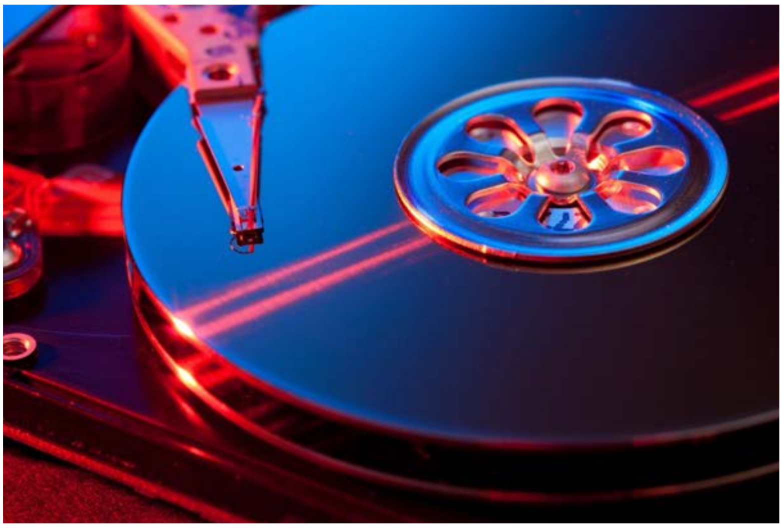
Quantitative Verification

Formal Guarantees for Timeliness, Reliability and Performance

A Knowledge Transfer Report from the London Mathematical Society and Smith Institute for Industrial Mathematics and System Engineering

By Gethin Norman and David Parker



Copyright © 2014 by Gethin Norman and David Parker

Image credits:

Front cover top left: Eternalfeelings / Shutterstock.com
Front cover top right: Vereshchagin Dmitry / Shutterstock.com
Front cover bottom left: Tiggy Gallery! / Shutterstock.com
Front cover bottom right: ESA / D Ducros
Front cover background: Serg64 / Shutterstock.com
Page 10: coloursinmylife / Shutterstock.com

QUANTITATIVE VERIFICATION

Formal Guarantees for Timeliness, Reliability and Performance

By Gethin Norman and David Parker

Contents

June 2014

A Knowledge Transfer Report from the London Mathematical Society and
the Smith Institute for Industrial Mathematics and System Engineering
Edited by Robert Leese and Tom Melham

London Mathematical Society, De Morgan House, 57–58 Russell Square, London WC1B 4HS
Smith Institute, Surrey Technology Centre, Surrey Research Park, Guildford GU2 7YG

AUTHORS

Gethin Norman is a Lecturer in Computing Science at the University of Glasgow and was previously a senior post-doctoral researcher at the University of Oxford. The focus of his research has been on the theoretical underpinning of quantitative formal methods, particularly models and algorithms for real-time and probability, and quality of service properties. He is a key contributor to the probabilistic verification tool PRISM, developing many of PRISM's modelling case studies across a wide range of application domains, finding several faults and anomalies. He is a member of the steering committees for the International Conference on Quantitative Evaluation of Systems (QEST) and the International Workshop on Quantitative Aspects of Programming Languages and Systems (QAPL) and has served on the programme committees of many of the well-known international verification conferences and workshops.

www.dcs.gla.ac.uk/people/personal/gethin/
gethin.norman@glasgow.ac.uk

David Parker is a Lecturer in Computer Science at the University of Birmingham. Prior to that, he was a senior post-doctoral researcher at the University of Oxford. His main research interests are in the area of formal verification, with a particular focus on the analysis of quantitative aspects such as probabilistic and real-time behaviour, and he has published over 90 papers in this area. Recent work includes efficient techniques for scalable verification (e.g. abstraction, compositionality), game-theoretic verification methods, and applications of these approaches to areas such as systems biology, DNA computing, computer security and robotics. He leads development of the widely used probabilistic verification tool PRISM, regularly serves on the programme committees of international conferences such as TACAS, SEFM, CONCUR, TASE and QEST, and frequently gives invited tutorials on quantitative verification at summer schools and workshops.

www.cs.bham.ac.uk/~parkerdx/
d.a.parker@cs.bham.ac.uk

Executive Summary

Computerised systems appear in almost all aspects of our daily lives, often in safety-critical scenarios such as embedded control systems in cars and aircraft or medical devices such as pacemakers and sensors. We are thus increasingly reliant on these systems working correctly, despite often operating in unpredictable or unreliable environments. Designers of such devices need ways to guarantee that they will operate in a reliable and efficient manner.

Quantitative verification is a technique for analysing quantitative aspects of a system's design, such as timeliness, reliability or performance. It applies formal methods, based on a rigorous analysis of a mathematical model of the system, to automatically prove certain precisely specified properties, e.g. "the airbag will always deploy within 20 milliseconds after a crash" or "the probability of both sensors failing simultaneously is less than 0.001".

The ability to formally guarantee quantitative properties of this kind is beneficial across a wide range of application domains. For example, in safety-critical systems, it may be essential to establish credible bounds on the probability with which certain failures or combinations of failures can occur. In embedded control systems, it is often important to comply with strict constraints on timing or resources. More generally, being able to derive guarantees on precisely specified levels of performance or efficiency is a valuable tool in the design of, for example, wireless networking protocols, robotic systems or power management algorithms, to name but a few.

This report gives a short introduction to quantitative verification, focusing in particular on a widely used technique called model checking, and its generalisation to the analysis of quantitative aspects of a system such as timing, probabilistic behaviour or resource usage.

The intended audience is industrial designers and developers of systems such as those highlighted above who could benefit from the application of quantitative verification, but lack expertise in formal verification or modelling. This report, in addition to explaining the basics of quantitative verification, highlights a variety of successful practical applications of these methods and provides suggestions as to how interested readers can learn more about these techniques, and engage with the researchers that are developing them. We hope that this will spur further advances in this rapidly advancing area and its applicability to industrial-scale problems.

Quantitative Verification: An Introduction

Formal verification

Formal verification is an approach for checking the correctness of a computerised system during its design phase. In contrast to *testing*, which checks that the system behaves correctly under a finite number of test cases, formal verification is designed to be exhaustive: it uses mathematical reasoning to guarantee the absence of errors. Conversely, when system errors do exist, formal verification can also serve as an effective bug-hunting technique.

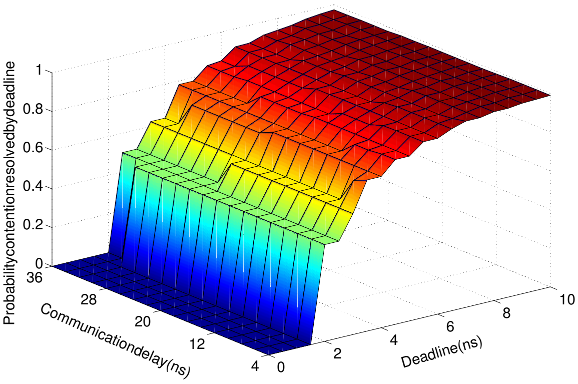


Figure 1: Quantitative verification, used for performance analysis of the root contention phase of the Firewire protocol [72].

Formal verification has become an essential part of the design phase in several industries. For example, in the development of integrated circuits, verification functionality in electronic design automation (EDA) tools is routinely used to eradicate errors that would be hugely expensive to fix later in the design process. In safety-critical domains such as the avionics industry, stringent regulations for certification have led to widespread usage of formal verification techniques to prove correctness of systems components.

Model checking [27] is a commonly used formal verification technique, which has been applied with great success to check the correctness of (and identify errors in), for example, hardware device drivers and both cryptographic and communication protocols. First, the correct behaviour of these systems is formally specified and then a mathematical model that captures all possible executions of the system is systematically constructed and analysed in order to verify that the correctness properties are satisfied. In some cases, these are abstract models, designed by hand and based on expert knowledge of the system; in others, they can be extracted directly, from source code or a high-level design document. A key appeal of model checking is that, once the model and its correctness properties have been specified, the verification process itself is typically fully automated.

Quantitative aspects of correctness

Computerised systems play a critical role in almost all aspects of our daily lives, from embedded control systems in cars and aircraft, to medical devices such as pacemakers and sensors, to the multitude of electronic devices that make up our communication networks. Often, these systems function in unpredictable or unreliable environments, yet we have become reliant on them working reliably and efficiently.

This means that *quantitative* (or “nonfunctional”) aspects of correctness have become increasingly important. Whether we are concerned with guarantees about the operation of a safety-critical system, or an analysis of the performance of a communication protocol (see, for example, Figure 1), verifying that systems function ‘correctly’ may require the ability to reason about:

- **Time**: Does the airbag successfully deploy within 20 milliseconds upon detection of a crash?
- **Probabilities**: Is the probability of successful message transmission greater than 0.99?
- **Resources**: Does the robot complete its mission without depleting its battery?

Quantitative verification generalises formal verification: it comprises a variety of techniques that can be used to produce formal guarantees about quantitative aspects of system behaviour, such as reliability, performance or timeliness. In this report, we give an overview of these techniques, with a particular emphasis on *timed* and *probabilistic* variants of model checking.

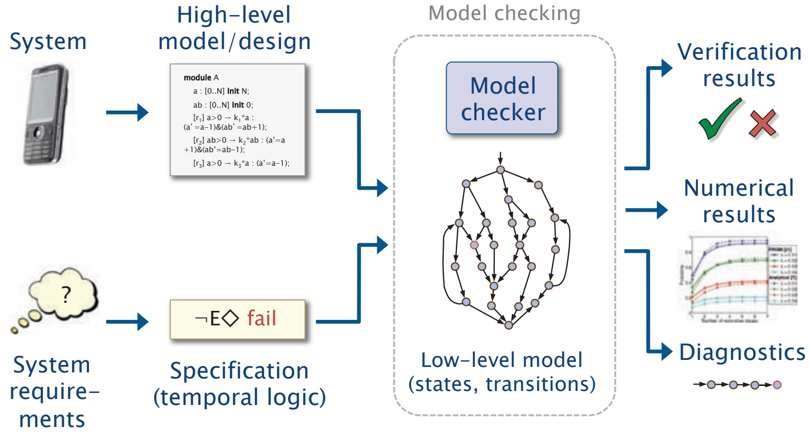
Model checking for real-time and probabilistic systems

Model-checking techniques, originally designed to prove or disprove the absence of errors in a system, have now been extended to reason in a more quantitative fashion about correctness.

The basic approach (see Figure 2 for an overview) remains to construct a *model* of a system's behaviour, but now incorporating information about the timing or likelihood of events that may occur. Typically, from the user's point of view, this is done using a *high-level* modelling language, often one that is specific to the model-checking software being used. From this, the software exhaustively constructs and analyses the underlying *low-level* model, i.e. it explores all the possible configurations (*states*) that the system can be in and the ways in which it can evolve between these states (*transitions*).

Real-time systems. When precise constraints on the timing of events are needed, *timed automata* are a popular high-level model. These allow the modeller to specify the delays that occur as transitions take place between states. Examples of their use include modelling real-time control systems for manufacturing processes or automotive applications.

Probabilistic systems. Another key ingredient when modelling and verifying quantitative aspects of a system's behaviour is *probability*. This can be used to model many sources of uncertainty, for example the potential failure of a physical component, the time delay associated with transmitting data across a busy wireless channel, or the presence of noise from unreliable sensors in an embedded system. Some systems also explicitly incorporate probabilistic behaviour through the use of randomisation: e.g. random back-off in wireless protocols and probabilistic network routing for efficiency or anonymity.



```
module A
    a : [0..N] init N;
    ab : [0..N] init 0;
    [r1] a>0 → k1*a :
(a' =a−1)&(ab' =ab+1);
    [r2] ab>0 → k2*ab : (a'=a
+1)&(ab'=ab−1);
    [r3] a>0 → k3*a : (a'=a−1);
```

Figure 2: An overview of the model-checking framework.

Many different models exist to capture the probabilistic behaviour of systems. Prominent examples include (discrete-time or continuous-time) *Markov chains* and *Markov decision processes*.

Property specifications. For verification, the required behaviour of a system needs to be formally specified. This is often done using *temporal logics* (but more user-friendly formats also exist - see Box 7). Examples include TCTL (Timed Computation Tree Logic) for real-time systems and PCTL (Probabilistic Computation Tree Logic) or CSL (Continuous Stochastic Logic) for probabilistic systems. An example of a PCTL specification and its meaning is:

- $detect \Rightarrow \mathrm{P}_{\geq 0.999}[\Diamond^{\leq 20}\, deploy]$: "With probability at least 0.999, the airbag successfully deploys within 20 milliseconds upon detection of a crash."

Verifying that models satisfy such properties is achieved using model-checking software. The most widely used examples for real-time and probabilistic systems are UPPAAL and PRISM, respectively.

Box 1: History of Key Developments

Late 80s: Basic theory of probabilistic model checking, e.g. [108].

1990: Timed automata formalism, for modelling real-time systems, proposed by Alur & Dill [7].

Mid 90s: Probabilistic logic PCTL and model-checking algorithms proposed [52, 16].

Mid 90s: Zone-based data structures developed to allow efficient analysis of timed automata [77].

1995: First release of the UPPAAL model checker, for modelling and verification of timed automata.

2000: First versions of the probabilistic model checkers PRISM and ETMCC (now MRMC).

Early 2000s: Probabilistic model checking extended to continuous-time Markov chains, for performance and reliability [8, 9].

2000s: New variants of UPPAAL for cost-based models, synthesising controllers and efficient, compositional verification.

2011: PRISM 4.0 adds support for systems with both probabilistic *and* real-time behaviour [69].

What Can Be Done with Quantitative Verification?

Quantitative verification techniques have been successfully deployed in a variety of application domains, both to prove correctness and to find bugs or anomalies. We identify three key areas of application:

- safety-critical systems;
- performance/reliability;
- scheduling/optimisation,

and illustrate each area with a number of success stories, concentrating on those with industrial connections.

Safety-critical systems

Safety-critical systems are those where failure is deemed to be unacceptable, for example because it could lead to loss of life or major environmental damage. Examples can be found in domains such as avionics, nuclear energy, process control, robotics, transport and medicine.

In many cases, strict constraints are imposed upon the *correctness* or *reliability* of computerised systems in these domains, for example by industrial regulations or certification procedures. Quantitative verification can be used to generate formal guarantees about such aspects of the design of a safety-critical system.

Automotive systems. Box 2 describes an example of using probabilistic verification to formally assess reliability in the context of a safety-critical system from the automotive domain.

Box 2: Failure Analysis for an Airbag System

Automotive systems are required to operate under strict safety constraints. Researchers at the University of Konstanz and Swinburne University, in conjunction with industrial partners TRW Automotive GmbH, used probabilistic model checking as part of a failure analysis for a car airbag system, as reported in [4]. Their approach was based on the FMEA (Failure Mode and Effects Analysis) process [1], one of the first systematic approaches for failure analysis developed by reliability engineers. FMEA analyses potential failures of system components, assessing and ranking the risks associated with them, and then identifying and addressing the most serious problems. The FMEA process can be time-intensive and the analysis is sometimes informal. Quantitative verification was applied in order to overcome these limitations. The analysis was based on the *probabilistic FMEA process* of Grunske et. al. [50], which combines FMEA with probabilistic model checking and the generation of probabilistic counterexamples [5].

The airbag system consists of three major component types: sensors, crash evaluators and actuators. The sensors are used to detect accidents such as impacts or the car rolling, and the information from the sensors is then processed by two independent crash evaluators. If both evaluators agree that a crash has occurred, then the actuators respond by deploying the airbags. The use of a second crash evaluator is a recent addition to airbag systems, aimed at avoiding unnecessary deployment, which is seen as the most dangerous malfunction that can occur. The failure analysis considered variants of the airbag system with both one and two crash evaluators.

The probabilistic model checker PRISM was used to construct models of the two different variants of the system, with their behaviour being modelled using a continuous-time Markov chain. Probabilistic FMEA analysis was then performed on the models. The requirements for the system were developed based on a draft of the ISO standard 26262 for road vehicles, which states that the airbag system must comply with ASIL D (Automated Safety Integration Level D) for unintended deployment. The requirements were formalised using the temporal logic CSL and then verified against the models using PRISM. The analysis found that certain ASIL D requirements were violated in the one-processor variant of the system. Furthermore, using counterexample generation and visualisation, the critical aspect of this violation was identified as the failure of the micro-processor.

An evaluation of the work carried out concluded that PRISM's modelling language could be learned quickly by the engineers on the project to specify models. However the same could not be said for property specification using the CSL logic. To resolve this, property specification patterns [49] (see Box 7) were used. Other limiting factors were the size of the system models that needed to be constructed and the time required to verify some of the CSL properties.

Real-time communication. Another example is the use of the real-time systems model-checking tool UPPAAL to model and verify the commercial real-time communications protocol AF100 (Advant Field-bus 100) [28], developed and implemented by ABB for safety-critical applications. This case study is one of the largest to which UPPAAL has been applied and the verification had to be applied to a sequence of models, representing different levels of abstraction.

The conclusion of this work was that, although it is possible to implement the communications protocol such that its specified requirements are satisfied, care must be taken to avoid certain race conditions (e.g. situations where more than process tries to access a resource at the same time) and delays. The analysis also demonstrated several imperfections in the protocol's logic and implementation. The sources of these errors were then debugged using abstract models of the protocol and solutions to these problems were given.

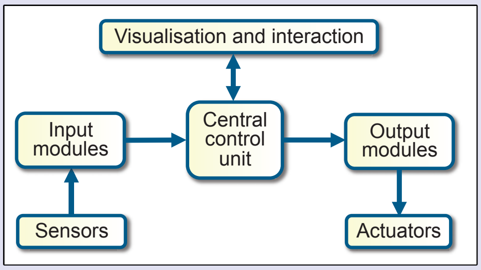
Performance and reliability

Our second highlighted area for the application of quantitative verification techniques is the formal analysis of *performance* and *reliability* properties. Temporal logics such as PCTL, and in particular CSL, have proved to be an effective way of formally capturing a wide range of useful properties, and probabilistic verification has been used for their analysis in a variety of domains.

Process control. Box 3 gives an illustration of this kind of analysis: a reliability analysis of an industrial process control system, which includes application of the probabilistic model checker PRISM.

Box 3: Reliability of an Industrial Process Control System

Engineers from ABB Corporate Research, working with the EU-funded research project Q-ImPrESS, applied probabilistic verification to study the reliability of a large-scale process control system used in a variety of industrial settings, including power generation, chemical processes and material handling [67]. The control system is used to access the sensors and actuators of an industrial process. It provides a graphical visualisation of important values from the sensors and allows interaction with the actuators (e.g. opening or closing valves). In total, the control system comprises over 3 million lines of code and 100 components, structured into 9 subsystems.



The study applied an architecture-based software reliability analysis (ABSRA) approach [47, 48], the main objective of which is to obtain an estimate of the overall reliability of a software application through the reliability of both the application's individual components and its architecture. These methods are also used to identify the most critical components of the application and quantify their influence on the overall reliability.

The reliability of individual components was estimated using failure reports from ABB customers and the Littlewood-Verrall software reliability growth model [82]. The overall system was modelled as a discrete-time Markov chain and sensitivity analysis was performed using the probabilistic model checker PRISM. The sensitivity analysis corresponded to finding the influence of the failure rates of individual components on the overall failure rate of the system, and hence how critical each component is to the overall reliability of the system.

One important motivation for this case study was to provide an empirical evaluation of the ABSRA approach on a larger scale industrial software system than had previously been considered, and to assess the benefits and costs of using these methods. It was found that the main effort was in data collection and processing, and that performing accurate modelling was an expensive process. In addition, there was a trade-off in terms of the level of abstraction at which the system was modelled: a high level of abstraction ensures that verification is tractable, but this comes at the cost of the usefulness of the results of the analysis. Further case studies and details of a more general framework for model-driven reliability analysis of component-based systems are presented in [26], including a discussion of "feedback provisioning", which analyses the results of quantitative verification in order to try to improve reliability.

Cloud computing. Researchers at Fujitsu used probabilistic model checking to analyse the performance of resource management operations in cloud computing [65]. Based on performance data collected from an experimental virtualised system, a continuous-time Markov chain model of the system was built, and performance properties were expressed in CSL. The probabilistic model checker PRISM was used for the analysis. The results showed that quantitative verification gave cloud computing service administrators a way to analyse the management operations, and hence help in providing efficient and reliable services.

Satellite platforms. The COMPASS project, working with the European Space Agency (ESA) developed a tool chain to analyse a modern satellite platform [38], including the probabilistic model checker MRMC. This study considered a variety of different behavioural characteristics, including discrete, real-time, hybrid and stochastic features, and used a range of quantitative (and nonquantitative) verification methods to analyse performance, reliability, correctness and safety.

Scheduling and optimisation

A further use of quantitative verification techniques is to solve *scheduling* or *optimisation* problems. Rather than analysing the timeliness or performance of an existing, fully specified system, verification methods and tools are used for the reverse problem of finding a system configuration that satisfies (or optimises) a particular performance criterion.

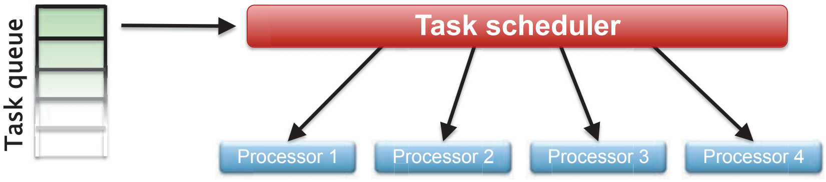


Figure 3: Structure of an example task scheduling problem.

Scheduling or resource allocation problems occur in many different application domains, including transport, manufacturing, telecommunications networks and parallel computing.

In general, a scheduling problem involves finding a way to assign a set of tasks to a set of resources, typically under certain constraints on the ordering in which tasks can be performed and the number of tasks that can be assigned to resources simultaneously (see Figure 3 for an example problem). The goal is to execute all tasks, while either satisfying a given constraint on time or resource usage, or in a way that minimises execution time or resource usage. Closely related is the dual problem of finding the worst case execution time (WCET), i.e. the maximum time taken to complete all tasks, given some additional constraints on how the tasks are scheduled.

Finding solutions to these problems is often addressed in fields such as operational research, artificial intelligence and queueing theory, using techniques that include constraint programming, genetic algorithms and mixed integer linear programming. However, quantitative verification methods, in particular using (priced) timed automata, have been shown to provide a competitive alternative [2, 14].

Box 4 outlines an industrial case study using quantitative verification to perform schedulability analysis on a satellite system.

Additional case studies using this approach are discussed below.

Real-time Java. In [17] the authors use the SARTS tool to perform schedulability analysis on hard real-time Java programs. SARTS automatic translates these programs to timed automata models (written in the input language for UPPAAL). A WCET analysis of a Java program is then performed by analysing the corresponding timed automaton. The approach is fully automated, so there is no need for the user to understand the underlying model-checking process.

This case study demonstrated that quantitative verification is comparable with alternative approaches for performing WCET analysis and in fact can yield more accurate results than the traditional techniques. However, the improvement in accuracy does come at a cost, both in terms of the time required for the analysis and its scalability. To combat the latter, compositional approaches have since been developed to analyse safety-critical Java programs [18].

Hydraulic pump controllers. An industrial case study concerning a controller for a hydraulic oil pump provided by the HYDAC company can be found in [24]. The controller aims both to keep oil and gas pressure levels within safe intervals and to minimise the energy accumulated in the system. Controllers were synthesised using UPPAAL, verified using the tool PHAVer and simulated with Simulink [101]. They were shown to outperform the controllers designed by HYDAC, while also being robust and provably correct.

Power Management. Dynamic power management is the use of runtime strategies to achieve a tradeoff between the performance and power consumption of a system and its components. In [89], a framework to synthesise and analyse dynamic power management strategies using probabilistic model checking is presented. Figure 4 illustrates the optimal expected power consumption given varying constraints on the average request queue size and expected number of lost requests for an IBM TravelStar VP disk-drive. We see that tightening the performance constraints (requiring the average queue size to be smaller or that fewer requests are lost) leads to a "less optimal" controller (since the power consumption increases).

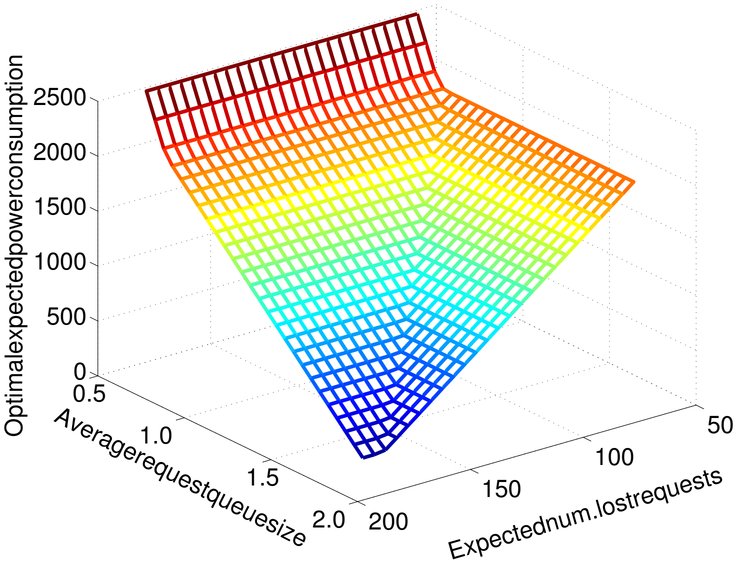


Figure 4: An analysis of optimal power management policies for an IBM TravelStar VP disk-drive using probabilistic model checking [89].

Box 4: The Herschel-Planck Satellite System

A good example of the applicability of quantitative verification to scheduling problems is the use of the timed model checker UPPAAL to analyse schedulability in the Herschel-Planck satellite system [85]. Each satellite comprises a single processor, which runs a real-time operating system and has a selection of software tasks to execute with deadlines. The goal was to investigate whether these tasks could be scheduled on the single processor without violating any of the deadlines. To achieve this, an earlier formulation of this scheduling problem [92] was encoded as a timed automaton. The model captures the behaviour of the processor's scheduler, the software tasks that need to be executed, and various resource usage measures and deadlines. Required properties of the schedule, including worst-case blocking and response times, processor utilisation and deadline satisfaction, are modelled within the timed automaton model and UPPAAL then is used to analyse the model and produce a schedule that satisfies the constraints. The resulting schedule can be visualised graphically as a Gantt chart using UPPAAL's simulator.

An interesting conclusion of the work was that the quantitative verification approach with timed automata yielded less pessimistic response time estimates than classical techniques, such as those in [92]. This meant that it was able to produce a task schedule that satisfied all the required constraints in cases where alternative scheduling techniques could not find such a solution. In particular, one conclusion of [92] was that a particular software task on the satellite could potentially violate its deadline requirements, although this behaviour had never been seen in practice. The analysis in [85] demonstrated that the deadline requirements were actually met.

Other applications

Security. Quantitative behaviour is an important aspect of modelling and analysing security protocols. For example, time delays between communications may unwittingly leak hidden information or an attacker may be able to guess a password with some probability.

One example of applying quantitative verification to the security domain is [104], which studies the *PIN blocks* used to encrypt and transmit customers' PINs in banking networks. The probabilistic model checker PRISM was used to detect the most effective ways to construct PIN block attacks, consisting of sequences of API commands that enable an attacker to determine the value of a PIN from an encrypted PIN block.

Other applications of quantitative verification to the security domain include anonymity networks [100], access control mechanisms [88], denial of service threats [12], information flow [45], quantum cryptography [41] and contract signing [90].

Communication, network and multimedia protocols. Again, in this setting, quantitative aspects of system behaviour play an important role. For example, randomisation is often used to break symmetry between devices communicating with the same protocol, and real-time constraints frequently appear in protocols or the mediums under which they are designed to operate. Quantitative verification case studies in this area include routing protocols [40], network configuration [46, 70], Bluetooth device discovery [36] and collision avoidance schemes [62, 71].

Further examples of applications of quantitative verification include gearbox controllers [81], web services [97], cardiac pacemakers [25], robotics [75] and systems biology [53].

Quantitative Verification: In Depth

Model checking for real-time systems

Automated formal verification of systems whose behaviour depends on *real-time* constraints is often performed using a modelling formalism called *timed automata*, first proposed by Alur and Dill in the early 90s [7].

A timed automaton is a state transition system augmented with real-valued *clocks*. States (which are referred to as *locations*) represent different possible configurations of the system being modelled. Transitions between these locations, representing the ways that the system can evolve, are annotated with *guards*, which constrain when the transitions can be taken, based on the current values of the clocks. Locations are also labelled with *invariants*, indicating how long can be spent in a particular location before a transition must be taken.

Box 5 illustrates the use of timed automata to model a real-time system: a train controller. This example also demonstrates another prominent feature of the formalism: models are typically constructed as *networks* of timed automata, representing different components of the system. The automata can communicate with each other by sending data through *channels*.

To verify that a system modelled using timed automata behaves correctly, we perform model checking. First, the required properties of the system are specified formally, typically using temporal logic. A simple example of a property is "it is impossible to reach the location Error", which can be expressed in the temporal logic CTL (Computation Tree Logic) as $\mathtt{A}[\Box(\neg\mathit{Error})]$. A wider range of quantitative properties can be expressed using the logic TCTL [6], e.g. $\mathtt{A}[\Box(\mathit{Send} \Rightarrow \mathtt{A}[\Diamond^{\leq 200}\mathit{Ack}])]$ – "whenever location Send is visited, we are guaranteed to reach location Ack within 200 milliseconds". In this report, we omit precise details of the notation for these logics, instead just giving simple illustrative examples. See the pointers given in the "Next Steps" section for information.

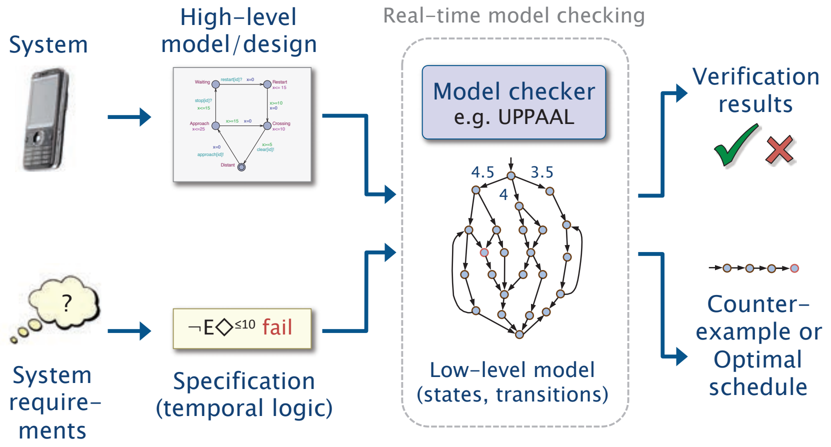

System
High-level model/design
Real-time model checking
Model checker e.g. UPPAAL
4.5
3.5
4
Low-level model (states, transitions)
Verification results
✓ ✗
Counter-example or Optimal schedule
?
System require-ments
¬E◇≤10 fail
Specification (temporal logic)

Figure 5: An overview of model checking for real-time systems.

Properties such as these can be verified on a timed automaton, through a systematic exploration of the possible executions of the model. In order to scale this approach up to models of realistic size, *symbolic* techniques were developed, which efficiently represent the possible values that clocks can take using *zones*. Thanks to these methods and to mature software support in the form of tools like UPPAAL [78] and KRONOS [20], timed automata are widely used and have been successfully applied to verify, for example, real-time controllers and real-time communication protocols.

Timed automata can also be extended in various ways. For example, *weights* or *prices* can be added and used to reason about resource usage or energy consumption; and *timed game* variants of the model can be used to model uncontrollable or adversarial aspects of a system's environment. Separate branches of UPPAAL (UPPAAL CORA and UPPAAL TIGA, respectively) have been developed to verify these models.

These extended models have proved to be particularly successful for *scheduling* and *controller synthesis* problems. Here, rather than verifying the correct behaviour of an existing system, timed automata are used to construct optimal or provably correct solutions to problems such as scheduling a set of tasks against a set of real-time constraints, or constructing real-time controllers for embedded systems. See [19] for an overview.

Box 5: Timed Automata - An Example

Figures 6 and 7 show timed automata used to model and verify a train controller which coordinates the safe passage of two trains across a single section of track running over a bridge.

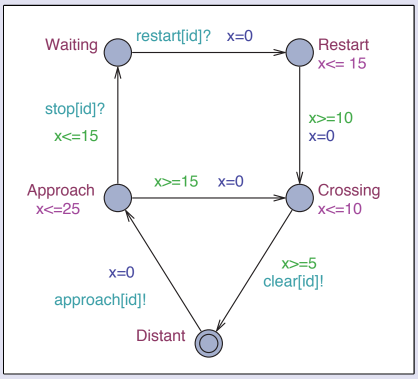


Figure 6: Train automaton.

Figure 6 refers to a single train. We model two trains, using two copies of this automaton, with the *id* parameter set to either 0 or 1. The initial location, indicated by a double circle, is *Distant*, representing the situation where the train is far away from the bridge. When the train nears, the automaton moves to the *Approach* location, notifying the controller of this using channel *approach*[*id*]. Sending and receiving on a channel are denoted by the symbols ! and ?, respectively. The timed automaton also has a local clock, x, which is reset to 0 at this point. While approaching, if no more than 15 time units have elapsed (i.e. the train is not too close to the bridge to stop), the train can be stopped by the controller, via channel *stop*[*id*], resulting in a transition to location *Waiting*. Otherwise, the train will cross the bridge after between 15 and 25 time units, moving to the location *Crossing*. Notice how the time constraints are encoded: the transitions to *Waiting* and *Crossing* have guards $x \leq 15$ and $x \geq 15$, indicating when they can be taken. The *Approach* location also has an invariant $x \leq 25$, meaning that it must be left before clock x exceeds 25 time units. If the train has been stopped, it waits for communication on channel *restart*[*id*], and then starts crossing after between 10 and 15 time units. Crossing the bridge takes between 5 and 10 seconds, after which the train notifies the controller (via channel *clear*[*id*]) and returns to the location *Distant*.

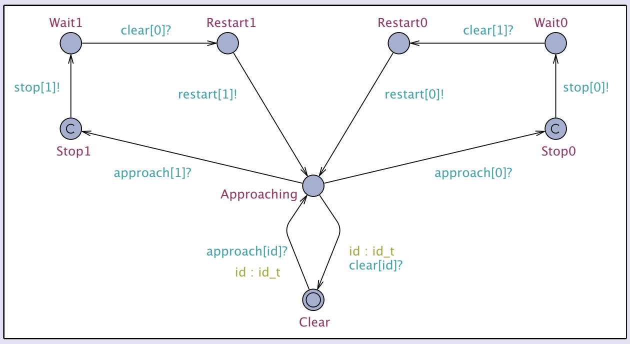


Figure 7: Controller automaton.

The timed automaton for the controller is given in Figure 7. It keeps track of whether the trains are approaching, stopped or travelling over the bridge. The controller starts in location *Clear*, corresponding to the situation where the bridge is clear and no trains are approaching.

If a single train approaches and crosses the bridge before another train arrives, the controller will receive a message over the channel *approach*[*id*] move to the location *Approach*, then receive a message over *clear*[*id*] and return to location *Clear*.

On the other hand, if two trains approach the bridge at the same time (i.e. if two successive messages along channels *approach*[*id*] are received, and none along a channel *clear*[*id*]), then the controller moves to location *Stop0* or *Stop1* after which it immediately instructs one of the trains to stop (using the channel *stop*[*id*]) and moves to *Wait0* or *Wait1*). This immediacy is encoded in the automaton by making *Stop0* and *Stop1* *committed locations* (labelled by a "c"), implying that the location must be left immediately, without further time elapsing. When the bridge again becomes clear (i.e. the controller receives a message on the channel *clear*[*id*]), the controller moves to the location *Restart0* or *Restart1*, instructs the stopped train to restart through the channel *restart*[*id*] and moves back to *Approaching*.

Example correctness properties for the model described here, written in TCTL, include:

- $\mathtt{A}[\Box\neg(\mathit{Train}(0).\mathit{Crossing} \land \mathit{Train}(1).\mathit{Crossing})]$ – "At most one train at a time will be crossing the bridge"
- $\mathtt{A}[\Box(\mathit{Train}(0).\mathit{Approach} \Rightarrow \mathtt{A}[\Diamond \mathit{Train}(0).\mathit{Crossing}])]$ – "Whenever train 0 approaches, it eventually crosses"
- $\mathtt{A}[\Box(\mathit{Train}(0).\mathit{Waiting} \Rightarrow \mathtt{A}[\Diamond^{\leq 30}\mathit{Train}(0).\mathit{Restart}])]$ – "Train 0 is only ever stopped for at most 30 time units"

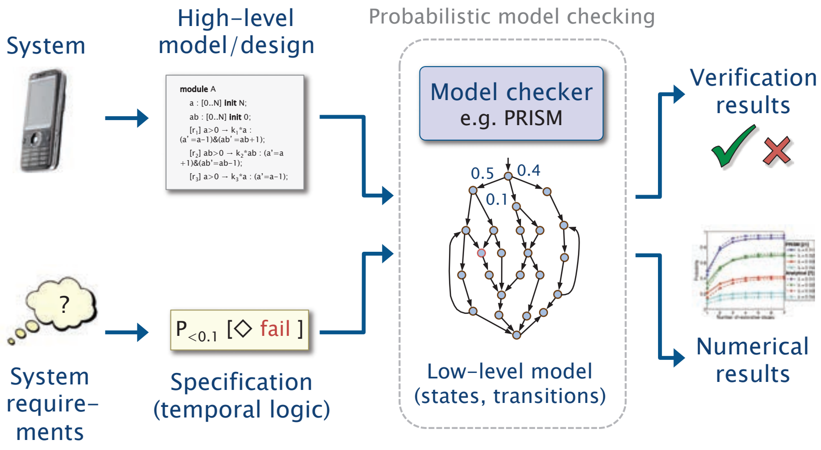
Probabilistic model checking

Probabilistic model checking (sometimes known as stochastic model checking) is a generalisation of model checking for verifying quantitative properties of systems which exhibit stochastic behaviour, for example due to failures or uncertainty about the environment.

Like other forms of model checking, it is based on the rigorous construction and analysis of a system model (see Figure 8). In this case the models are probabilistic, i.e. they capture the likelihood that each possible execution of the system occurs. The simplest type of model is a *discrete-time Markov chain* (DTMC), which can be thought of as a state transition system where the transitions between states are labelled with the probability that they are taken. Box 6 illustrates the use of a DTMC to model an embedded system whose components can fail.

Another popular type of model is a *continuous-time Markov chain* (CTMC), which captures not just the probability of making transitions between states, but also of the delays incurred before making transitions. These random delays are represented using exponential probability distributions, making CTMCs well suited to the modelling and analysis of, for example, performance and reliability of computer systems.

A third common model is a *Markov decision process* (MDP), as used in fields such as control theory and robotics. MDPs allows us to model the effect on our system of a separate entity such as a controller, e.g. for a robotic system.



```
module A
    a : [0..N] init N;
    ab : [0..N] init 0;
    [r1] a>0 → k1*a :
(a' =a–1)&(ab' =ab+1);
    [r2] ab>0 → k2*ab : (a'=a
+1)&(ab'=ab–1);
    [r3] a>0 → k3*a : (a'=a–1);
```

Figure 8: An overview of probabilistic model checking.

We can formalise the required properties of these models using probabilistic temporal logics. Typically, these properties capture not just "correctness", but a variety of quantitative characteristics of the system, such as reliability or performance. Examples, using the probabilistic temporal logic PCTL [52], include:

- $\mathrm{P}_{<0.01}\,[\,\Diamond(\mathit{Fail}_1 \wedge \mathit{Fail}_2)\,]$
  "The probability of both sensors eventually failing simultaneously is less than 0.01"
- $\mathit{Sent} \Rightarrow \mathrm{P}_{\geq 0.95}\,[\,\Diamond^{\leq 10}\,\mathit{Arrive}\,]$
  "Every packet that is sent arrives within 10ms with probability at least 0.95"

In practice, it is also common to use numerical variants of such properties which, for example, query the actual probability of an event occurring, rather than checking that its above or below a specified threshold:

- $\mathrm{P}_{=?}\,[\,\Diamond(\mathit{Fail}_1 \wedge \mathit{Fail}_2)\,]$
  "What is the probability of both sensors eventually failing simultaneously?"

Probabilistic model checking is used to evaluate the result of queries such as those above, based on an analysis of a probabilistic model of system behaviour. Crucially, this analysis is performed in a *rigorous* fashion: models are explored exhaustively to check for all possible executions and then queries are solved through numerical solution methods, for example by solving systems of linear equations or linear optimisation problems.

This is in contrast to *simulation* techniques, which can produce approximate answers, computed by averaging over a large sample of simulated system executions.

A trade-off between these techniques can be achieved using *statistical model checking* [79], which performs statistical methods on simulations to provide approximate results to formally specified verification queries.

Prominent software tools for probabilistic model checking include PRISM [69] and MRMC [64]. These have been used to verify quantitative properties of a wide variety of real-life systems, from wireless communication protocols [36], to aerospace designs [21], to DNA circuits [76].

Box 6: Markov chains - An Example

Discrete-time Markov chains (DTMCs) can be used to model a wide range of systems with probabilistic behaviour. Here, we illustrate their application to model an embedded system comprising a processor which reads and processes data from two sensors.

At each clock cycle, a variety of failures can occur, whose probability we can estimate based on known or measured failure rates. There is a chance of a single sensor failing, for which the probability is dependent on the number of sensors currently operational. The processor uses these sensors in dual modular redundancy, meaning that it can function effectively as long as at least one of the two sensors is operational. If both become unavailable, then during the next clock cycle the processor shuts the system down. The processor itself also has a probability of failure during a clock cycle. This can be either a permanent fault or a transient fault. In the latter case, the situation can be rectified automatically by the processor rebooting itself during the next clock cycle. Rebooting also has a chance of failure and if this happens, then the processor just repeatedly tries during each clock cycle to reboot until successful.

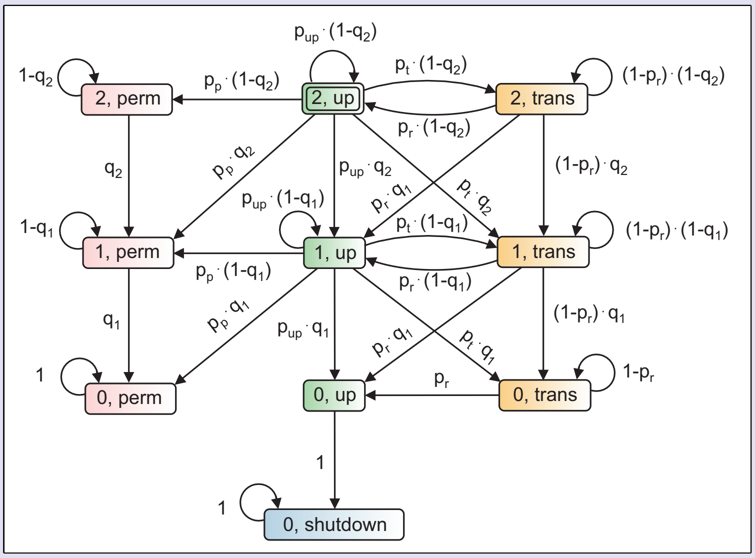


Figure 9: DTMC model of an embedded system.

Figure 9 illustrates a DTMC model of the embedded system. Each state is labelled by the number of operational sensors (0, 1, or 2) and the status of the processor: operational (*up*), transient fault (*trans*), permanent fault (*perm*) or shutdown (*shutdown*). Initially all components are working. The probabilities that the processor suffers a temporary or permanent fault during a clock cycle are p_t and p_p, respectively, and $p_{up} = 1 - (p_t + p_p)$ is the probability that neither occur. In addition, p_r denotes the probability the processor successfully reboots and q_i is the probability that a sensor fails when i are currently operational. Notice that in the DTMC it is possible both for a sensor to fail and for the processor to fail or repair, and since we suppose these happen independently, the probability of both occurring is given by multiplying the probability of the individual events.

To specify properties of this model, we associate some states with *labels* (or *atomic propositions*, to use the terminology of temporal logic). We attach $fail_s$ to states where both sensors have failed and $fail_p$ to those where the processor has permanently failed. Example properties in the temporal logic PTCL include:

- $\mathrm{P}_{<0.01}$ [$\diamond$ $(fail_s \wedge fail_p)$] – “The probability that the processor and sensors eventually fail is less than 0.01”
- $\mathrm{P}_{=?}$ [$\diamond$ $(fail_s \wedge fail_p)$] – “What is the probability that the processor and sensors eventually fail?”
- $\mathrm{P}_{=?}$ [$\diamond^{\leq 1000}$ $fail_s$] – “What is the probability that both sensors fail within the first 1000 cycles?”

Modelling and property specification languages

As mentioned earlier, model checkers typically take as input a *high-level* description of a system or its design, from which they explore and construct a *low-level* model of the states and transitions than can arise.

A variety of *modelling languages and formalisms* exist for this purpose. Often, they are specific to a piece of software. For example, Figure 10 shows a screenshot of UPPAAL, whose user interface includes a graphical editor for designing networks of timed automata. A separate textual description attached to these automata describes additional variables used in the model, and can incorporate fragments of C code to specify how they are updated.

Figure 11 gives an example of the modelling language used by the PRISM tool. This is a textual language, used to describe a variety of different types of probabilistic models.

Other tools and tool chains sometimes use adaptations or extensions of more mainstream modelling languages. An example is the QuantUM toolset [80], which uses UML (Unified Modeling Language) and SysML (Systems Modeling Language), extended with additional quantitative information such as failure probabilities. Figure 12 shows screenshots from QuantUM. A similar approach, mentioned earlier, is the COMPASS toolset, which uses a customised version of AADL (Architecture Analysis & Design Language) [22].

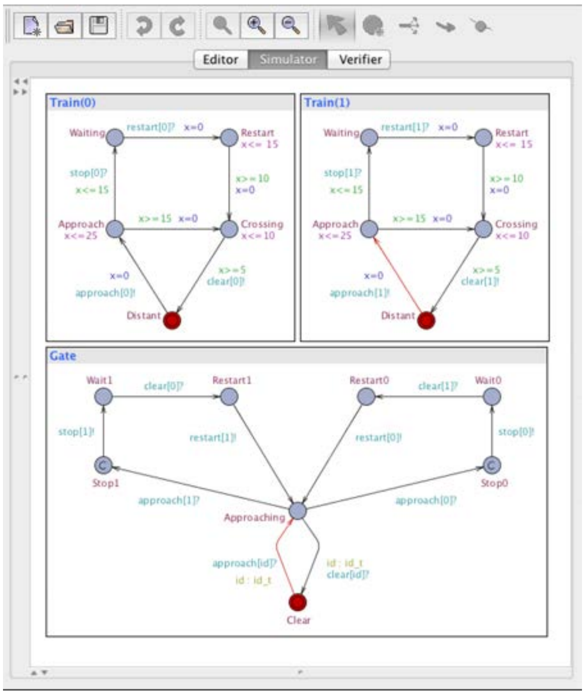


Figure 10: A screenshot of the real-time model checker UPPAAL, showing graphical editing of a collection of timed automata.

```
dtmc

const double q2; // probability of sensor failure (2 sensors operational)
const double q1; // probability of sensor failure (1 sensors operational)

const double pt; // probability of a transient failure of the processor
const double pp; // probability of a permanent failure of the processor
const double pup = 1 - (pt+pp); // process remains operational

const double pr; // probability processor repaired (transient fault)

// module for sensors behaviour
module sensors

        s : [0..2] init 2; // operational sensors

        // sensor can fail when system has not shut down
        [cycle] s=2 & p>0 -> q2 : (s'=s-1) + 1-q2 : (s'=s);
        [cycle] s=1 & p>0 -> q1 : (s'=s-1) + 1-q1 : (s'=s);

        // loop when system has shut down
        [cycle] s=0 & p>0 -> 1 : (s'=s);

endmodule

module processor

        p : [0..3] init 3; // 0 - shutdown, 1 - perm, 2 - trans, 3 - up

        // when up and some sensors operational can fail
        [cycle] p=3 & s>0 -> pup : (p'=3) + pt : (p'=2) + pp : (p'=1);
        // when up and no sensors operational go to shutdown
        [cycle] p=3 & s=0 -> 1 : (p'=0);

        // repair from transient fault
        [cycle] p=2 -> pr : (p'=3) + (1-pr) : (p'=2);

        // loop when system has shut down or permanent failure
        [cycle] p<2 -> 1 : (p'=p);

endmodule
```

Figure 11: A description, in PRISM's textual modelling language, representing the Markov chain model from Box 6.

Model checkers also require specifications of the properties to be checked of a model. Temporal logics provide a precise and unambiguous way of specifying a wide range of properties: various examples have been illustrated in this document. However, to users who are unfamiliar with such formalisms, this approach to property specification can be rather unintuitive.

In practice, the majority of properties needed can be expressed using a relatively small set of different classes of logical formulae. A good solution is therefore to use *patterns*: commonly occurring classes of formulae, typically identified by studying a large set of system properties extracted from requirements documents and/or verification case study reports. Box 7 shows some examples.

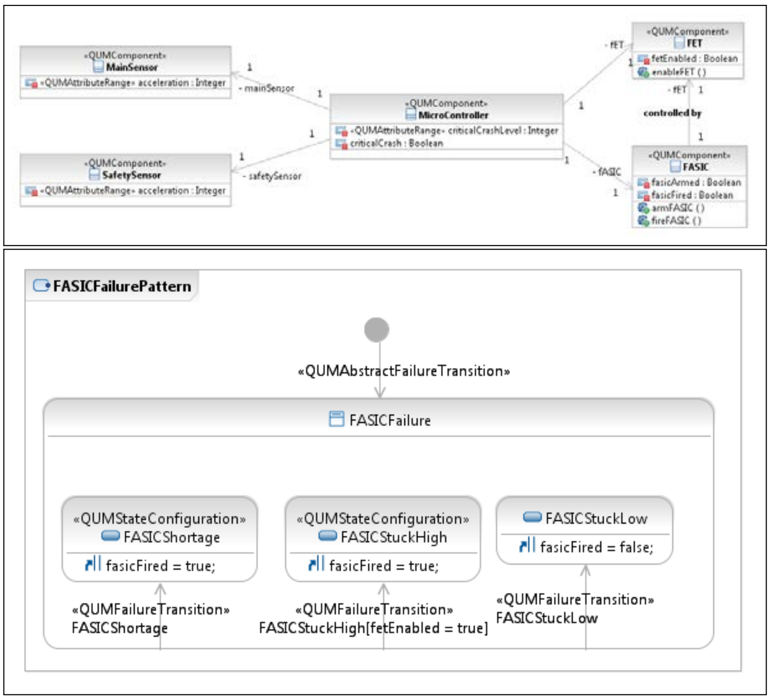


Figure 12: Screenshots from the QuantUM toolset, showing its UML-based modelling approach: the structure of an airbag model (top); and a specification of failures for a system component (bottom).

Box 7: Quantitative Property Specifications using Patterns

The table below shows *patterns* for timed and probabilistic properties, as proposed by Konrad/Cheng [66] and Grunske [49], respectively. The top two are timed and use the logic TCTL; the bottom four are probabilistic and use the logic CSL. Colours indicate parameters to the patterns that are provided by the user.

Pattern name	**Example (temporal logic formula & meaning)**	
Bounded recurrence	$\mathtt{A}[\Box(\mathtt{A}[\Diamond^{\leq 100}\, \mathit{poll}]]$	"The controller polls the server for messages at least every 100 milliseconds."
Bounded response	$\mathtt{A}[\Box(\mathit{sense} \Rightarrow \mathtt{A}[\Diamond^{\leq 30}\, \mathit{notif}])]$	"Whenever the sensor detects a movement, the processor is notified within 30 milliseconds."
Probabilistic invariance	$\mathrm{P}_{\geq 0.999}[\Box^{\leq 3600}\, \neg \mathit{fail}]$	"With probability at least 0.999, no server failure will occur in the next hour."
Probabilistic response	$\mathrm{P}_{\geq 1}[\Box(\mathit{send} \Rightarrow \mathrm{P}_{\geq 0.95}[\Diamond^{\leq 2}\, \mathit{rec}])]$	"Every time a message is sent, the probability of it being successfully received within 2 second is at least 0.95."
Transient state probability	$\mathrm{P}_{>0.99}[\Diamond^{=t}\, \mathit{both}]$	"The probability of both sensors being operational at time t is greater than 0.99."
Steady-state probability	$\mathrm{S}_{\leq 0.05}[\neg \mathit{min\_qos}]$	"The long-run probability of a minimum service level not being delivered is at most 0.05."

Current Challenges

Quantitative verification is an active and growing research area. We outline below a few of the key challenges facing the area, focusing on those in which progress is likely to improve the applicability of the techniques to industrial-scale problems and thus to a wider audience.

Scalability and efficiency are always a challenge for verification, particularly for techniques like model checking which are based on an exhaustive exploration of a model. Often, the burden falls on the user of a verification tool to model a system at the right level of abstraction such that its analysis is feasible, but this can be difficult for users new to the area.

Many fully-automated techniques are being developed to improve scalability and efficiency. Examples include automatic generation of model *abstractions*; *compositional* methods (decomposing verification tasks); *parallel, distributed or GPU-based* approaches; and *simulation-based* methods such as statistical model checking. Typically, these start out as prototype software implementations and then gradually make their way into more established tools.

Mainstream languages. As mentioned above, most current quantitative verification tools require the user to describe the system to be verified in a tool-specific modelling language. Some tools have at least partial support for more mainstream languages, for example UPPAAL timed automata models can incorporate fragments of C code and other tools support variants of UML-based languages.

Generally, though, to make quantitative verification accessible to a wider audience, there is a need to support more mainstream programming and modelling languages. As has been found in the field of nonquantitative verification, sophisticated verification methods are needed to cope with the complexity of programs developed in more expressive, mainstream languages.

Cyber-physical systems comprise embedded sensing or control systems that interact closely with their physical environment. Examples include smart grids, medical monitoring devices and autonomous automotive or avionic systems. Clearly, in such application domains, there is a need for rigorous guarantees on safety or performance levels, but the complexity of such systems makes applying quantitative verification a major challenge.

Since system models need to combine discrete aspects (for computation devices) and continuous aspects (for their physical environment), *hybrid systems* are an appropriate class of models. Techniques and tools are under development for various subclasses of such models, but much remains to be done.

Next Steps

Get involved

We hope that this report has provided a good illustration of the potential benefits of applying quantitative verification techniques, and of the wide range of possible application domains to which they can be applied.

Many quantitative verification software tools, along with guidance for their usage and background information about the techniques that they use, are openly available. We provide some pointers in the sections below.

Of course, investigating the applicability of these methods to new problems or domains, especially without background knowledge of the area, can be challenging. An excellent basis for achieving this is through collaboration with the researchers who are actively working in the area of quantitative verification and who are in general very keen to tackle industrially relevant problems.

Effective ways to go about achieving such collaborations include:

- acting as an industrial partner on a research project;
- sponsoring, co-sponsoring or co-supervising a PhD student;
- providing details and expertise regarding an industrially relevant case study.

We include, in Appendix 2, a list of research groups actively working in quantitative verification, along with pointers to web sites with details of the people involved and their contact details.

Hands-on experience

The best way to understand what can be done with quantitative verification is to get some hands-on experience with the tools and techniques on offer. In Appendix 1, we provide a list of relevant software tools supporting the techniques described in this report, most of which are freely available.

Tutorials for these tools provide a good way to learn about their features and their modelling or specification languages.

- The UPPAAL website [107] has a “documentation” section with pointers to several papers (e.g. [13]) with detailed descriptions of the software.
- The PRISM website [95] has a series of online tutorials with step-by-step instructions for working with different features of the tool and for analysing various kinds of models.

For other tools, see the web links given in Appendix 1.

Case studies also provide a good practical introduction. Both the UPPAAL and PRISM websites (see above) include case study repositories, which give an indication of the application domains to which the tools are applicable and the problems that can be solved. In some cases, model files are provided, which can then be experimented with in the software, or used as a basis for a related case study.

User support for many tools in this area is provided by online forums or mailing lists, populated by users and tool developers/maintainers.

Further reading

The “In Depth” section of this report provides some insight into the underlying models and techniques used in quantitative verification. This helps users to understand, for example, which kind of model is most appropriate for a given scenario, or which techniques/tools may be most efficient for a given problem. There is a wealth of material available covering these topics in greater detail.

Textbooks in the area mostly offer broader coverage. A good starting point is the book [11], which covers the fundamentals for all aspects of model checking, including chapters on timed and probabilistic systems. We also mention [99], which provides a detailed tutorial for many aspects of probabilistic model checking, and [84], which gives an introduction to different modelling approaches, analysis techniques and tools for real-time systems.

Tutorial papers, often freely available on the web, are another good source of in-depth learning material. See [15] for an overview of the semantic and algorithmic aspects of verification tools for timed automata. For probabilistic model checking, [68] provides a detailed introduction to the modelling and verification of (discrete- and continuous-time) Markov chains, while [43] focuses on model checking of Markov decision processes, as well as a number of more advanced topics, and [91] covers models with both probabilistic and real-time aspects.

We also recommend [19] and [10, 63], which offer gentler introductions to the areas of verification for timed and probabilistic systems.

Conferences and workshops

The techniques described in this report are mostly still active research topics. Conferences and workshop venues in the area provide an up-to-date view of the latest developments. They are also a good source of information about successful or novel verification case studies. Typically, they also welcome industrial contributions and participation.

Focused events, with a specific emphasis on quantitative modelling and verification, include:

- International Conference on Quantitative Evaluation of SysTems (**QEST**) [60];
- International Conference on Formal Modeling and Analysis of Timed Systems (**FORMATS**);
- International Conference on Hybrid Systems: Computation and Control (**HSCC**) [55].

In addition, the major international conferences on formal methods include sessions on quantitative topics. Examples include:

- European Joint Conferences on Theory and Practice of Software (**ETAPS**) [39];
- International Conference on Computer Aided Verification (**CAV**) [59].

These events, and others, often include more focused workshops on quantitative verification, for example, the International Workshop on Quantitative Aspects of Programming Languages and Systems (**QAPL**).

Appendix 1: Quantitative Verification Tools

We list below some of the most well-known software tools for quantitative verification, summarising what each offers and what support and documentation are available. We restrict our attention to general-purpose tools, rather than those that implement a specific verification technique. We also only list tools that are currently available (in most cases, for free) and are still actively developed. We group the tools according to the types of models that they support, since this is usually the key factor to consider when selecting the most appropriate tool.

Real-time Verification Software

UPPAAL [107] is an integrated tool environment for modelling, validation and verification of real-time systems modelled as networks of timed automata. The tool is free for noncommercial applications in academia. The website provides links to a discussion forum, a large range of case studies and small examples, manuals and research papers. There are also links to extensions of UPPAAL including: **TIMES**, for schedulability analysis and synthesis; **CORA**, for cost-optimal reachability analysis; **TRON**, for black-box conformance testing; and **TIGA**, for solving timed game automata. Recently, support has been added for probabilistic aspects of timed automata using statistical model-checking techniques.

Probabilistic Verification Software

PRISM [95] is a probabilistic model checker, which can be used to model, analyse and verify many types of stochastic models, such as discrete-time and continuous-time Markov chains and Markov decision processes. It recently added support for probabilistic real-time systems, modelled as probabilistic timed automata. PRISM is free and open source, and has been used to analyse systems from many different application domains, including communication and multimedia protocols, security protocols, dynamic power management schemes, biological systems and many others. The website includes a case study repository, documentation (FAQ, manual, tutorials, lectures), research papers and a discussion/help group.

MRMC [83] is a probabilistic model checker with a particular focus on systems modelled as discrete-time and continuous-time Markov chains, augmented with reward information. It supports verification of a variety of different probabilistic temporal logics. MRMC is freely available and the website provides a manual, research papers and information on related tools.

Modest Toolset [87] supports the modelling and analysis of hybrid, real-time, distributed and stochastic systems. The toolset is freely available and provides a modular framework centered around the stochastic hybrid automata formalism [51]. It provides a variety of input languages and backend tools, including: **mcpta**, which connects to **PRISM** for probabilistic real-time model checking; **mctau**, which connects to **UPPAAL** for real-time model checking; **modes**, which performs simulation-based analysis; **mosta**, a visualisation tool; and **ProHVer**, which connects to **PHAVer** for analysis of probabilistic hybrid systems. The website provides documentation on the language, research papers and a small number of case studies.

Further Related Tools

CADP [23] is the “Construction and Analysis of Distributed Processes” toolset, for the design of asynchronous concurrent systems, such as communication protocols, distributed systems, asynchronous circuits, multiprocessor architectures and web services. The primary focus is on nonquantitative verification, but support for some probabilistic models is also included. The toolset contains more than 42 tools and 17 software components and is is free for academic use. The website includes links to manuals, tutorials, many case studies, research papers and user forums.

Möbius [86] is a multi-formalism modelling and analysis toolset for stochastic systems. Although originally developed for the study of reliability, availability and performance of computer and network systems, it is now used for a broad range of discrete-event systems, from biochemical reactions networks to the effects of malicious attackers on secure computer systems. The tool is free for academic use and the website provides links to a manual and academic papers.

Appendix 2: Active Research Groups

Below, we give details of some of the research groups actively working in topics related to quantitative verification, both in the UK and internationally. The list is not intended to be exhaustive and we apologise in advance to anyone we have omitted.

In the UK

- **Universities of Birmingham, Oxford and Glasgow:** The PRISM tool [95] has been developed by groups based at these three sites. More generally, the research carried out includes theory and practice, involving the development of formalisms, theories, algorithms and tools, and their application to real-world case studies.

  Oxford also has a large *Automated Verification* group [29], incorporating a *Quantitative Analysis and Verification* group [96], working on a variety of areas including the verification of real-time, probabilistic, hybrid and infinite-state systems.

- **University of Edinburgh:** *PEPA* group [93], part of the *Laboratory for Foundations of Computer Science* [74]. These groups have expertise in topics such as process algebraic techniques, continuous-time models and verification of infinite-state probabilistic systems.

- **Imperial College London:** *Analysis, Engineering, Simulation and Optimization of Performance (AESOP)* group [3]. The main areas of interest are analytic and simulated solutions to real performance problems.

- **University of Newcastle:** *Dependability* group [30]. This group investigates fundamental concepts, development techniques, models, architectures and mechanisms that directly contribute to creating modern information systems, networks and infrastructures that are dependable and secure in all aspects.

Internationally

- **RWTH University of Aachen:** *Software Modeling and Verification (MOVES)* group [102]. Focuses on the modelling and verification of trustworthiness aspects (such as safety, reliability, performance and survivability) of software systems by applying mathematical theories and methods.

- **Aalborg University:** *Distributed and Embedded Systems (DES)* group [35]. This group is concerned with the modelling, analysis and realization of computer programs, with an emphasis on distributed and embedded systems.

- **ENS Cachan:** *Laboratoire Spécification et Vérification (LSV)* [73]. Research focuses on the verification of computerised systems, of databases and of security protocols, developing the mathematical and algorithmic foundations for tools to automatically prove correctness and detect flaws.

- **Carnegie Mellon University:** *Formal Methods* [44]. Research covers various aspects of formal methods with a particular focus on model checking, including verification of hybrid systems and statistical model checking.

- **Universidad de Córdoba:** *Dependable Systems* group [32]. Focuses on techniques of dependability (safety, reliability, availability, security) of computer systems through formally based specification and analysis techniques.

- **Technische Universität Dresden:** *Algebraic and Logical Foundations of Computer Science* group [105]. Research is focused on analysis and model-checking algorithms for quantitative systems, temporal and modal logics, automata-based approaches, game theory and infinite-state systems.

- **INRIA:** *Preuves et Vérification* group [57]. Incorporates several research teams working on a variety of topics within quantitative verification, including probabilistic and real-time systems.

- **IST Austria:** *Henzinger* [54] and *Chatterjee* [61] groups. Research covers the design and analysis of concurrent and embedded systems, verification and game theory, including a focus on quantitative aspects of these topics.

- **Masaryk University:** *Institute for Theoretical Computer Science* [58]. Research focuses on stochastic systems, probabilistic temporal logics and game theory.
- **Technische Universität München:** *Foundations of Software Reliability and Theoretical Computer Science* group [106]. The group is interested in all aspects of software reliability, with special emphasis on model checking and program analysis techniques.
- **University of New South Wales and Macquarie University:** *Specification and Development of Probabilistic Systems* [103]. The group is concerned with formal specification and refinement of probabilistic systems and recent work has focused on information security and anonymity.
- **University of Oldenburg:** *Hybrid Systems* group [56]. The group's focus is on algorithms and tools for the verification of hybrid systems.
- **University of Pennsylvania:** *Penn Research in Embedded Computing and Integrated Systems Engineering (PRECISE)* centre [94]. This centre focuses on developing new modelling formalisms for analysing reliability of computer-based systems; algorithms and tools for efficient analysis of these formalisms; and real-world case studies drawn from a range of applications.
- **University of Saarland:** *Dependable Systems and Software* [31] and Reactive Systems [98] groups. Research focuses on the design-time assurance of performance and dependability for reactive, embedded, distributed and mobile systems and computer-aided methods for the synthesis and verification of reactive systems.
- **University of Torino:** [34] *Performance Evaluation and System Validation* group. The group's research concerns the development of tools and techniques for performance evaluation and system validation with focus on techniques for modelling, performance evaluation and probabilistic verification.
- **University of Twente:** *Design and Analysis of Communication Systems (DACS)* group [33]. This group's mission is to contribute to the design and implementation of dependable networked systems, as well as methods and techniques to support the design and dimensioning of such systems, such that they are dependable, in all phases of their lifecycle.

  In addition, the *Formal Methods and Tools (FMT)* group [42] is concerned with the development of formal theories of concurrency, design methodologies for distributed systems and correctness assessment using verification and validation techniques.
- **Uppsala University:** *Embedded Systems* group [37]. Aims towards scalable and precise techniques for timing analysis and correctness verification of embedded systems.
- **VERIMAG:** *Timed and Hybrid Systems* group [109]. This group is interested in all aspects of system design, ranging from theoretical foundations, via design techniques, down to implementation. In terms of models, the group has a particular focus on timed and hybrid systems.

References

[1] Int. Standard IEC 60812. Analysis techniques for system reliability procedure for failure mode and effects analysis (FMEA). Int. Electrotechnical Commission, 2nd edition, 2006.

[2] A. Abdeddaïm, E. Asarin, and O. Maler. Scheduling with timed automata. *Theoretical Computer Science*, 354(2):272–300, 2006.

[3] AESOP - Performance Analysis research group in Computing at Imperial College London. aesop.doc.ic.ac.uk, June 2014.

[4] H. Aljazzar, M. Fischer, L. Grunske, M. Kuntz, F. Leitner, and S. Leue. Safety analysis of an airbag system using probabilistic FMEA and probabilistic counterexamples. In *Proc. 6th Int. Conf. Quantitative Evaluation of Systems (QEST'09)*, 2009.

[5] H. Aljazzar and S. Leue. Debugging of dependability models using interactive visualization of counterexamples. In *Proc. 5th Int. Conf. Quantitative Evaluation of Systems (QEST'08)*, pages 189–198. IEEE CS Press, 2008.

[6] R. Alur, C. Courcoubetis, and D. Dill. Model checking in dense real time. *Information and Computation*, 104(1):2–34, 1993.

[7] R. Alur and D. Dill. Automata for modeling real-time systems. In M. Paterson, editor, *Proc. 17th Int. Colloq. Automata, Languages and Programming (ICALP'90)*, volume 443 of *LNCS*, pages 322–335. Springer, 1990.

[8] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model-checking continuous time Markov chains. *ACM Trans. Computational Logic*, 1(1):162–170, 2000.

[9] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. Software Engineering*, 29(6):524–541, 2003.

[10] C. Baier, B. Haverkort, H. Hermanns, and J-P. Katoen. Performance evaluation and model checking join forces. *Communications of the ACM*, 53(9):76–85, 2010.

[11] C. Baier and J-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[12] S. Basagiannis, P. Katsaros, A. Pombortsis, and N. Alexiou. Probabilistic model checking for the quantification of DoS security threats. *Computers & Security*, 28(6):450–465, 2009.

[13] G. Behrmann, A. David, and K. Larsen. A tutorial on UPPAAL. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *LNCS*, pages 200–236. Springer, 2004.

[14] G. Behrmann, A. Fehnker, T. Hune, K. Larsen, P. Pettersson, and J. Romijn. Efficient guiding towards cost-optimality in UPPAAL. In T. Margaria and W. Yi, editors, *Proc. 7th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, volume 2031 of *LNCS*, pages 174–188. Springer, 2001.

[15] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *LNCS*, pages 87–124. Springer, 2004.

[16] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. Thiagarajan, editor, *Proc. 15th Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS'95)*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.

[17] T. Bøgholm, H. Kragh-Hansen, P. Olsen, B. Thomsen, and K. Larsen. Model-based schedulability analysis of safety critical hard real-time Java programs. In G. Bollella and C. Locke, editors, *Proc. 6th Int. Workshop Java Technologies for Real-time and Embedded Systems (JTRES 2008)*, volume 343 of *ACM Int. Conf. Proceeding Series*, pages 106–114, 2008.

[18] T. Bogholm, B. Thomsen, K. Larsen, and A. Mycroft. Schedulability analysis abstractions for safety critical Java. In *IEEE 15th Int. Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, pages 71–78, 2012.

[19] P. Bouyer, U. Fahrenberg, K. Larsen, and N. Markey. Quantitative analysis of real-time systems using priced timed automata. *Communications of the ACM*, 54(9):78–87, 2011.

[20] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: a model-checking tool for real-time systems. In *Proc. CAV'98*, Vancouver, Canada, June 1998. Springer.

[21] M. Bozzano, A. Cimatti, J.-P. Katoen, V.-Y. Nguyen, T. Noll, and M. Roveri. The COMPASS approach: correctness, modelling and performability of aerospace systems. In *Proc. 28th Int. Conf. Computer Safety, Reliability and Security (SAFECOMP'09)*, volume 5775 of *LNCS*, pages 173–186. Springer, 2009.

[22] M. Bozzano, A. Cimatti, J.-P. Katoen, V.-Y. Nguyen, T. Noll, M. Roveri, and R. Wimmer. A model checker for AADL. In *Proc. 22nd International Conference on Computer Aided Verification (CAV'10)*, volume 6174 of *LNCS*, pages 562–565. Springer, 2010.

[23] CADP Home page. www.inrialpes.fr/vasy/cadp/, June 2014.

[24] F. Cassez, J. Jessen, K. Larsen, J.-F. Raskin, and P.-A. Reynier. Automatic synthesis of robust and optimal controllers: an industrial case study. In R. Majumdar and P. Tabuada, editors, *Proc. 12th Int. Conf. Hybrid Systems: Computation and Control (HSCC 2009)*, volume 5469 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2009.

[25] T Chen, M. Diciolla, M. Kwiatkowska, and A. Mereacre. Quantitative verification of implantable cardiac pacemakers. In *In Proc. 33rd IEEE Real-Time Systems Symposium (RTSS'12)*, pages 263–272. IEEE, 2012.

[26] A. Ciancone, M. Drago, A. Filieri, V. Grassi, H. Koziolek, and R. Mirandola. The KlaperSuite framework for model-driven reliability analysis of component-based systems. *Software & Systems Modeling*, 2013. To appear.

[27] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 2000.

[28] A. David and W. Yi. Modelling and analysis of a commercial field bus protocol. In *Proc. 12th Euromicro Conf. Real-time Systems*, pages 165–172. IEEE Computer Society, 2000.

[29] Department of Computer Science - Research Theme: Automated Verification. www.cs.ox.ac.uk/research/verification/, June 2014.

[30] Dependability - Computing Science - Newcastle University. www.ncl.ac.uk/computing/research/groups/dependability/, June 2014.

[31] Dependable Systems and Software. depend.cs.uni-sb.de, June 2014.

[32] Dependable Systems Group. gsd.famaf.unc.edu.ar, June 2014.

[33] Design and Analysis of Communication Systems. www.utwente.nl/ewi/dacs/, June 2014.

[34] Dipartimento di Informatica: Performance Evaluation and System Validation. www.unito.it/unitoWAR/page/dipartimenti1/D004_en/, June 2014.

[35] Distributed and Embedded Systems (DES). www.cs.aau.dk/en/research/des/, June 2014.

[36] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker. A formal analysis of Bluetooth device discovery. *Int. Journal on Software Tools for Technology Transfer*, 8(6):621–632, 2006.

[37] Embedded Systems. www.it.uu.se/research/group/darts, June 2014.

[38] M-A. Esteve, J-P. Katoen, V. Nguyen, B. Postma, and Y. Yushtein. Formal correctness, safety, dependability, and performance analysis of a satellite. In *Proc. 2012 Int. Conf. Software Engineering*, ICSE 2012, pages 1022–1031. IEEE Press, 2012.

[39] European Joint Conferences on Theory and Practice of Software. www.etaps.org, June 2014.

[40] A. Fehnker, R. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. L. Tan. Automated analysis of AODV using UPPAAL. In *Proc. 18th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12)*, volume 7214 of *LNCS*, pages 173–187. Springer, 2012.

[41] V. Fernández, M.-J. García-Martínez, L. Hernández-Encinas, and A. Martiín. Formal verification of the security of a free-space quantum key distribution system. In *Proc. 2011 World Congress in Computer Science, Computer Engineering, and Applied Computing (WORLDCOMP'11), 2011 International Conference on Security and Management (SAM'11)*, 2011.

[42] FMT - Formal Methods and Tools. fmt.cs.utwente.nl, June 2014.

[43] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems (SFM'11)*, volume 6659 of *LNCS*, pages 53–113. Springer, 2011.

[44] Formal Methods Research in the Computer Science Department at Carnegie Mellon. www.csd.cs.cmu.edu/research/areas/formalmethods/, June 2014.

[45] G. Gardey, J. Mullins, and O. Roux. Non-interference control synthesis for security timed automata. In *Proc. Int. Workshop Security and Concurrency (SecCo'2005)*, volume 180 of *ENTCS*, pages 35–53, 2007.

[46] B. Gebremichael, F. Vaandrager, and M. Zhang. Analysis of the zeroconf protocol using UPPAAL. In *Proc. 6th ACM & IEEE International Conference on Embedded Software (EMSOFT'06)*, pages 242–251. ACM, 2006.

[47] S. Gokhale. Architecture-based software reliability analysis: Overview and limitations. *IEEE Trans. Dependable and Secure Computing*, 4(1):32–40, 2007.

[48] K. Goševa-Popstojanova and K. Trivedi. Architecture-based approach to reliability assessment of software systems. *Performance Evaluation*, 45(2–3):179–204, 2001.

[49] L. Grunske. Specification patterns for probabilistic quality properties. In *Proc. 30th Int. Conf. Software Engineering (ICSE'08)*, pages 31–40. ACM, 2008.

[50] L. Grunske, R. Colvin, and K. Winter. Probabilistic model-checking support for FMEA. In *Proc. 4th Int. Conf. Quantitative Evaluation of Systems (QEST'07)*, pages 119–128. IEEE Press, 2007.

[51] E. M. Hahn, A. Hartmanns, H. Hermanns, and J-P. Katoen. A compositional modelling and analysis framework for stochastic hybrid systems. *Formal Methods in System Design*, 43(2):191–232, 2012.

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

[53] J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 319(3):239–257, 2008.

[54] Henzinger Group. pub.ist.ac.at/group_henzinger/, June 2014.

[55] Hybrid Systems: Computation and Control. www.hscc-conference.org, June 2014.

[56] Hybride Systeme. hs.informatik.uni-oldenburg.de, June 2014.

[57] INRIA - Inventors for the digital world. www.inria.fr/en/, June 2014.

[58] Institute for Theoretical Computer Science. www.muni.cz/fi/335200/people/, June 2014.

[59] International Conference on Computer Aided Verification. cavconference.org, June 2014.

[60] International Conference on Quantitative Evaluation of SysTems (QEST). www.qest.org, June 2014.

[61] IST Austria: Chatterjee Group. ist.ac.at/research/research-groups/chatterjee-group/, June 2014.

[62] H. Jensen, J. Kim, K. Larsen, and A. Skou. Modelling and analysis of a collision avoidance protocol using SPIN and UPPAAL. In *Proc. 2nd Int. Workshop on the SPIN Verification System*, volume 32 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 33–50, 1996.

[63] J-P. Katoen. Model checking meets probability: A gentle introduction. *NATO Science for Peace and Security Series - D: Information and Communication Security*, 34: Engineering Dependable Software Systems:1–29, 2013.

[64] J.-P. Katoen, I. Zapreev, E. M. Hahn, H. Hermanns, and D. Jansen. The ins and outs of the probabilistic model checker MRMC. In *Proc. 6th International Conference on Quantitative Evaluation of Systems (QEST'09)*, pages 167–176. IEEE CS Press, 2009.

[65] S. Kikuchi and Y. Matsumoto. Performance modeling of concurrent live migration operations in cloud computing systems using PRISM probabilistic model checker. In *Proc. 4th Int. Conf. Cloud Computing (IEEE Cloud 2011)*, 2011.

[66] S. Konrad and B. Cheng. Real-time specification patterns. In *Proc. 27th Int. Conf. Software Engineering (ICSE'05)*, pages 372–381. ACM, 2008.

[67] H. Koziolek, B. Schlich, and C. Bilich. A large-scale industrial case study on architecture-based software reliability analysis. In *Proc. 21st IEEE Int. Symp. Software Reliability Engineering (ISSRE'10)*, pages 279–288. IEEE Computer Society, 2010.

[68] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.

[69] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd Int. Conf. Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[70] M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 29:33–78, 2006.

[71] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In H. Hermanns and R. Segala, editors, *Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)*, volume 2399 of *LNCS*, pages 169–187. Springer, 2002.

[72] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. *Formal Aspects of Computing*, 14(3):295–318, 2003.

[73] Laboratoire Spécification et Vérification. `www.lsv.ens-cachan.fr`, June 2014.

[74] Laboratory for Foundations of Computer Science. `www.inf.ed.ac.uk/research/lfcs/`, June 2014.

[75] M. Lahijanian, J. Wasniewski, S. B. Andersson, and C. Belta. Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In *Proc. 2010 IEEE International Conference on Robotics and Automation*, pages 3227–3232, 2010.

[76] M. Lakin, D. Parker, L. Cardelli, M. Kwiatkowska, and A. Phillips. Design and analysis of DNA strand displacement devices using probabilistic model checking. *Journal of the Royal Society Interface*, 9(72):1470–1485, 2012.

[77] K. Larsen, P. Pettersson, and W. Yi. Model-checking for real-time systems. In *Proc. 10th Int. Symp. Fundamentals of Computation Theory (FCT'95)*, volume 965 of *LNCS*, pages 62–88. Springer, 1995.

[78] K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.

[79] A. Legay, B. Delahaye, and S. Bensalem. Statistical model checking: An overview. In H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. Pace, G. Roşu, O. Sokolsky, and N. Tillmann, editors, *Proc. 1st. Int. Conf. Runtime Verification (RV 2010)*, volume 6418 of *LNCS*, pages 122–135. Springer, 2010.

[80] Florian Leitner-Fischer and Stefan Leue. QuantUM: Quantitative safety analysis of UML models. In *Proc. 9th Workshop on Quantitative Aspects of Programming Languages (QAPL'11)*, 2011.

[81] M. Lindahl, P. Pettersson, and W. Yi. Formal design and analysis of a gear-box controller. In *Proc. 4th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *LNCS*, pages 281–297. Springer, 1998.

[82] B. Littlewood and J. Verrall. A Bayesian reliability growth model for computer software. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 22(3):332–346, 1973.

[83] Markov Reward Model Checker. www.mrmc-tool.org, June 2014.

[84] S. Merz and N. Navet, editors. *Modeling and Verification of Real-Time Systems*. Wiley, 2008.

[85] M. Mikučionis, K. Larsen, J. Rasmussen, B. Nielsen, A. Skou, S. Palm, J. Pedersen, and P. Hougaard. Schedulability analysis using Uppaal: Herschel-Planck case study. In *Proc. 4th Int. Conf. Leveraging Applications of Formal Methods, Verification and Validation - Part II*, pages 175–190. Springer, 2010.

[86] The Möbius Tool. www.mobius.illinois.edu, June 2014.

[87] Modest Toolset. www.modestchecker.net, June 2014.

[88] S. Mondal and S. Sural. Security analysis of temporal-RBAC using timed automata. In *4th Int. Conf. Information Assurance and Security (ISIAS'08)*, pages 37–40, 2008.

[89] G. Norman, D. Parker, M. Kwiatkowska, S. Shukla, and R. Gupta. Using probabilistic model checking for dynamic power management. *Formal Aspects of Computing*, 17(2):160–176, 2005.

[90] G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. *Journal of Computer Security*, 14(6):561–589, 2006.

[91] Gethin Norman, David Parker, and Jeremy Sproston. Model checking for probabilistic timed automata. *Formal Methods in System Design*, 43(2):164–190, 2013.

[92] S. Palm. Herschel-Planck ACC ASW: sizing, timing and schedulability analysis. Technical report, Terma A/S, 2006.

[93] PEPA - Performance Evaluation Process Algebra. www.dcs.ed.ac.uk/pepa/, June 2014.

[94] PRECISE. precise.seas.upenn.edu, June 2014.

[95] PRISM – Probabilistic Symbolic Model Checker. www.prismmodelchecker.org, June 2014.

[96] Quantitative Analysis and Verification. qav.cs.ox.ac.uk, June 2014.

[97] A. Ravn, J. Srba, and S. Vighio. Modelling and verification of web services business activity protocol. In *Proc. 17th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11)*, volume 6605 of *LNCS*, pages 357–371. Springer, 2011.

[98] Reactive Systems Group. www.react.uni-saarland.de, June 2014.

[99] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems, P. Panangaden and F. van Breugel (eds.)*, volume 23 of *CRM Monograph Series*. American Mathematical Society, 2004.

[100] V. Shmatikov. Probabilistic model checking of an anonymity system. *Journal of Computer Security*, 12(3/4):355–377, 2004.

[101] Simulink - Simulation and Model-Based Design - MathWorks United Kingdom. www.mathworks.co.uk/products/simulink/, June 2014.

[102] Software Modeling and Verification. moves.rwth-aachen.de, June 2014.

[103] Specification and Development of Probabilistic Systems. www.cse.unsw.edu.au/~carrollm/probs/, June 2014.

[104] G. Steel. Formal analysis of PIN block attacks. *Theoretical Computer Science*, 367(1-2):257–270, 2006.

[105] TUD - Chair of Algebraic and Logical Foundations of Computer Science. www.inf.tu-dresden.de/index.php?node_id=1438&ln=en, June 2014.

[106] TUM - Chair VII - Foundations of Software Reliability and Theoretical Computer Science. www7.in.tum.de/home/index.php, June 2014.

[107] UPPAAL. www.uppaal.org, June 2014.

[108] M. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proc. 26th Annual Symp. Foundations of Computer Science (FOCS'85)*, pages 327–338. IEEE Computer Society Press, 1985.

[109] VERIMAG: Tempo. www-verimag.imag.fr/Tempo,32.html, June 2014.

Papers in the Series:

1. Managing Risk in the Modern World

Applications of Bayesian Networks

Norman Fenton and Martin Neil

2. The GPU Computing Revolution

From Multi-Core CPUs to Many-Core Graphics Processors

Simon McIntosh-Smith

3. Problem Solving for the 21st Century

Efficient Solvers for Satisfiability Modulo Theories

Clark Barrett, Daniel Kroening and Tom Melham

4. Quantitative Verification

Formal Guarantees for Timeliness, Reliability and Performance

Gethin Norman and David Parker

Quantitative Verification

Formal Guarantees for Timeliness, Reliability and Performance

A Knowledge Transfer Report from the London Mathematical Society and the Smith Institute
for Industrial Mathematics and System Engineering

by Gethin Norman and David Parker

The London Mathematical Society (LMS) is the UK's learned society for mathematics. Founded in 1865 for the promotion and extension of mathematical knowledge, the Society is concerned with all branches of mathematics and its applications. It is an independent and self-financing charity, with a membership of around 2,300 drawn from all parts of the UK and overseas. Its principal activities are the organisation of meetings and conferences, the publication of periodicals and books, the provision of financial support for mathematical activities, and the contribution to public debates on issues related to mathematics research and education. It works collaboratively with other mathematical bodies worldwide. It is the UK's adhering body to the International Mathematical Union and is a member of the Council for the Mathematical Sciences, which also comprises the Institute of Mathematics and its Applications, the Royal Statistical Society, the Operational Research Society and the Edinburgh Mathematical Society.

www.lms.ac.uk

The Smith Institute for Industrial Mathematics and System Engineering is a leader in the UK for harnessing mathematics as an engine of business innovation. Established as an independent organisation in 1997, it adopts a systems approach, connecting modelling, data, algorithms and implementation. It works with an extensive network of collaborators in industry, government and the university research base to improve products, services and processes. The Smith Institute represents the UK internationally in the field of industrial mathematics and works closely with UK Research Councils and other funding agencies to create new and effective approaches to business-university interaction. It places an emphasis on raising awareness outside the mathematical community of the benefits of adopting a mathematical way of thinking, often transferring ideas across application domains to create fresh insights that deliver real value.

www.smithinst.co.uk

The LMS-Smith Knowledge Transfer Reports are an initiative that is coordinated jointly by the Smith Institute and the Computer Science Committee of the LMS. The reports are being produced as an occasional series, each one addressing an area where mathematics and computing have come together to provide significant new capability that is on the cusp of mainstream industrial uptake. They are written by senior researchers in each chosen area, for a mixed audience in business and government. The reports are designed to raise awareness among managers and decision-makers of new tools and techniques, in a format that allows them to assess rapidly the potential for exploitation in their own fields, alongside information about potential collaborators and suppliers.

LONDON
MATHEMATICAL
SOCIETY

Smith institute
for industrial mathematics and system engineering