal45b] A. Wald. Sequential Tests of Statistical Hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, June 1945. 19, 74

[WG93] P. Wolper and P. Godefroid. Partial-order methods for temporal verification. In *CONCUR*, volume 715 of *LNCS*. Springer, 1993. 4

[Whi85] D. White. Real applications of Markov decision processes. *Interfaces*, 15(6), 1985. 63

[Whi88] D. White. Further real applications of Markov decision processes. *Interfaces*, 18(5), 1988. 63

[Whi93] D. White. A survey of applications of Markov decision processes. *Journal of the Operational Research Society*, 44(11), 1993. 63

[YCGS05] H. Younes, E. M. Clarke, G. J. Gordon, and J. G. Schneider. Verification and planning for stochastic processes with asynchronous events. Technical report, 2005. 86

[You05a] H. L. S. Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Carnegie Mellon, 2005. 6, 7, 10, 11, 101

[You05b] H. L. S. Younes. Verification and planning for stochastic processes with asynchronous events. PhD thesis, Carnegie Mellon University, 2005. 9, 16

[You05c] Håkan L. S. Younes. Ymer: A statistical model checker. In *CAV*, volume 3576 of *LNCS*, pages 429–433. Springer, 2005. 101

[You06] H. L. S. Younes. Error control for probabilistic model checking. In *Proc. of 7th Int. Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI*, LNCS 3855, pages 142–156. springer-verlag, 2006. 10

[YS02a] H. Younes and R. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *CAV*, volume 2404, pages 23–39. Springer, 2002. 12, 65

[YS02b] 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. 6, 101

[ZBC12] P. Zuliani, C. Baier, and E; M. Clarke. Rare-event verification for stochastic hybrid systems. In *HSCC*, pages 217–226, New York, NY, USA, 2012. ACM. 115

[ZC06a] J. Zhang and B. H. C. Cheng. Model-based development of dynamically adaptive software. In *ICSE*. ACM, 2006. 88

[ZC06b] J. Zhang and B. H.C. Cheng. Using temporal logic to specify adaptive program semantics. *Journal of Systems and Software*, 79(10), 2006. 88, 97

[ZPC10] P. Zuliani, A. Platzer, and E. M. Clarke. Bayesian statistical model checking with application to simulink/stateflow verification. In *HSCC*, pages 243–252. ACM ACM, 2010. 42, 101, 115